Funktions- / Strukturorientierte Pflanzenmodellierung in E-Learning-Szenarien

Dissertationsschrift

zur Erlangung des Doktorgrades der Fakultät für Forstwissenschaften und Waldökologie der Georg-August-Universität Göttingen

vorgelegt von Dirk Lanwert
geboren in Osnabrück
Göttingen 2007

D7

1. Gutachter: Prof. Dr. Dr. h.c. B. Sloboda[1]
2. Gutachter: Prof. Dr. W. Kurth[2]
3. Gutachter: Prof. Dr. J. Nagel[3]

Tag der mündlichen Prüfung: 8. Juni 2007

Diese Arbeit wurde elektronisch publiziert unter:
http://resolver.sub.uni-goettingen.de/purl/?webdoc-1692

[1] Institut für Forstliche Biometrie und Informatik der Georg-August Universität Göttingen

[2] Lehrstuhl für Graphische Systeme der Brandenburgischen Technischen Universität Cottbus

[3] Abteilung Waldwachstum der Nordwestdeutschen Forstlichen Versuchsanstalt

Kurzzusammenfassung

Lanwert, D. 2007: Funktions- / Strukturorientierte Pflanzenmodellierung in E-Learning-Szenarien. Fakultät für Forstwissenschaften und Waldökologie der Georg-August Universität Göttingen, Dissertation

Diese Arbeit behandelt den Aufbau eines E-Learning-Systems, welches den Ansatz der Funktions-/Strukturmodellierung für die Verwendung in der universitären Lehre zur Waldwachstumsmodellierung benutzt. Das E-Learning-System verwendet „Virtuelle Realität“ – in Form von begehbaren, interaktiven virtuellen Waldbeständen – kombiniert mit Waldwachstumssimulatoren – zur Prognoseberechnung möglicher Waldentwicklungen – als explorative Lernumgebung zur Erforschung der Wachstumsmodellierung durch die Studierenden der Forstwissenschaften und Waldökologie.

Das System integriert verschiedene Programme in eine offene, internetbasierte Architektur. Grundlage des Systems ist ein spezielles Beschreibungsformat für Waldbestände auf Basis der Virtual Reality Modelling Language. Die Visualisierung der Bestandesszenen übernimmt der Virtual Forester Client – ein eigens für die forstliche Lehre entworfener interaktiver VRML-Viewer. Er bildet die graphische Schnittstelle für die Benutzerin und ermöglicht das interaktive Behandeln des Bestandes (Bäume fällen, Bäume auszeichnen, Baumdaten abfragen etc.).

Die Verbindung verschiedener Clients für das gemeinsame Arbeiten in einer Bestandesszene realisiert der Elan Sim Server.

Als Wachstumssimulator wurde die Growth Grammar-related Interactive Modelling Platform (GroIMP) [KNIEMEYER] in das E-Lerning-System mit eingebunden. Die Modellkomponente wird auf Basis der Relationalen Wachstumsgrammatiken [KURTH] in der regelbasierten Sprache XL umgesetzt.

Der Modellansatz umfasst ein einfaches, räumliches, biomassebezogenes, ökophysiologisches Modell [SLOBODA & PFREUNDT] mit Berechnungen der Photosynthese – in Abhängigkeit der beschattenden Biomasse –, der Respiration und des Wachstums in Jahresschritten für die 5 Baumkompartimente Nadeln, Äste, Fein- und Grobwurzeln sowie den Stamm. Das Modell ist initial mit Literaturwerten für Fichte und Kiefer parametrisiert, kann aber für andere Nadelbaumarten erweitert werden.

Primäre Zielsetzung des E-Learning-Systems ist der Aufbau einer vorbereiteten Arbeitsumgebung für das explorative Lernen der Studierenden. Die vollständige Offenlegung der Modellkomponente und die regelbasierte Sprache XL bieten den Studierenden ein effizientes Umfeld, um von der Analyse des vorliegenden Modells über dessen schrittweise Veränderung bis hin zur selbstständigen Konstruktion eigener Modellansätze zu gelangen.

Die offene, internetbasierte Architektur des Systems und der einfache, flexible Aufbau der Bestandesbeschreibung erlaubt die Einbindung weiterer Programme – z. B. alternative Wachstumssimulatoren wie TreeGrOSS [NAGEL] und SIBYLA [FABRIKA]– und somit die Erweiterung der möglichen Anwendungsszenarien.

1 Motivation

1.1 Allgemeine Motivation

1.1.1 Lehre von forstlichen Eingriffen

Waldbau kann man durchaus als eine der Königsdisziplinen im Studium der Forstwissenschaften und Waldökologie betrachten. In ihr fließen die Kenntnisse der verschiedenen Fachdisziplinen zusammen und vereinen so die verschiedenen Blickwinkel von der Ökologie bis zur Ökonomie in konkreten Maßnahmen der Waldbehandlung. Zugleich stellen die waldbaulichen Planungen die Lehre aber auch vor die größten Probleme. Die Beziehung von Ursache und Wirkung einer Durchforstungsmaßnahme umfasst eine Zeitspanne, die sich einer direkten Beobachtung durch die Studierenden entzieht. Wirtschaftliche Lebenszyklen der Bestände von 80 bis über 150 Jahre lassen sich im Studium nur schwer erfahrbar machen.

Um jenseits der Anschauung die zukünftige Entwicklung eines Bestandes zu verdeutlichen, versorgt uns die Waldwachstumslehre mit Modellannahmen zum Wachstum der Bäume. Diese Modellannahmen sind aus der Forschung abgeleitet und basieren in der Regel auf in aufwändigen Langzeiterhebungen gewonnenen, empirischen Daten. Sie beschreiben für uns in forstlichen Kenngrößen das Entwicklungspotenzial von Beständen in Abhängigkeit von standort-, baumarten- und behandlungsspezifischen Eingangsparametern. So gelingt es, die lange Zeitspanne zwischen Ursache und Wirkung mittels mathematischer Modelle zu überbrücken. Doch wohnen gerade dieser Überbrückung in Bezug auf die Lehre – aber auch für die Praxis – besondere Schwierigkeiten inne. Lange Zeit war die Ertragstafel [Schwappach 1889, Wiedemann 1336] die allgemein gültige Modellannahme zur Entwicklung der deutschen Forste. In tabellarischer Form aufgeführte, nach Bonität und Durchforstungsstärke aufgeteilte Kenngrößen wie Vorrat und Zuwachs für jede Altersstufe gaben eine übersichtliche Vorstellung von der Zukunft eines Bestandes – eine Vorstellung, die insbesondere wegen ihrer Übersichtlichkeit in der Lehre gut zu vermitteln war.

Leider genügt die Ertragstafel nicht mehr, um den aktuellen Wissenstand in der Waldwachstumslehre zu repräsentieren. Sowohl neue Bestandesformen und andere Behandlungskonzepte als auch veränderte ökologische Rahmenbedingungen und aktuelle Forschungsergebnisse definieren den Bedarf nach weitergehenden Modellannahmen, deren Dynamik und Komplexität eine abstrakte, mathematische Darstellung notwendig macht.

Und genau dieser Abstraktionsgrad erhöht die Anforderung an die Studierenden für die Anwendungen solcher Modellannahmen.

1.1.2 Wachstumssimulatoren

Die modernen Werkzeuge zur Vorhersage der zukünftigen Bestandesentwicklung sind Wachstumssimulatoren. Wachstumssimulatoren sind Computerprogramme, die ausgehend von beschreibenden Bestandesdaten – mittels parametrisierter Wachstumsfunktionen – eine Prognose der Bestandesentwicklung berechnen und in aussagekräftigen Kenngrößen darstellen. Diese Computerprogramme haben zahlreiche Vorteile: So können leicht verschiedene Wachstumsfunktionen und Parametersätze verwendet werden, um die Prognosen an veränderte Bedingungen anzupassen. Hinzu kommt die

Möglichkeit, mittels stochastischer Komponenten und Verteilungsfunktionen für die Parametersätze eine natürliche Variabilität im Wachstum der Bäume abzubilden und über verschiedene Simulationsläufe alternative Prognosen zu betrachten. Besonders interessant ist auch die Möglichkeit, weitere Kenngrößen wie Baumartenverteilung und Biodiversitätsindizes zu berechnen, die Kenngrößen detaillierter und übersichtlicher darzustellen und sie ggf. mit anderen Programmen weiterzubearbeiten. Wachstumssimulatoren sind daher oft Bestandteil einer umfangreicheren Softwaresuite zur forstlichen Planung.

Eine besondere Kategorie von Vorteilen ergibt sich mit der Verwendung eines einzelbaumorientierten Wachstumssimulators. Hierzu wird über einen Strukturgenerator – ein weiteres Computerprogramm – anhand der Bestandeskenngrößen ein virtueller räumlicher Bestand erzeugt. Dieser virtuelle Bestand ist nun Basis der Prognoseberechnung. So eröffnet sich die Möglichkeit, die räumliche Struktur des Bestandes in die Simulation mit einzubeziehen und ggf. über manuelle Eingriffe zu manipulieren. Konkurrenzsituationen von Bäumen und Baumarten können so dargestellt werden. Dies ist eine wichtige Grundvoraussetzung, um Wachstumssimulatoren im Lernprozess von Bestandesbehandlungskonzepten einzubinden.

Geht man auf dem Weg der Suche nach dem Verständnis des Waldwachstums noch eine Ebene tiefer, stößt man auf die Klasse der Ökophysiologischen Modelle [siehe Pretzsch 2002, S. 82]. Diese versuchen, die Prozesse innerhalb des Baumes in die Wachstumssimulation mit einzubeziehen, und können je nach Komplexitätsgrad auf detaillierten Zeitskalen bis in den Sekundenbereich arbeiten. Die Zielsetzung solch detaillierter Modelle ist überwiegend die Erklärung von Zusammenhängen in den Ökosystemen.

1.1.3 Funktions- / Strukturmodellierung

Ein guter Weg im Grenzbereich von Einzelbaummodellen und Ökophysiologischen Modellen ist der aktuelle Ansatz der Funktions- / Strukturmodellierung von Pflanzen (FSPM). "The FSMs [Functional / Structural Models: Anmerkung des Autors] contain descriptions of metabolic (physiological) processes that are combined in the presentation of the 3D structure of the tree." [Sievänen et al. 2000]. Die FSPM verknüpfen hierbei Methoden der 3-dimensionalen Darstellung von Pflanzen mit physiologischen Modellen und Methoden der Informatik und Mathematik. Das Ziel der FSPM ist es, durch die kombinierte Modellierung der grundlegenden räumlichen Struktur der Pflanzen mit den daran gekoppelten Prozessen die Funktion und das Wachstum der Pflanze besser zu verstehen.

Die Kombination von Struktur und Funktion kann dabei auf allen Skalenebenen eines Ökosystems stattfinden – von den Genen bis zur Bestandesebene.

Die hierfür speziell entwickelten Methoden, Sprachen und Werkzeuge erlauben die einfache und effiziente Entwicklung von verständlichen Modellen für Fragestellungen sowohl auf den unterschiedlichsten Skalenebenen als auch skalenübergreifend.

Mit dem skalenübergreifenden Ansatz der Funktions-/Strukturmodellierung ist es also möglich, mit einfachen ökophysiologischen Ansätzen die einzelbaumorientierte Wachstumssimulation um physiologische Betrachtungen zu erweitern. So können stärker die Wirkungsmechanismen des Waldwachstums beleuchtet werden, ohne zu detaillierte Betrachtungsebenen beschreiten zu müssen.

1.1.4 E-Learning

Es stellt sich nun die Frage, welche Möglichkeiten sich aus der Entwicklung und Verwendung von einzelbaumorientierten Wachstumssimulatoren für die Ausbildung der Studierenden ergeben.

Jedwede Maßnahme zur Behandlung eines Bestandes basiert auf der Modellannahme des Handelnden zur zukünftigen Entwicklung dieses Bestandes. Es ist das Ziel des Studiums der Forstwissenschaften und Waldökologie, das Wissen über solche Modelle zu vermitteln und die Fähigkeit zu formen, dieses Wissen in der Praxis anzuwenden. Neben planerischen Größen auf Bestandesebene und höher, umfasst dies auch die am Einzelbaum orientierte Durchforstungsmaßnahme. Wird in der Ursache-Wirkungs-Beziehung von Durchforstung und Waldentwicklung die Ursache in Exkursionen gründlich und anschaulich unterrichtet, bleibt die Wirkung und der Wirkungsmechanismus aufgrund der zeitlichen Dimensionen des Baumwachstums im Abstrakten. Dies gilt insbesondere für Einzelmaßnahmen und deren Auswirkungen.

Die rasant wachsenden Möglichkeiten der PC-Hard- und Software zur räumlichen Visualisierung in Verbindung mit dem Reifegrad und der Verfügbarkeit von Wachstumssimulatoren und forstlichen Planungstools lassen es sinnvoll erscheinen, diesen Lernprozess der Studierenden mit einer E-Learning-Komponente zu unterstützen.

Dabei wird aber nicht das Ziel verfolgt, den Studierenden durch verbesserte Darstellungsmöglichkeiten das Gefühl zu vermitteln, die Wachstumsmodelle entsprächen der wirklichen Bestandesentwicklung. Vielmehr geht es darum, ein besseres Verständnis der Wachstumsmodelle und der zugrundeliegenden Annahmen und Theorien zu erreichen, und damit um eine bessere Einschätzung ihrer Beziehung zur Realität.

1.1.5 Gründe für E-Learning

Seit wenigen Jahren wird an zahlreichen deutschen Hochschulen mit verschiedenen E-Learning-Ansätzen experimentiert. Zahlreiche diese Initiativen befinden sich an der Schwelle von der Experimentalphase zur Übernahme in den regulären Betrieb. So ist auch diese Arbeit im Rahmen einer Förderinitiative des Landes Niedersachsen zu E-Learning, dem E-Learning Academic Network Niedersachsen [ELAN 2005 & 2006], entstanden. Das ELAN-Netzwerk besteht aus 3 Pilotvorhaben mit jeweils 2 kooperierenden Universitäten und befindet sich am Ende der 2. Förderphase, mit dem Ziel, die gewonnen Erkenntnisse und Ansätze an den Universitäten nachhaltig zu verankern.

Die Gründe und Zielsetzungen für die Initiativen, E-Learning als Teil des methodischen Repertoires in der Lehre der deutschen Hochschulen zu etablieren, sind vielfältig. So wird oft die Erwartung genannt, für die Universität eine Kostenersparnis zu erzielen oder über die Vermarktung von kostenpflichtigen Materialien Einnahmequellen zu generieren. Weitere Gründe sind die Wünsche nach Zeitersparnis für die Dozentinnen, nach einem modernen Image der Einrichtungen bis hin zur Attraktivitätssteigerung des Lehrangebots importierter Kurse. Es gibt noch zahlreiche andere strategische, hochschulpolitische oder finanzielle Betrachtungswinkel, die zu evaluieren sich lohnen würde. Im Folgenden sollen aber ausschließlich die E-Learning-Aspekte betrachtet werden, die sich mit der Fragestellung befassen, ob sich mit dem diskutierten Konzept die Wissensvermittlung an der Fakultät für Forstwissenschaften und Waldökologie verbessern lässt.

1.1.6 Was will diese Arbeit

Diese Arbeit zeigt das Konzept, wie Funktion-/Strukturmodelle von Pflanzen in einer Lernsoftware integriert und in E-Learning-Szenarien verwendet werden können, um im Rahmen der Ausbildung ein verbessertes Verständnis der Lernenden von Waldwachstumsmodellen und der Waldwachstumsmodellierung zu erreichen. Hierzu wird das Zusammenspiel der verschiedenen Softwarekomponenten in der Lernsoftware erläutert und am Beispiel eines ökophysiologischen Einzelbaummodells die mögliche Umsetzung aufgezeigt.

1.2 Didaktische Grundannahmen

"*In jeder Lernsoftware schlägt sich ein theoretisches Lernmodell nieder. Egal ob dieser theoretische Ansatz nun von den Autorinnen auch tatsächlich expliziert worden ist oder nicht, spiegelt die Lernsoftware – angefangen vom behandelten Thema über den Aufbau bzw. die Struktur des Softwarepaketes bis hin zur Benutzeroberfläche des Lernprogramms – ein pädagogisches und didaktisches Modell wider, das in ihr implementiert wurde.*" [Baumgartner 1997, S. 244]. Um die Frage nach der zugrunde liegenden Lerntheorie für den Einsatz von Funktions- und Strukturmodellen im Studium der Forstwissenschaften und Waldökologie beantworten zu können, bemühen wir uns um eine kurze Charakterisierung der "drei einflussreichsten Theoriesysteme dieses Jahrhunderts – Behaviorismus, Kognitivismus und Konstruktivismus" [Baumgartner & Payr 1997]. Baumgartner und Payr ordnen den Zusammenhang von Lernparadigmen und Softwaretopologie wie in Tabelle 1 dargestellt:

Kategorie	*Behaviorismus*	*Kognitivismus*	*Konstruktivismus*
Hirn ist ein	passiver Behälter	informationsverarbeitendes "Gerät"	informationell geschlossenes System
Wissen wird	abgelagert	verarbeitet	konstruiert
Wissen ist	eine korrekte Input -Output-Relation	ein adäquater interner Verarbeitungsprozess	mit einer Situation operieren zu können
Lernziele	richtige Antworten	richtige Methoden zur Antwortfindung	komplexe Situationen bewältigen
Paradigma	Stimulus-Response	Problemlösung	Konstruktion
Strategie	lehren	beobachten und helfen	Kooperation
Lehrer ist	Autorität	Tutor	Coach, (Spieler)Trainer
Feedback	extern vorgegeben	extern modelliert	intern modelliert
Interaktion	starr vorgegeben	dynamisch in Abhängigkeit des externen Lernmodells	selbstreferenziell, zirkulär, strukturdeterminiert (autonom)
Programmmerkmale	starrer Ablauf, quantitative Zeit- und Antwortstatistik	dynamisch gesteuerter Ablauf, vorgegebene Problemstellung, Antwortanalyse	dynamisch, komplex vernetzte Systeme, keine vorgegebene Problemstellung
Software-Paradigma	Lernmaschine	Künstliche Intelligenz	sozio-technische Umgebungen
"idealer" Softwaretypus	Tutorielle Systeme, Drill & Practice	adaptive Systeme, IST (Intelligente Tutorielle Systeme)	Simulationen, Mikrowelten, Hypermedia

Tabelle 1: Vergleich der drei großen Lerntheorien: Behaviorismus, Kognitivismus und Konstruktivismus [Zusammenstellung von Gausche 2003 nach Baumgartner & Payr (94, 110, 174)]

In dieser Aufstellung werden Simulationen und Hypermedia dem Konstruktivismus zugeordnet, Intelligente Tutorielle Systeme hingegen dem Kognitivismus. Von der Art des Lernens unterscheiden sich die beiden wie folgt: "*Die Art des Lernens, die im Kognitivismus im Mittelpunkt der Forschung steht, ist das Problemlösen: Es geht nicht mehr darum, auf gewisse Stimuli die (einzig) richtige Antwort zu produzieren, sondern weit allgemeiner darum, richtige Methoden und Verfahren zur Problemlösung zu lernen, deren Anwendung dann erst die (eine oder mehreren) richtigen Antworten generiert. Aus der Sichtweise vernetzter Systeme geht es auch nicht mehr darum, die eine richtige Antwort im Sinne einer Maximierung zu finden, sondern es können vielmehr verschiedene Verfahren zu optimalen Ergebnissen führen."* [Baumgartner & Payr 1997, S. 3].

Im Konstruktivistischen Ansatz wird Lernen "*als ein aktiver Prozess gesehen, bei dem Menschen ihr Wissen in Beziehung zu ihren früheren Erfahrungen in komplexen realen Lebenssituationen konstruieren. Im praktischen Leben sind Menschen mit einzigartigen, nicht vorhersehbaren Situationen konfrontiert, deren Probleme nicht evident sind. Im Gegensatz zum Kognitivismus steht im Konstruktivismus daher nicht das Lösen bereits existierender, bloß zu entdeckender Probleme im Vordergrund, sondern das eigenständige Generieren von Problemen*" [Baumgartner & Payr 1997, S. 4].

"*Im Konstruktivismus steht die eigene, persönliche Erfahrung im Vordergrund. Lernende sollen komplexe Situationen bewältigen und müssen dabei erst die notwendigen Aufgaben- und Problemstellungen generieren."* [Baumgartner & Payr 1997, S. 5].

Wir können daran erkennen, dass die Zuordnung der Lernsoftware zu den einzelnen Lernparadigmen nicht nur von der Art der Software und des begleitenden Lernmaterials abhängig ist, sondern insbesondere von der Aufgabenstellung, der Rolle der Dozentin und der entsprechenden Einbindung in den Lernprozess des Studierenden. Auf jeden Fall ist die in dieser Arbeit beschriebene Software angelegt für einen Ansatz des explorativen Lernens und befindet sich im Grenzbereich zwischen Kognitivismus und Konstruktivismus.

1.3 Angestrebte Ziele

Um einen sinnvollen Einsatz von Funktions- / Strukturmodellen in der Lehre zu konzipieren und deren Umsetzung zu bewerten, ist es unbedingt notwendig, im Vorfeld eine genaue Definition der Zielsetzungen vorzunehmen. Gemäß der in Kapitel 1.1.5 getroffenen Grundaussage, sich auf dem Mehrwert bei der Wissensvermittlung zu beschränken, und anderweitige strategische oder finanzielle Betrachtungen außen vor zu lassen, werden für das vorgestellte Konzept folgende didaktische Zielsetzungen definiert:

1.3.1 Funktions- / Strukturmodellierung

Die Lernenden sollen die Möglichkeiten und Grenzen der Funktions- / Strukturorientierten Modellierung für Einzelbäume und Waldbestände erfassen und verschiedene Methoden der Modellierung selbst erproben.

1.3.2 Verdeutlichung von Modellannahmen zur Walddynamik

Die Studierenden sollen sich die einer entsprechenden Wachstumsprognose zugrunde liegenden Modelle über das Baumwachstum auf Bestandesebene bewusst machen und ein verbessertes Verständnis über die Einschränkungen und Grundprämissen dieser Modelle erwerben.

1.3.3 Trainingswerkzeug zur einzelbaumorientierten Durchforstung

Die Studierenden sollen ein verbessertes Verständnis entwickeln für die Folgen von an Einzelbäumen und Baumgruppen orientierten Durchforstungsmaßnahmen (z.B. Baumeigenschaften, Konkurrenz als Selektionskriterium). Dies beinhaltet zwei Teilziele:

Verstehen des Zusammenhanges zwischen Einzelmaßnahmen und der entsprechenden Veränderung der Bestandesgrößen (z.B. Mitteldurchmesser, Vorrat, Zuwachs).

Verstehen der Folgen von Einzelbaummaßnahmen auf die konkurrierenden Bäume (z.B. auf Qualität, Lichtkonkurrenz).

1.3.4 Folgeabschätzung eines Eingriffes

Die Lernenden sollen zu einer verbesserten Einschätzung der weiter gefassten Folgen von Durchforstungseingriffen gelangen. Ziel ist es, unterschiedliche Behandlungskonzepte auf ihre Eignung in verschiedenen Ausgangssituationen beurteilen zu lernen und die Folgewirkung (z.B. auf Biodiversität, Ökonomie) abschätzen zu lernen.

1.3.5 Offene Schnittstelle für diverse Wachstumssimulatoren

Die Studierenden sollen sich die Möglichkeiten und Einschränkungen von Wachstumssimulatoren für die forstliche Arbeitspraxis im Computerexperiment erarbeiten.

1.3.6 Sekundäres Ziel

Als übergeordnetes Ziel mit indirektem – weil unspezifiziertem – Einfluss auf die Lehre wird die Möglichkeit einer allgemein verbesserten visuellen Darstellung von Modellbeständen im Rahmen weiterer forstlicher Unterrichtseinheiten angestrebt.

1.4 Abgeleitete Anforderungen an das System

Unter Berücksichtigung der didaktischen Grundannahmen werden für die Erreichung der angestrebten Ziele folgende Anforderungen an die Lernsoftware abgeleitet:

1.4.1 Einfach zu nutzen

Eine Anbindung an verschiedene Wachstumssimulatoren sowie das Ziel der Nutzung in verschiedenen Lehrszenarien und Unterrichtseinheiten definiert den Bedarf nach einer einfachen Form der Bestandesbeschreibung. Diese muss folgende Eigenschaften aufweisen:

- Andere Programme müssen sie ohne großen Aufwand (Programmieraufwand) schreiben und lesen können
- Bestandesbeschreibungen müssen mit einem Editor manuell erstellt und verändert werden können
- Die Bestandesbeschreibung muss durch einfaches Lesen für die Benutzerin interpretierbar sein

1.4.2 Visualisierung aller wesentlichen Parameter

Die Lösung zur Visualisierung der Bäume und Bestände muss alle wesentlichen morphologischen Eigenschaften darstellen können, die als entscheidungsrelevant für Durchforstungen angesehen werden. Hierzu gehören folgende Eigenschaften:

- Position im Bestand (X,Y,Z - Koordinaten)
- Höhe, Kronenansatzpunkt
- Stammdurchmesser, -länge, -form
- Kronenbreite, -länge und -form
- typische optische Erscheinungsbilder wie Rindenstruktur, Farbe, Blattstruktur, Blattdichte und Vergilbungsgrad
- weitere Qualitätsanzeiger, wie Drehwuchs, Wasserreiser, Beschädigungen

Zudem sollte das System so offen sein, dass außergewöhnliche Situationen jederzeit ohne Programmieraufwand ergänzt werden können.

1.4.3 Einfache Interaktion zur Durchführung von Durchforstungsmaßnahmen

Die Schnittstelle zur Nutzerin des Systems muss einfache graphische Möglichkeiten zur virtuellen Durchführung von Durchforstungsmaßnahmen bereitstellen. Hierzu gehören

- das Markieren von Bäumen als negativ selektiert
- das Markieren von Bäumen als positiv selektiert
- das Fällen von Bäumen

1.4.4 Variationen der Modellannahmen/Simulationswerkzeuge

Die Gesamtarchitektur der Lernsoftware sollte so offen sein, dass verschiedene Wachstumssimulatoren mit vertretbarem Aufwand in das System integriert werden können. Nach einmaligem Adaptions- und Installationsaufwand sollte dies durch die Nutzer selbstständig erfolgen können. Der Adaptionsaufwand sollte sich auf Seiten der neu zu

integrierenden Programmteile befinden, ohne die Notwendigkeit der erneuten Anpassung für bereits integrierte Simulatoren zu erzeugen.

1.4.5 Kommunikationskomponenten

Die Selbstkonstruktion von Wissen durch die Studierenden, durch das Lösen von gegebenen Problemstellungen oder durch die Entwicklungen eigener Problemstellungen, bedarf der Unterstützung durch das Lehrpersonal. Um den klassischen Vorteil des E-Learnings – das orts- und zeitunabhängige Lernen – zu ermöglichen, sind hierfür geeignete Kommunikationskomponenten Teil der Anforderungsliste. Da der Lernprozess zudem im Team denkbar ist, sollten die Kommunikationskomponenten multiuserfähig sein.

1.4.6 Möglichkeit zur Anbindung erweiterter Einzelbaumorientierung

Das Ziel der eigenen Erprobung der Funktions- / Strukturmodellierung von Baumwachstum erfordert die Anbindung einer Simulationsumgebung mit offenem Userinterface zur Programmierung von Modellen. Im Vorgriff auf die später beschriebenen Lösungen – unter Berücksichtigung der am Institut für Forstliche Biometrie und Informatik vorhandenen Expertise im Einsatz von Lindenmayersystemen – wird für diese Arbeit der Ansatz der regelbasierten Sprache XL und das Programm GroIMP [Kniemeyer 2007] gewählt. Über die offene Schnittstelle zur Anbindung anderer Wachstumssimulatoren sind zu einem späteren Zeitpunkt auch andere Ansätze möglich.

Bedingung für die Gestaltung eines sinnvollen Lernszenarios ist aber entsprechendes Lernmaterial über den Modellierungsansatz und die entsprechende Programmiersprache.

2 State of the Art

2.1 Kurze Charakterisierung von Pflanzenmodellen

Zum besseren Verständnis ordnet Kurth [siehe Kurth 1994a, Kurth et al. 2006] die vielen verschiedenen mathematischen Modelle und Simulationssysteme im Forschungsfeld des Baumwachstums und der Walddynamik in ein Dreieck (Abbildung 1). An der Spitze des Dreiecks befinden sich die aggregierten Modelle, die ganze Populationen – z. B. Waldbestände – mittels statistischer Methoden behandeln. Klassische Vertreter dieser Kategorie sind Ertragsfunktionen, die den Bestand über wenige Kenngrößen – wie Grundfläche und Häufigkeitsverteilungen – darstellen [Prodan 1965, Kramer 1984, Wenk 1990] und Ertragstafeln [Schober 1987, Assmann & Franz 1963, Wenk et al. 1982].

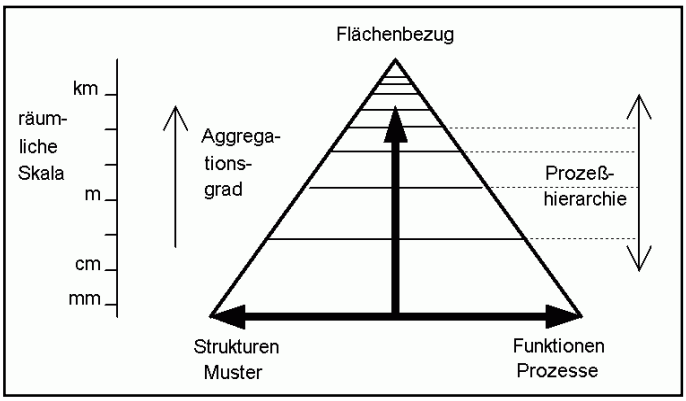


Abbildung 1: Das Dreieck der Pflanzenmodelle [Kurth et al. 2006].

Die rechte untere Ecke wird von den Prozessmodellen besetzt, die darauf zielen, unbeachtet jeder Struktur der Pflanze, die biologischen Prozesse und Funktionen zu modellieren. Dieser Typ von Modell wird vielfach durch Differentialgleichungen ausgedrückt. Als typische Vertreter dieser Kategorie nennt TREEDYN von Bossel [1996] sowie die Einzelbaummodelle für Buche von Stickan et al. [1994] und von Hoffmann [1995]. Die Kante zwischen den aggregierten Modellen und den Prozessmodellen wird abwärts durch eine zunehmend detaillierte Auflösung der Prozesse gebildet.

Die linke untere Ecke hingegen wird von den morphologischen Modellen besetzt, welche die Entwicklung der räumlichen Struktur der Pflanze – losgelöst von ihren Prozessen – betrachtet. Die linke Kante des Dreiecks steht – abwärts gerichtet – für eine immer genauere Betrachtung der Morphologie der Pflanze. Klassische Vertreter dieser Richtung sind mit nicht-sensitiven Lindenmayer-Systemen [Prusinkiewicz & Lindenmayer 1990] erstellte Einzelbaummodelle (siehe Kapitel 2.2.6.).

Die verbleibende Kante an der Basis des Dreiecks drückt den entsprechenden Grad aus, mit dem die Struktur (nach links) bzw. die Prozesse (nach rechts) in einem Modell Eingang finden [vergleiche Kurth 1994a, Kurth 1999, Kurth et al. 2006].

Eine Vielzahl der Modelle bildet eine Zwischenform, d. h. sie liegen im Innern des Dreiecks. Für die Verwendung zur Vorhersage des Baumwachstums in Abhängigkeit von Durchforstungsmaßnahmen ist allerdings Folgendes zu beachten: Eine Abwärtsbewegung im Modelldreieck ist gleichzusetzten mit einer höheren räumlichen oder zeitlichen Auflösung bzw. einer detaillierteren Prozessbetrachtung im Sinne der Hierarchietheorie [O'Neill et al. 1986]. Dementsprechend steigt der Komplexitätsgrad des Modells – und somit auch der Berechnungsaufwand und die hierfür benötigten Informationen. Eine Aufwärtsbewegung im Modelldreieck beinhaltet hingegen die Gefahr des Verlustes von Detailinformationen, die für die Interpretation der Konsequenzen

von Durchforstungsmaßnahmen auf Einzelbäume benötigt werden. Jedes Modell liegt also im Spannungsfeld zwischen verfügbarem Wissen, Rechenaufwand und gewünschter Aussagekraft. Maßgebend hierbei ist also der Verwendungszweck, d. h. im vorliegenden Fall das gewünschte Lernziel.

2.2 Übersicht über einige aktuelle Wachstumssimulatoren für den deutschen Raum

Nagel unterteilt Wachstumssimulatoren in zwei Hauptkomponenten. Mit der ersten Hauptkomponente umfasst er die "*waldwachstumskundlichen Modelle, die auf Grundlage der Versuchsflächendaten Prozesse wie Wachstum, Mortalität und Einwuchs beschreiben. Diese Modelle lassen sich in mathematische Formeln fassen und unterscheiden sich baumartenweise in ihren Koeffizienten*" [Nagel 2003 S. 33]. Die zweite Hauptkomponente ist "*eine bedienungsfreundliche und einfach zu handhabende Software*" [Nagel 2003, S. 33].

Wachstumssimulatoren unterscheiden sich daher nicht nur im verwendeten Modellansatz, sondern im starken Maße auch in der programmtechnischen und lizenzrechtlichen Umsetzung. Die Bandbreite geht von voll integrierten Software-Paketen mit zahlreichen Features und Dialogen – bei denen der Soureccode für die Benutzerin eine Black-Box darstellt – bis hin zu Open Source Programmbibliotheken zum Einbau in eigene Computerprogramme.

Die Gestaltung der zweiten Hauptkomponente ist für den Einsatz in einem offenen Lernsystem aber ebenso wichtig wie die verwendete Modellkomponente (vergleiche auch Kapitel 3.8.4). Im Folgenden werden einige ausgewählte Wachstumssimulatoren, die aktuell an deutschen Universitäten in der Lehre eingesetzt werden, kurz charakterisiert. Neben den ausgewählten Bestandeswuchsmodelle, gibt es noch zahlreiche weitere Modellansätze, die hier nicht besprochen werden können [vergl. z. B. Ek & Monserud 1994, v. Gadow 1987, Sterba et al. 1995, Hasenauer 1994].

2.2.1 Die Ertragstafel

Die Ertragstafeln können nur im weiteren Sinne als Wachstumssimulator bezeichnet werden, wenn sie durch ein Computerprogramm angebunden werden und zur Vorhersage des Bestandeswachstums genutzt werden. Basierend auf Langzeituntersuchungen auf Probeflächen sind Ertragstafeln hoch aggregierte Modelle, die das Bestandeswachstum in Abhängigkeit von Baumart, Alter, Baumhöhe, Grundfläche und Durchforstungsstärke angeben. Überwiegend erstellt für Reinbestände, waren Ertragstafeln für lange Zeit das gebräuchlichste Werkzeug in der forstlichen Praxis. Neuere Erkenntnisse in der Waldwachstumslehre, der zunehmende Anteil von Mischbeständen sowie sich ändernde klimatische Bedingungen zeigen die Schwachstellen der Ertragstafeln.

Die Anwendung von Ertragstafeln ist einfach und setzt nur geringe Berechnungen voraus. Eine Berücksichtigung von einzelnen Bäumen erfolgt nicht – sie kann nur als Element in die Berechnung der Durchforstungsstärke eingehen. Gängige Ertragstafeln sind die von Assmann & Franz [1963] für Fichte, Schober [1987] für diverse Baumarten und Lembcke et al. [2000] für Kiefer.

2.2.2 SILVA

SILVA ist ein einzelbaumorientierter Wachstumssimulator, welcher am Lehrstuhl für Waldwachstumskunde der Technischen Universität München unter der Leitung von Pretzsch entstanden ist [siehe hierzu Pretzsch 1990 & 1992, Kahn & Pretzsch 1997, Pretzsch & Kahn 1998, Seifert 2002, Pretzsch 2001, Pretzsch et al. 2002].

Pretzsch bezeichnet das Programm Silva als ein "*Hybrid aus Prognose- und Erklärungsmodell […] für den Einsatz in der forstwirtschaftlichen Praxis, Forschung und Ausbildung konzipiert. Es prognostiziert das Wachstum von Rein- und Mischbeständen aller Alterszusammensetzungen und kann für die Entwicklung und Optimierung waldbaulicher Behandlungsstrategien auf Bestandesebene eingesetzt werden*" [Pretzsch 2001, S. 193].

Die Parametrisierung von SILVA stützt sich auf Messdaten von forstlichen Langzeitversuchsflächen in Deutschland und der Schweiz [Kahn 1994] mit verschiedenen Wachstums- und Standortverhältnissen sowie auf das Versuchsflächennetz des Münchener Lehrstuhls für Waldwachstumskunde.

Eingangsgrößen für SILVA sind Bestandeskennwerte, Standortvariablen und Vorgaben der waldbaulichen Behandlung. "*Das Model abstrahiert den Bestand als Mosaik von Einzelbäumen, deren Entwicklung von den gegebenen Standortfaktoren, der Wuchskonstellation des Baumes im Bestand und seiner individuellen Entwicklung determiniert wird.*" [Pretzsch 2001, S. 1994].

Grundlage der Wachstumsprognose ist ein Standort-Leistungsmodell für die einzelnen Baumarten. Wesentliches Element für die einzelbaumorientierte Wachstumsprognoseberechnung ist aber, neben dem baumarten- und standortspezifischen Wachstumspotenzial, die individuelle Konkurrenzsituation der Bäume. Pretzsch et al. benutzen für die räumliche Konkurrenzanalyse drei Maßzahlen, welche sie soziale Stellung eines Baumes (*KKL*), die Symmetrie bzw. Asymmetrie der Konkurrenzstellung (*NDIST*) und die Baumartenmischung im Umfeld des betrachteten Baumes (*KMA*) nennen [vergl. Pretzsch 2001, S. 128ff].

Bei der Berechnung des *KKL* als Indikator der Kronenkonkurrenz wird für jeden Baum (B_z) überprüft, welche benachbarten Bäume mit ihrer Spitze in einem nach oben offenen 60°-Kegel liegen, der im Kronenmittelpunkt bei 60% der Höhe des zu untersuchenden Zentralbaumes (B_z) entsteht. Für die Menge der Konkurrenten wird die Winkelabweichung (wa_{iz}) der Verbindungslinie Kegelursprung – Baumspitze von der Senkrechten bestimmt und von 60 subtrahiert. Diese aufsummierten Winkel ($Beta_i = 60 - wa_i$) ergeben den *KKL* als ein relatives Maß für den Konkurrenzdruck des betrachteten Baumes (B_z) durch seine Nachbarn. Um neben der Entfernungs- und Höhenrelation auch die Größenrelation zu berücksichtigen, wird der Winkel beta mit dem Quotienten der Kronenquerschnittsflächen (*KQF*) zwischen dem Zentralbaum und dem Nachbarn in Höhe des Kegelursprunges (60% der Höhe des Zentralbaumes) gewichtet. Als weiterer Gewichtungsfaktor wird ein Lichttransmissionskoeffizient (TM_l) in Anlehnung an Ellenberg (1963 zitiert in Pretzsch 2001) von 1,0 für Fagus Silvatica L. und Abies alba Mill., von 0,5 für Quercus petraea (Mattuschka) Liebl. und 0,2 für Pinus silvestris L. verwendet. Formel (1) zeigt die Berechnung des relativen Konkurrenzindexes *KKL*:

(1) $$KKL_z = \sum_{\substack{i=1 \\ i \neq z}}^{n} (60 - wa_{iz}) \cdot \frac{KQF_i}{KQF_z} \cdot TM_i$$

mit B_z als Zentralstamm und B_i als Konkurrent.

Von entscheidender Bedeutung für die Berechnung des *KKL* ist die zugrundegelegte Kronenform. In SILVA wird hierfür eine einheitliche Berechnungsform für verschiedene Baumarten verwendet. Durch eine unterschiedliche Parametrisierung ergeben sich die artspezifischen Ausformungen der Krone. Unterschieden wird zwischen Schatten- und Lichtkrone.

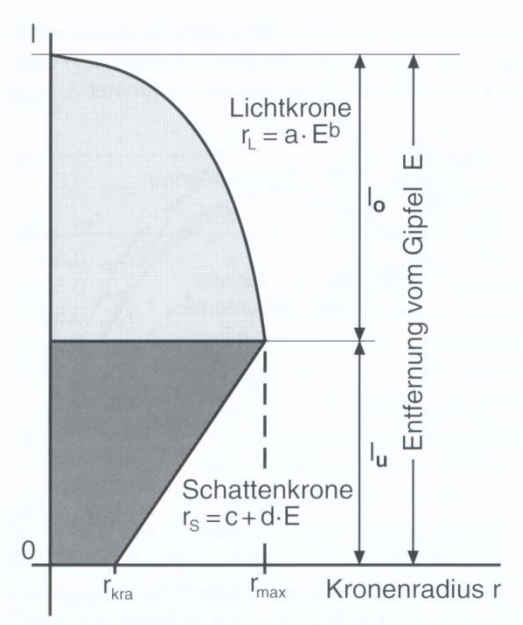
Der Radius der Lichtkronen wird in Abhängigkeit zur Distanz *E* von der Baumspitze mit Formel (**2**) berechnet:

(2) $$r_l = a \cdot E^b$$

Für die Schattenkrone erfolgt die Berechnung als Kegelstumpf mit Formel (3):

(3) $$r_s = c + E \cdot d$$

Abbildung 2 zeigt die Einteilung in Licht- und Schattenkrone. Tabelle 2 enthält die artenspezifischen Parameter für die wichtigsten deutschen Wirtschaftsbaumarten. Eingangsgrößen sind die Kronenlänge und der maximale Kronenradius. Die hier beschriebenen Kronenformmodelle sind Grundlagen für die visuelle Darstellung der Baumkronen im späten verwendeten Modell in GroIMP (siehe Kapitel 3.7).



r	**Kronenradius**
l	**Kronenlänge**
rl	**Kronenradius Lichtkrone**
lo	**Länge der Lichtkrone**
rs	**Kronenradius der Schattenkrone**
lu	**Länge der Schattenkrone**
r_{max}	**maximaler Kronenradius**
r_{kra}	**Kronenradius am Kronenansatz**
E	**Entfernung zur Baumspitze**

a-d **baumartenspezifische Parameter**

Abbildung 2. Biometrische Nachbildung der Kronenform in SILVA mit $r_i = a \cdot E^b$ für die Lichtkrone und $r_s = c + E \cdot d$ für die Schattenkrone [aus Pretzsch 2001, S. 205].

Baumart	Lichtkrone			Schattenkrone		
	a	l_o	b	c	d	r_{kra}
Picea abies	r_{max}/l_o	$l \cdot 0{,}66$	1,00	für alle Baumarten gilt: $r_{max} - d \cdot l_o$	für alle Baumarten gilt: $\frac{r_{kra} - r_{max}}{l - l_o}$	$r_{max} \cdot 0{,}50$
Abies alba	$r_{max}/(l_o)^{0,5}$	$l \cdot 0{,}50$	0,50			$r_{max} \cdot 0{,}50$
Pinus silvestris	$r_{max}/(l_o)^{0,5}$	$l \cdot 0{,}64$	0,50			$r_{max} \cdot 0{,}50$
Larix decidua	$r_{max}/(l_o)^{0,45}$	$l \cdot 0{,}80$	0,45			$r_{max} \cdot 0{,}80$
Fagus silvatica	$r_{max}/(l_o)^{0,33}$	$l \cdot 0{,}40$	0,33			$r_{max} \cdot 0{,}33$
Quercus petraea	$r_{max}/(l_o)^{0,5}$	$l \cdot 0{,}50$	0,50			$r_{max} \cdot 0{,}50$
Alnus glutinosa	$r_{max}/(l_o)^{0,5}$	$l \cdot 0{,}56$	0,50			r_{max}

Tabelle 2: Parameter der SILVA-Kronenformmodelle für Licht- und- Schattenkrone für wichtige Wirtschaftsbaumarten [aus Pretzsch 2001, S. 206].

SILVA umfasst neben dem Prognoseteil zahlreiche weitere Programmteile zur Generierung [vergl. Pretzsch 1993 & 2001, S. 212ff], Darstellung [vergl. Pretzsch & Seifert 1999, Seifert 2002] und Analyse von Bestandesdaten [Pretzsch2001, S. 255ff]. Die Software ist voll integriert, der Sourcecode nicht allgemein offen.

2.2.3 SIBYLA

SIBYLA ist ein weiterer einzelbaumorientierter Waldwachstumssimulator, der in seiner Modellkomponente sehr stark von SILVA beeinflusst worden ist. SIBYLA ist von Fabrika für die Slowakische Forstverwaltung entwickelt worden und umfasst ebenso wie SILVA vielfältige Programmkomponenten zur Strukturgenerierung, Prognoserechnung, Analyse und Visualisierung. Das Prognosemodul ist mit Daten aus slowakischen Inventurstichproben parametrisiert worden. [siehe Fabrika 2003 & 2005, Fabrika & Ďurský 2005, Fabrika et al. 2005].

Für den deutschen Raum verfügt SIBYLA über die Option, das Prognosemodul von SILVA zu verwenden. Die im Kapitel 2.2.2 ausgeführten Grundlagen zur Konkurrenz gelten also auch für SIBYLA.

Im Rahmen des E-Learning Academic Network Niedersachsen Moduls Forst [ELAN 2005 S. 121] wurde in einer Kooperation zwischen Fabrika und dem Institut für Forstliche Biometrie und Informatik die Verbindung zwischen Niedersächsischen Forstinventurdaten und SIBYLA geschaffen. Als Resultat stehen durch SIBYLA generierte einzelbaumbasierende Strukturdaten für alle Bestände des Forstamtes Winnefeld im Solling zur Verfügung [Zindler 2005].

SIBYLA wurde in Delphi entwickelt. Der Sourcecode von SIBYLA ist nicht allgemein verfügbar. Das Datenmodell liegt offen, auf die Acess-Datenbank kann direkt zugegriffen werden. Die Nutzung von SIBYLA im Rahmen der Lehre an der Universität Göttingen ist lizensiert. Eine Programmversion für den Batchbetrieb ist vorhanden. Die in Sibyla integrierte Virtual Reality Komponente war wesentlicher Ideengeber für den Virtual Forester Client (siehe Kapitel 3.3)

2.2.4 BWIN

Die BWIN-Familie bezeichnet die an der Niedersächsischen Forstlichen Versuchsanstalt unter der Leitung von Jürgen Nagel entwickelten Wachstumssimulatoren. Wir sprechen hier von einer Programmfamilie, da der Wachstumssimulator BWIN seit 1995 erhebliche Änderungen durchlaufen hat. Diese umfassen nicht nur die Modellkomponente, sondern insbesondere auch die zweite Hauptkomponente, die softwaretechni-

sche Umsetzung. So hat sich BWIN von einer in Delphi programmierten properitären Lösung für Windows Betriebssysteme [Nagel 1999] zur aktuellen Version BWINPro 7.1 entwickelt [vergl. Nagel et al. 2006], welche in Java programmiert wurde und auf den Open Source Kernbibliotheken TreeGross (Tree Growth Open Source Software) [siehe Nagel 2002 & 2003] beruhen. So haben Nagel et al. zum einen die Entwicklung eines voll integrierten Softwarepaketes mit den üblichen umfangreichen Programmpaketen zur Strukturgenerierung, Prognoserechnung, Analyse und Visualisierung (Abbildung 3) vorangetrieben, zum anderen aber die Kernroutine des Waldwachstumssimulators als Java-Klassenbibliotheken zur Integration in eigene Programme zur Verfügung gestellt.

Die "*Modellgleichungen für Zuwachsprognose und Datenergänzungsroutinen wurden 1995 mit Hilfe der zum damaligen Zeitpunkt vorhandenen Versuchsflächendaten parametrisiert, die von Versuchsflächen der Niedersächsischen Versuchsanstalt in Nordwestdeutschland stammen*" [Nagel et al. 2002, Seite 486]. Im Folgenden wurde die Datengrundlage auf 12 Baumarten erweitert und um Daten von Mischbestandesversuchsflächen ergänzt [vergleiche Nagel et al. 2002, Seite 487].

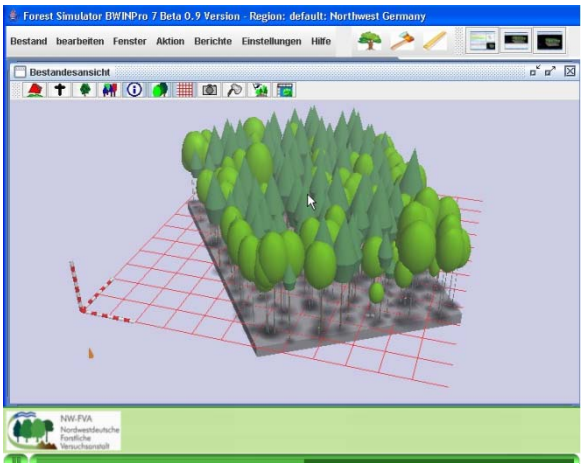


Abbildung 3: 3D-Ansicht in BWINPro 7.1.

BWINPro ist ebenfalls ein einzelbaumorientierter Wachstumssimulator. Grundlage des Baumwachstums ist auch hier ein Standort-Leistungs-Potential für die Baumarten in Abhängigkeit vom Baumalter, welches durch die Kronengröße und Konkurrenzsituation an die individuelle Baumsituation angepasst wird.

Wesentlicher Unterschied zu SILVA ist aber, dass der verwendete Konkurrenzindex C66 positionsunabhängig verwendet wird. "*Der Konkurrenzindex C66 eines Einzelbaumes ist die Kronenschirmfläche (ks) aller Bäume, wenn sie in einer Höhe von 66 Prozent der Kronenlänge des Bezugsbaumes (Abbildung 4, Baum 2) geschnitten werden. Liegt der Kronenansatz eines Baumes über der Schnitthöhe (Abbildung 4, Baum 4), so wird seine volle Kronenschirmfläche berücksichtigt*" [Nagel 1999, Seite 20, Nummer der Abbildung durch den Autor angepasst].

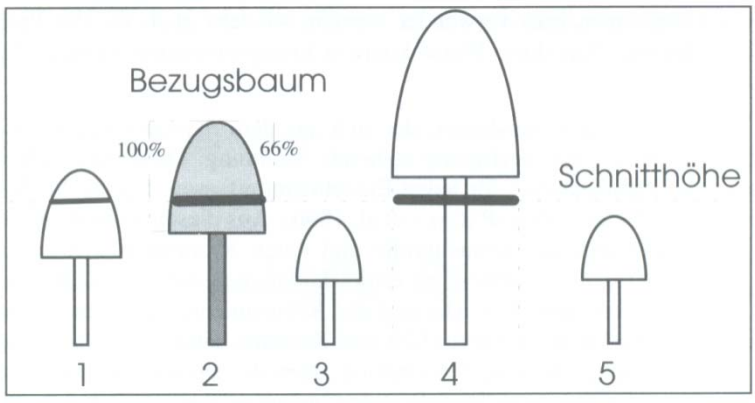


Abbildung 4: Herleitung des Kronenkonkurrenzindexes *C66* (zweidimensional) [aus Nagel 1999, Seite 20].

Formel (4) zeigt die Herleitung des Konkurrenzindexes *C66* für einen Bezugsbaum z wie er in BWIN verwendet wird:

(4) $$C66_z = \frac{\sum_{i=1}^{n} ks_i \, 66\% Kronenlänge_z}{A}$$; mit A als Bestandsfläche

Die Veränderung der Konkurrenz bei Durchforstungsmaßnahmen wird durch folgende Berechnung ausgedrückt:

(5) $$C66c = C66_{vor\_Durchforstung} - c66_{nach\_Durchforstung}$$

Alter, Kronenmantelfläche, *C66* und *C66c* gehen sowohl in die Durchmesser-, wie auch die Höhenzuwachsberechnung ein. Die positionsunabhängige Betrachtung des Konkurrenzindexes wirkt sich bei größeren Beständen daher auf die Beurteilung von Einzeldurchforstungsmaßnahmen aus.

2.2.5 Komplexe ökophysiologische Prozessmodelle

Es gibt zahlreiche verschiedene Ansätze für ökophysiologische Prozessmodelle mit hohem Komplexitätsgrad. Prominente Vertreter dieser Modellklasse sind Tragic++ [Hauhs et al. 1995 & 2003] entwickelt in Bayreuth, Treedyn [Bossel 1994 & 1996] und Lignum aus Finnland [Sievänen 1993]. Gemeinsam ist den Modellansätzen die sehr detaillierte Betrachtung der Prozesse innerhalb von Einzelbäumen und die damit verbundenen Schwierigkeiten der Parametrisierung für mehrere Baumarten. Zudem ist die Umsetzung der obengenannten Modelle in Simulationsprogramme auf Ebene klassischer objektorientierter bzw. prozeduraler Programmierung erfolgt, so dass eine Anpassung Expertenkenntnisse in der entsprechenden Programmiersprache erfordert. Der potentielle Erkenntnisgewinn ist bei Verwendung komplexer ökophysiologischer Modelle in der Lehre zwar sehr hoch, stellt aber an die Lernenden enorme Anforderungen hinsichtlich Vorkenntnisse und Ansprüche. Es besteht also Bedarf für einfachere Einstiegsmodelle mit einem offenen Ansatz der Modellumsetzung für die Lehre.

2.2.6 GROGRA / GroIMP

Die Programme GROGRA [Kurth 1994b, 1999] und GroIMP [Kniemeyer 2004] gehören im engeren Sinne nicht in die Gruppe der Simulatoren für Bestandeswachstum. Die Programme sind vielmehr allgemeine Modellschalen zur Durchführung von Simulation auf Basis von Modellen, die in speziellen regelbasierten Modellsprachen geschrieben werden. Interpreter und Modell sind klar von einander getrennt. Die Modellschale dient als Graphisches Userinterface und als Interpreter der Modelle. Bei Verwendung entsprechender Modelle können die Programme als Wachstumssimulatoren für Waldbestände verwendet werden.

Der **Growth Grammar Interpreter** (GROGRA) [Kurth 1994b, 1999] interpretiert sensitive und nicht-sensitive Wachstumsgrammatiken [siehe Kurth 1994b & Kurth & Sloboda 1999a, b], die eine Erweiterung der Lindenmayer-Systeme von Aristid Lindenmayer darstellen [L-Systems nach Lindenmayer 1968]. L-Systeme sind ein Beispiel für einen regelbasierten Ansatz [siehe Kurth 2002] in der Strukturmodellierung

von Pflanzen [vergl. Prusinkiewicz & Lindenmayer 1990] und bestehen aus einem Satz von Ersetzungsregeln in der Form

```
linke Regelseite -> rechte Regelseite
```

die wiederholt auf einen String angewendet werden. Die einzelnen Symbole können dabei parametrisiert werden. Durch die wiederholte Anwendung der Regeln auf einen String kann so – ausgehend von einem Startwort (Axiom) – eine komplexe Struktur entstehen. Verzweigungen werden durch ein gesondertes Symbol im String ausgedrückt.

Bei Verwendung eines speziellen Satzes von Symbolen [siehe Kurth 1999] kann der String mit dem Ansatz der Turtle Geometrie [Abelson & diSessa 1982] in eine räumliche Objektstruktur umgesetzt werden.

Problem der auf einen String angewandten einfachen Ersetzungsregeln ist die nicht vorhandene Beziehung zwischen den Symbolen – abgesehen von der Position im String – und die eingeschränkte Sensitivität. Sensitivität ist die Abhängigkeit einer Ersetzungsregel von externen Einflüssen. Dies können Einflüsse der Umgebung oder von anderen Teilen der eigenen Struktur sein [siehe Kurth 1998, S.10]. Als Verallgemeinerung der Kontext-Sensitivität [erläutert bei Prusinkiewicz & Lindenmayer 1990] hat Kurth deshalb das Konzept der globalen Sensitivität in GROGRA implementiert [siehe Kurth 1994b]. Mit diesem Ansatz verfolgt Kurth unter anderem das Ziel "*to establish links between the different approaches [Struktur und Prozessmodellierung: Anmerkung des Autors] and to show the usefullness and manageability of models with higher spatial resolution*" [Kurth 1994a, S. 5].

Die Integration der Sensitivität wird über spezielle Erweiterungsfunktionen erreicht, die in den Ersetzungsregeln verwendet werden können – z. B. für die Berechnung der räumlichen Nähe zweier Objekte. Die Funktionen sind aber Teil der Modellschale; für jede neue sensitive Funktion ist daher eine Neukompilierung von GROGRA erforderlich.

GROGRA wurde im Feld der Waldökosystemforschung zur Simulation der Struktur von Pflanzen eingesetzt [siehe z. B. Kurth & Anzola 1997, Kurth & Bredemeyer 1996, Kurth 1999, Kurth & Sloboda 2001] und kann mit stärker prozessorientierten Programmen verbunden werden [vergl. Früh 1995, Früh & Kurth 1999, Lanwert et al. 1998, Anzola 2002, Dzierzon 2003].

Zudem wurde im Rahmen des E-Learning Academic Network Projektes an der Fakultät für Forstwissenschaften und Waldökologie eine multimediale Lern-CD zum Thema Struktur-/Funktionsmodellierung mit Lindenmayer-Systemen und GROGRA entwickelt [Lanwert et al. 2004] und in der Lehre eingesetzt.

Es bleiben aber die oben genannten Einschränkungen in der Sensitivität. Einen Ausweg bietet hier die Erweiterung des regelbasierten Modellierungsansatzes auf Graph-Grammatiken an. Dieser Ansatz wird im Folgenden in dieser Arbeit verwendet. Eine detailliertere Beschreibung der **Relationalen Wachstumsgrammatiken**, der Modellsprache **XL** und der Modellschale **GroIMP** wird daher im Kapitel 3.6 gegeben.

2.3 E-Learning in der Wachstumsmodellierung

Bislang gibt es nur wenige, verteilte Ansätze zur Integration von E-Learning-Kursen zum Themengebiet Waldwachstumsmodellierung in den Universitäten.

So verwendet Nagel z. B. in seinem webbasiertem Java-Kurs für Studierende der Forstwissenschaften und Waldökologie an der Universität Göttingen die TreeGrOSS-Bibliotheken (s. Kapitel 2.2.4).

Zudem hat Nagel [2006] eine CD Forest Tools 2 zusammengestellt, mit einer Sammlung von Programmen und Lernmaterial zu den Themengebieten Forsteinrichtung, Waldmesslehre und Waldwachstum. Diese CD richtet sich an Praktiker und Studierende und enthält unter anderem auch die Programme BWINPro und TreeGrOSS.

Zu BWINPro und zum Thema Waldwachstum gibt es von Nagel, zum Thema Waldwachstumskunde von Staupendahl Online Material im Lernmanagement System StudIP[4] der Universität Göttingen.

Weiterhin existiert auf der WebSite der Nordwestdeutschen Forstlichen das Spiel SIMWALD, mit dem die Auswirkung der Waldbewirtschaftung auf die Nachhaltigkeit von Waldfunktionen erkundet werden können [Nagel 2007]. Der Einsatzschwerpunkt, der auf TreeGrOSS basierenden Software, liegt aber in Schulen und der Öffentlichkeitsarbeit.

Mit der Lern-CD zur Funktions-/Strukturmodellierung für den Masterschwerpunkt Waldökosystemanalyse und Informationsverarbeitung an der Universität Göttingen wurde ein erstes vollständiges Kursangebot für einen Onlinekurs zur Pflanzenmodellierung mit Lindenmayer-Systemen und GROGRA erstellt [Lanwert et al. 2003]. Eine Aktualisierung des Angebotes für XL mit GroIMP ist für 2008 geplant.

Sowohl die Programme BWINPro, SILVA als auch SIBYLA werden unterrichtsbegleitend in verschiedenen Veranstaltungen an den deutschen Forstfakultäten verwendet.

[4] http://www.studip.uni-goettingen.de

3 Aufbau des Systems

3.1 Übersicht der Komponenten

Das vorliegende System besteht aus 6 Komponenten. Diese sind:

1 Datenkomponente

Die **Forester-Bestandesbeschreibungen** enthalten Angaben zu den wesentlichen Elementen der virtuellen Bestände und sind im Virtual Reality Modelling Language (VRML)-Format unter Verwendung eines für das Projekt definierten Prototypen geschrieben (siehe Anhang 10.1).

4 Programmkomponenten

Der **Virtual Forester** ist das Graphische Interface zur Nutzerin, welches die Visualisierung der virtuellen Bestände übernimmt und die Funktionalität für Eingriffe in die Bestandesstruktur bereitstellt. Der Virtual Forester wird im Netzwerk- und im standalone-Betrieb verwendet.

Der **Elan-Sim Server** (**ESS**) ist die Serverkomponente des Virtual Forester. Das Programm verbindet mehrere Virtual Forester-Instanzen für die Teamarbeit und distribuiert Szenen und Texturen. Als zweiten Aufgabenbereich übernimmt das Programm die Kommunikation mit extern angebundenen Wachstumsgeneratoren. Der Elan-Sim Server wird nur im Netzwerkmodus benötigt.

Der **Wachstumsgenerator** GroIMP stellt die Modellschale für die Anwendung von Wachstumsmodellen. Die Modelle werden mit erweiterten L-Systemen (XL) geschrieben. GroIMP als Wachstumsgenerator kann in beiden Betriebsmodi verwendet werden und kann durch andere geeignete Wachstumsgeneratoren ersetzt werden. Bestandteil von GroIMP ist ein integrierter HTTP-Server.

Der **Quality Identifier** ist ein optionales Tool zur verbesserten Darstellung von Baumqualitätsmerkmalen. Bei entsprechend vorhandenen Informationen über typische Qualitätsmerkmale der Bäume (z. B. Art & Zwiesel) im Baum-Datenbereich einer Bestandesszene wird mit Hilfe einer Texturdatenbank die Bestandesszene so umgeschrieben, dass die relevanten Qualitätskriterien im Virtual Forester besser erkennbar sind.

1 Modellkomponente

Die **Modellkomponente** umfasst die tatsächlichen Berechnungen für die Wachstumsprognose. Das in dieser Arbeit beschriebene Modell ist ein einfaches, ökophysiologisches, beschattungsabhängiges Wachstumsmodell, welches in XL für die Modellschale GroIMP umgesetzt wurde. Die Klassen zur Integration der Bestandesbeschreibungen wurden als ein Erweiterungspaket für GroIMP in Java konzipiert und sind daher nicht Teil eines spezifischen Modells. Dies ermöglicht es – gemäß der Anforderung, die Modellkomponente für den Lernprozess variabel zu halten (siehe Kapitel 1.4.1) –, beliebige andere Modellansätze zu verwenden.

Im Folgenden werden ab Kapitel 3.2 die einzelnen Komponenten detailliert beschrieben. Für ein besseres Verständnis betrachten wir aber zuerst das Zusammenspiel der einzelnen Komponenten in der Übersicht.

3.1.1 Zusammenspiel der Systemkomponenten

Zur Beschreibung der Zusammenarbeit der einzelnen Systemkomponenten ist es notwendig, zwei Betriebsvarianten zu unterscheiden.

Einzelplatzbetrieb

In dieser Variante wird das System auf einem Einzelarbeitsplatz ausgeführt. Ein Netzwerkanschluss ist nicht erforderlich. Der Wachstumsgenerator GroIMP und der Virtual Forester Client müssen lokal auf dem Rechner gestartet werden. Zur Verwendung im Virtual Forester Client muss eine Bestandesbeschreibung von der Platte geladen werden. Der Austausch der Bestandesbeschreibung zwischen den Programmen wird von der Nutzerin durchgeführt. Hierzu ist es notwendig, die aktuell verwendete Bestandesbeschreibung auf einem Datenträger zu speichern und per http [Fielding et al. 1999] als HTML-Formular [W3C: HTML 4.01 1999] aus einem Browser heraus an den Wachstumsgenerator zu senden. Die Antwort enthält die Adresse eines weiteren HTML-Formulars, über welches die Bestandesbeschreibung mit den Ergebnissen der Simulation abgerufen werden kann und zur Speicherung bereit steht. Zur Verwendung im Virtual Forester Client muss die neue Bestandesbeschreibung von der Platte geladen werden.

Netzwerkbetrieb

In der Netzwerkvariante werden mehre Virtual Forester Clients über eine Serverkomponente miteinander verbunden (Abbildung 5). Diese Verbindung erlaubt die Eröffnung einer Session, d.h. das gemeinsame Betreten und Arbeiten in einer Bestandesszene mit mehreren Clients. Änderungen an der Szene – z. B. das Fällen eines Baumes – werden in allen verbundenen Clients gleichzeitig ausgeführt.

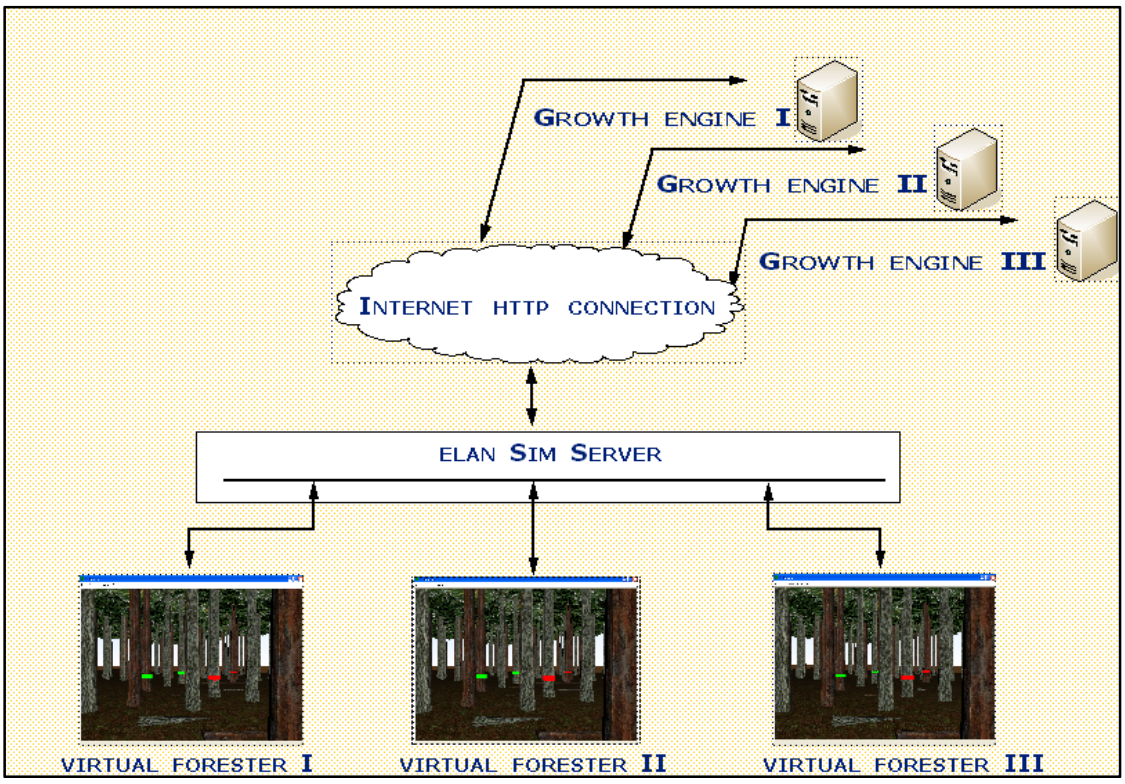


Abbildung 5: Schematische Übersicht des Zusammenspiels der Programmkomponenten des E-Learning-Systems im Netzwerkbetrieb.

Bestandesbeschreibungen werden im Netzwerkbetrieb vom Server bereitgestellt. Die Nutzerinnen können hierzu nach Anmeldung aus einer Szenenliste wählen. Auch für die Simulation ist kein lokales Speichern auf den Clientrechnern notwendig. Ausgelöst durch den Session-Master, wird die aktuelle Bestandesbeschreibung vom Elan-Sim Server per HTTP-POST [Fielding et al. 1999, S. 35] an den ausgewählten Wachstumsgenerator geschickt und die neue Bestandesbeschreibung mit den Simulationsergebnissen in die Szenenliste aufgenommen.

Die URLs [Berners-Lee et al.2005] möglicher Wachstumsgeneratoren sind Bestandteile der Bestandesbeschreibung. So wird beim Erzeugen der Ausgangsbeschreibung festgelegt, welche Wachstumsgeneratoren und -modelle in einem Lernszenario verwendet werden können.

3.1.2 Protokolle

Für die Ansprache des Wachstumsgenerators wurde HTTP [Fielding et al. 1999] als Protokoll festgelegt. Maßgebend für diese Entscheidung waren 3 Gründe:

- HTTP ist ein bewährtes Standardprotokoll für den Datentransfer im Internet. Es gibt für viele Programmiersprachen und Plattformen Codebibliotheken mit frei verfügbaren Paketen zur Nutzung von HTTP. Dies vereinfacht die Integration einer auf HTTP basierenden Schnittstelle in bestehende Programme zur Wachstumssimulation zur Anbindung an den Elan Sim Server. Die genauen Vorgaben zur automatischen Anbindung eines Wachstumsgenerators an den ESS werden in Kapitel 3.4.5 ausgeführt.
- Wachtumsgeneratoren ohne geeignete eigene Schnittstelle können ggf. mit Hilfe der api-Schnittstelle eines Webservers angebunden werden. Der Webserver übernimmt dabei die Kommunikation, speichert die Szene, startet den Wachstumsgenerator als Prozess und liefert nach Beendigung der Berechnung die Ergebnisszene aus.
- Bei Verwendung von HTTP als Protokoll kann ein Wachstumsgenerator mit entsprechender Schnittstelle in Form eines Web-Service verwendet werden. Auch ohne direkte Anbindung an den ESS können geeignete Bestandesszenen – manuell unter Verwendung eines Standard-Webbrowsers – zur Prognoseberechnung an den Wachstumsgenerator geschickt werden.

3.2 Die Bestandesbeschreibung (Eine VRML-Waldszene)

Grundlage der Bestandesbeschreibung ist die Virtual Reality Modelling Language VRML in der Version 2.0 [Web3D: VRML 1997]. VRML wurde entwickelt, um begehbare dreidimensionale Welten zu definieren und diese über das Internet zu verbinden. Version 1 basierte auf der Sprache Open Inventor von Silicon Graphics [SGI] und wurde unter Federführung von Mark Pesce 1994 entworfen. Mit der Version 2.0 – welche auf der Sprache Moving Worlds von SGI basiert – wurde 1996 VRML um dynamische Komponenten erweitert [vergl. Hartmann & Wernecke 1996]. Hinzugekommen sind – unter anderem – event-auslösende Sensoren, die Möglichkeit zur dynamischen Behandlung der ausgelösten Events, die Integration mit JavaScript oder Java sowie Prototypendefinitionen von eigenen Elementtypen (node types). Aber "*eines der ursprünglichen Entwicklungsziele von VRML bleibt auch bei VRML 2.0 ungelöst: Es gibt immer noch keinen Standard für die Interaktion mehrerer Benutzer in einer 3D-Szene*" [Diehl 1997].

Zum Begehen einer VRML-Welt (Szene) wird ein VRML-Browser benötigt. Dies kann eine eigenständige Applikation sein [FreeWRL], ein Applet [blaxxun:3D Viewer] oder ein VRML-Plugin für einen Web-Browser [blaxxun Contact, OpenVRML]. Dieses Programm übernimmt die Interpretation und visuelle Umsetzung der VRML-Welt und stellt die grundlegende Funktionalität für Aktionen und Interaktionen.

3.2.1 Grundprämisse der Bestandesbeschreibung

Eine Bestandesbeschreibung ist die textliche Beschreibung einer VRML-Welt, die einen Waldbestand darstellt und die virtuelle Ausführung der in Kapitel 1.4.3 definierten forstlichen Aufgabenstellungen erlauben soll.

Diese Anforderung an die Syntax und Semantik der Beschreibung der virtuellen Waldbestände steht in direktem Zusammenhang mit dem verwendeten Programm zur Darstellung der VRML-Welten. Entscheidend ist hierbei die Umsetzung der spezifischen interaktiven Eingriffe in die Bestandesstruktur des Waldes durch die Lernenden.

Im Wesentlichen stehen hier 2 mögliche Entwicklungslinien zur Wahl:

1. Möglichkeit: Die Definition aller Bestandeselemente, sowie deren Erscheinungsbild und deren spezifische Aktionen und Interaktionen, wird in der VRML-Datei vorgenommen. Aufbauend auf den grundlegenden dynamischen Möglichkeiten eines VRML-Browsers werden die besonderen Fähigkeiten der Waldbestandeselemente in der VRML-Beschreibung festgelegt. Dies erfolgt durch die Verwendung von Sensorknoten, Routes und Skriptknoten bei der Definition der Bestandesstruktur [vergl. Ames et al. 1997]. Dieser Weg erhöht deutlich die Komplexität der Szene, erlaubt aber den Einsatz eines Standardprogramms für den Browser in der VRML-Welt.
2. Möglichkeit: Die Definition der Bestandesstruktur sowie des Erscheinungsbildes der einzelnen Bestandeselemente wird in der VRML-Datei vorgenommen. Die Definition der spezifischen Aktionen und Interaktionen wird aber weitestgehend in den Sourcecode des VRML-Browsers verlagert. Dies erlaubt es, die Bestandesbeschreibung einfacher zu halten, erfordert aber die Programmierung eines eigenen VRML-Browsers, der weiß, welche besonderen Fähigkeiten die Waldbestandeselemente besitzen und wie er diese behandeln soll.

Gemäß der in Kapitel 1.3 definierten Ziele und der in Kapitel 1.4 daraus abgeleiteten Anforderungen an das System wurde für die Bestandesbeschreibung als Grundprämisse postuliert, dass sie für den Anwender ohne spezifische VRML-Kenntnisse verständlich und anwendbar sein muss.

Das Ziel war daher die Entwicklung einer auf VRML basierenden Beschreibung von virtuellen Waldbeständen, die sich durch einen hohen Grad an Einfachheit, Lesbarkeit und Wiederverwendbarkeit auszeichnet.

Es wurden daher folgende Vorgaben festgesetzt:

- Die Umsetzung der in Kapitel 1.4.3 genannten forstlichen Aufgabenstellungen wurde bei der Programmierung eines eigenen VRML-Clients umgesetzt (siehe hierzu Kapitel 3.3.3).
- Auf Sensorknoten und Eventhandling in der Szene wurde verzichtet.

- Es wurden vier eigene Forester Elementtypen (node types) definiert, die einen stärkeren Bezug zu Objekten und Methoden der forstlichen Praxis und Forschung herstellen sollen.
- Die Kerneigenschaften der Bäume sind durch wenige Parameter zu charakterisieren.
- Artenspezifische Eigenschaften des Erscheinungsbildes können definiert werden und den einzelnen Bäumen über einen Artnamen zugeordnet werden.

3.2.2 Aufbau der Bestandesbeschreibung

Die Bestandesbeschreibung kennt 5 Kategorien von Elementen:

1. Elemente im Kopfteil, welche nicht verändert werden dürfen. Hierzu gehören:
    a. VRML Head
    b. Prototypen der 4 Forester node types
    c. Variablendefinitionen in einem script node
2. Instanzen der Forester node types, die von anderen Forester-Elementen referenziert werden können. Dies sind:
    a. Species
    b. ShaftSystem
3. Elemente, die definiert sein müssen, deren Werte aber geändert werden können. Dies sind:
    a. Mindestens 1 Instanz des Forester node types GGInfo
    b. Definition des Default ShaftSystem mit der id -1
    c. Definition des Script Nodes BuildInfo
4. Beliebig viele Instanzen des Forester node type TREE.
5. Standard VRML Elemente:
    a. WorldInfo
    b. Background
    c. Shape zur Ground Definition

Im Folgenden werden die einzelnen Elemente einer Bestandesbeschreibung (Waldszene) beschrieben. Die Gliederung erfolgt analog ihres Bezugs zu einer realen Durchforstungssituation.

Die Beschreibung der einzelnen VRML-Elemente ist mit entsprechenden Farbmarkierungen versehen, die wie folgt zu lesen sind:

`WorldInfo { }` in hellblauer Farbe bezeichnet die Instanz eines VRML-Elementes vom Typ WorldInfo. Die geschweiften Klammern umfassen die Felder und Tochterelemente.

`info` in roter Farbe bezeichnet das Feld eines VRML-Elementes, welches zwingend definiert sein muss, dessen Werte aber geändert werden können.

`crown_values` in grüner Farbe bezeichnet das Feld eines VRML-Elementes, welches optional ist und dessen Werte geändert werden können.

`Id -1` in roter Farbe und Fettschrift bezeichnet das Feld und den Feldwert eines VRML-Elementes, welches zwingend mit genau diesem Wert definiert sein muss.

Die veränderlichen Werte eines Feldes werden in schwarzer Farbe angegeben.

`"Baum"`	gibt einen String-Wert an. Typ: `SFString`
`0.2345`	gibt einen Fließkommazahlenwert an. Typ `SFFloat`
`123`	gibt einen 32-Bit Ganzzahlwert an. Typ: `SFInt32`
`TRUE oder FALSE`	geben einen logischen Wert an. Typ: `SFBool`
`[ "1" "SM" "2.4" ]`	definiert den Wertebereich eines Feldes mit mehreren String-Werten. Die einzelnen Werte sind durch Leerzeichen getrennt. Typ: `MFString`

`0.2345 0.1 0.345` gibt ein Feld mit 3 Float-Werten an. Typ: `MFFloat`.

Im weiteren Verlauf dieser Arbeit werden folgende Kurzschreibweisen im Text verwendet:

TREE:status	für das status-Datenfeld des TREE-Elementes.
TREE:position=>y-Wert	für den y-Wert des position-Datenfeldes des TREE-Elementes.

3.2.3 Szene-Informationen

```
WorldInfo {
        title "1. Waldbestand"
        info [
        "Dieser Waldbestand ist"
        " Testbestand des Instituts für "
        " Forstliche Biometrie und Informatik"
        "Ökophysiologisches Model 2"
        "Groimp Model2 via HTTP Server"
        ]
}
```

Kurzcharakterisierung

Kategorie:
VRML-Standard

Pflichtelement:
nein

Vorkommen:
genau einmal

Werte veränderlich:
ja

Angezeigt durch:
alle VRML- Browser

spez. Forester Aktionen:
nein

referenziert durch:
-

`Worldinfo` ist ein VRML-Standardelement und kann allgemeine Informationen über die Szene enthalten. Der Inhalt des Elementes wird durch den VRML-Browser in einem Popup-Fenster angezeigt.

3.2.4 Der Bestand

3.2.4.1 Baum-Elemente

```
TREE {
        ID 1
        position 35.103 2.842 23.701
        stem_size 0.09 21.9 0.09
        crown_position 0 14.79 0
        crown_size 1.385 7.11 1.385

        crown_values [ 1 0.5 1 1 1 1 ]

        Species "Fichte"

        Shaftform 104
        stem_texture "fichte_7s.gif"
        crown_texture "fichte_7k.gif"

        status 1
        labels [ "Baum" "Alter" "BiomasseC" ]
        data   [ "1" "108" "0" ]

}
```

Kurzcharakterisierung

Kategorie:
Forester Node Type

Pflichtelement:
nein

Vorkommen:
beliebig oft

Werte veränderlich:
ja

Angezeigt durch:
alle VRML Browser

spez. Forester Aktionen:
ja

referenziert durch:
-

TREE ist das Hauptelement einer Forester-Bestandesbeschreibung. Jedes TREE-Element beschreibt einen Baum in der virtuellen Waldszene. Es wurde eigens als eigner Elementtyp (s. 3.2.8 Prototypen) entwickelt, um einen Baum mit möglichst wenigen einfachen Parametern beschreiben zu können.

Im Detail ist ein TREE-Element eine komplexe Zusammenfassung verschiedenen VRML-Standard-Elemente, welche die Darstellung eines Baumes bestimmen.

Die Eigenschaften eines Baumes werden durch die 13 Felder des TREE-Elementes bestimmt. Sie wurden so definiert, dass eine hohe Analogie mit forstlichen Kenngrößen besteht und die Konstruktion eines Bestandes einfach und intuitiv erfolgen kann. Nur die 4 Lage- und Größenparameter sind obligatorisch für die Darstellung. Die Reihenfolge der Felder ist beliebig.

Im Folgenden werden die einzelnen Felder mit ihren Bedeutungen und Abhängigkeiten näher beschrieben:

Der einfachste Baum:

Die Mindestanforderung für die Beschreibung eines Baumes im Bestand sind die 4 Größen- und Lageparameter. Als Maßstab wird eine Angabe in Metern angenommen. Es werden folgende Felder des TREE-Elementes zur Darstellung eines einzelnen Baumes verwendet:

`postion` (`MFFloat`) mit den x,y,z-Werten zur Lage des Baumes in der Szene. Dies definiert den Baumbasispunkt (Mittelpunkt des Stammfußes). Bezugsgröße ist der Szenenursprung.

`stemsize` (`MFFloat`) mit den x,y,z-Werten zur Skalierung des Stamms. Der x-Wert entspricht dem halben Durchmesser in Höhe 1.3, der y-Wert der Stammlänge.

`crown_position` (`MFFloat`) mit den x,y,z-Werten zur Lage der Krone. Bezugsgröße ist der Baumbasispunkt. Dies definiert den Kronenansatzpunkt.

`crown_size` (`MFFloat`) mit den x,y,z-Werten der Kronenausdehnung. Bezugspunkt ist der Kronenansatz. Der x-Wert entspricht dem Kronenradius an der dicksten Stelle, der y-Wert der Kronenlänge.

Achtung: Die Definition der Felder `stem_size` und `crown_size` erfordert die Angabe von drei Werten. Für die Darstellung der Krone und des Stammes im Browser werden aber nur der x-Wert für den Durchmesser und der y-Wert für die Höhe genutzt. Der zweite Durchmesser für die z-Achse wird an dieser Stelle ignoriert. Die y-Achse entspricht – gemäß der in VRML üblichen Definition der Achsen - dem Höhenwert der Elemente.

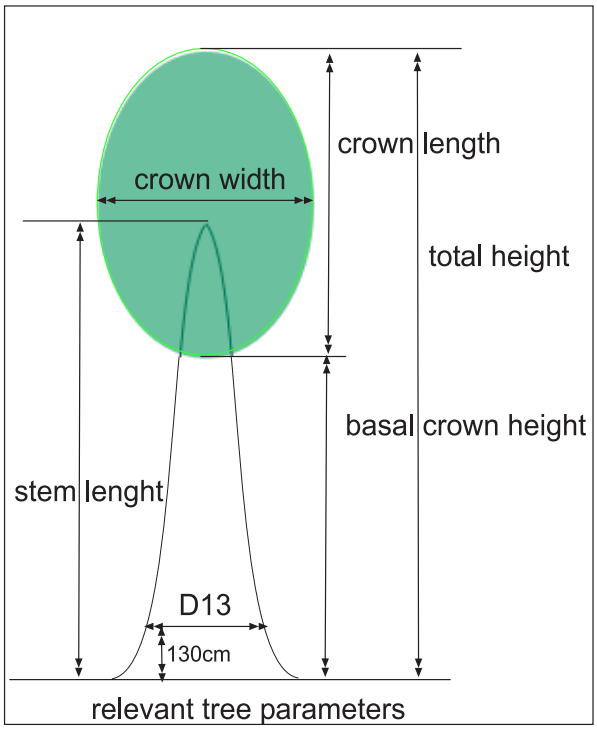
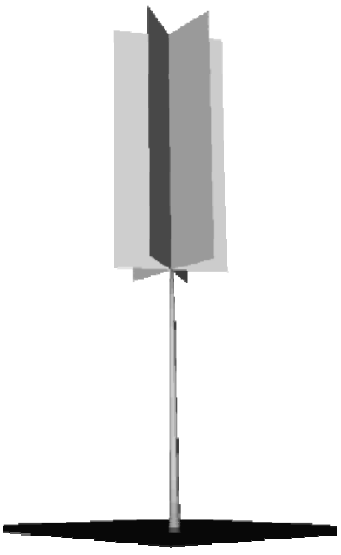


Abbildung 6: Die relevanten Parameter zur Beschreibung eines Baumes in einer Forester-Bestandesbeschreibung.

Komplexe Kronenform und Kronentexturen



Die Darstellung der Kronenform erfolgt mit einer baumspezifischen Textur mit transparentem Hintergrund. Die Textur zeigt die halbe Ansicht einer Krone mit dem gewünschten Habitus. Diese Textur wird auf 6 rechteckige Shape-Elemente aufgebracht, welche – ausgehend von der 9 Uhr-Position – im Uhrzeigersinn die Stammmittelachse symmetrisch umlaufen (Abbildung 7). Die Textur wird auf die Vorderseite und – gespiegelt – auf die Rückseite des Shapes aufgebracht. Dieses Vorgehen gewährleistet eine realistische Kronendarstellung aus allen Blickrichtungen mit vertretbarem Rechenaufwand.

Abbildung 7: Das Grundgerüst der Baumdarstellung mit dem Stamm und den 6 rechteckigen Shapes für die Kronendarstellung.

Im Standardfall wird die Größe der Shapes durch den Kronenradius und die Kronenlänge (`crown_size x,y-Werte`) bestimmt. Mit dem unterliegenden Shape wird auch die Kronentextur skaliert. Dieselbe Textur wird für Bäume mit unterschiedlichem Kronenansatz und Kronenlänge verwendet (Abbildung 8). Die Grenze der Skalierung wird so im Wesentlichen durch die Auflösung und das Länge-zu-Breite-Verhältnis der Texturdatei definiert. Sehr starke Skalierungen führen zu unbefriedigenden Darstellungsergebnissen.

Es gibt 2 verschiedene Möglichkeiten, die Kronentextur anzugeben. Die weitestgehende ist, die entsprechenden Felder für jedes `TREE`-Element zu benutzen. So kann für jeden Baum individuell der Habitus bestimmt werden. Die Felder hierfür lauten wie folgt:

Abbildung 8: Die Skalierung einer Kronentextur zur Darstellung verschiedener Kronenformen.

`stem_texture` (`SFString`) gibt die Stammtextur an. Dateien des Formats GIF [W3C: GIF 1989] und JPEG [ITU-T: JPEG 1992] sind erlaubt.

`crown_texture` (`SFString`) gibt die Kronentextur an. Aufgrund der notwendigen Transparenz sind nur Grafiken im GIF-Format [W3C: GIF 1989] mit transparentem Hintergrund erlaubt.

Der Name wird wie bei allen Texturen in der Forester-Bestandesbeschreibung ohne Pfad angegeben.

Für die zweite Möglichkeit der Texturangabe siehe unten (Punkt Referenz auf artenspezifische Definitionen).

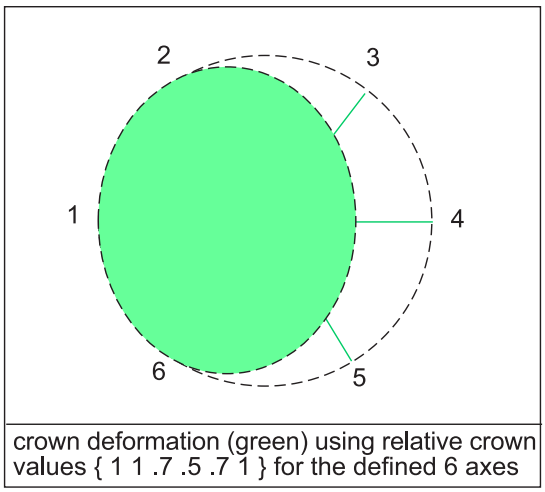


Abbildung 9: Die Darstellung von Kronendeformationen durch relative Skalierungen der Texturen entlang der 6 Darstellungsachsen.

Neben der symmetrischen Standarddarstellung der Krone wurde eine ergänzende Methodik zur Darstellung von deformierten Kronen implementiert. Abbildung 9 zeigt die 6 Darstellungsachsen der Kronentexturen in der Draufsicht.

`crown_value` (`MFFloat`) definiert für jede der Achsen einen individuellen Reduktionsfaktor [0..1], der die Kronentextur entlang dieser Achse staucht. Bezugsgröße ist der Kronenradius.

Die Methodik zur Darstellung der Kronendeformation ist Bestandteil des Virtual Forester Clients und wird von VRML-Standardbrowsern nicht unterstützt.

Stammform

`ShaftForm` (`SFInt32`) erlaubt die Referenzierung auf ein ShaftSystem-Element (siehe Kapitel 3.2.4.3) und somit auf den dort definierten Parametersatz zur Darstellung der Stammform (Siehe auch unten: Referenz auf artenspezifische Definitionen).

Referenz auf artenspezifische Definitionen

Das Konzept der artenspezifischen Definitionen wird in Kapitel 3.2.4.2 erläutert. Die Einbindung der entsprechenden Eigenschaften erfolgt mit einer Referenz auf ein entsprechendes Species-Element.

`Species` (`SFString`). gibt die Referenz auf ein `Species`-Element mit entsprechendem Wert im Description Feld. Das referenzierende `TREE`-Element erbt alle dort enthaltenen Eigenschaften.

Achtung: Das Feld `Species` darf nicht gemeinsam mit dem Schaftformfeld (`ShaftSystem`) verwendet werden.

Waldbaulicher Status

`status` (`SFInt32`) enthält den aktuellen waldbaulichen Status des Baumes.

Das Statusfeld wird vom Virtual Forester Client bei der Darstellung interpretiert. Es sind 5 Status-Werte vordefiniert und mit entsprechenden Aktionen versehen:

1- normaler Baum
2- Zukunftsbaum (grün markiert)
3- negative Auswahl (rot markiert)
4- gefällt
5- entfernt

Von Standardbrowsern in der Darstellung ignoriert, kommt dem Wert des Statusfeldes innerhalb einer forstlichen Simulation entscheidende Bedeutung zu (vergleiche hierzu auch Kapitel 3.3.3. Forstliche Tasks und 3.7.5.1 Auslesen der Bestandesszene).

Zusätzliche Informationen

Neben den Standardfeldern zur Beschreibung typischer Eigenschaften der Bäume ergibt sich je nach Herkunft und Verwendungszweck des virtuellen Bestandes der Bedarf, weitere Bauminformationen zu speichern. So werden zum Beispiel von einigen Wachstumsmodellen spezifische Baumparameter benötigt (Biomasse, Alter, Baumnummern etc.), die nicht aus den bisherigen Feldern abzuleiten sind. Um hier eine offene und anpassungsfähige Möglichkeit zu bieten, individuelle Baumdaten zu speichern, wurden 2 weitere Felder in das TREE-Element mit aufgenommen.

`labels` (`MFString`) und `data` (`MFString`) sind zwei Arrays mit einer beliebigen – aber identischen – Anzahl von Werten. Die Einträge in den beiden Feldern mit gleichem Index sind dabei jeweils ideell paarweise verbunden. Jeder Eintrag mit einem Index i im Feld `labels` enthält den Namen einer Wertekategorie und der Eintrag mit Index i im Feld `data` den dazugehörigen Wert. Alle Einträge werden als String – unabhängig von ihrem ursprünglichen Datentyp – angegeben. Die `labels/data`-Felder werden im vorliegenden Anwendungsbeispiel sowohl im Zusammenhang mit dem Wachstumsgenerator (siehe Kapitel 3.7.5.1) als auch mit der verbesserten Darstellung von Qualitätskriterien (siehe Kapitel 3.5.3) verwendet.

3.2.4.2 Artendefinitionen

```
Species    {
        Id 5
        Description "Fichte"
        Shaftform      104
        CrownTexture "tree_7_right.gif"
        StemTexture "wood_20.jpg"
}
```

Kurzcharakterisierung

Kategorie:
Forester Node Type

Pflichtelement:
nein

Vorkommen:
beliebig oft

Werte veränderlich:
ja

Angezeigt durch:
Nur Forester Client

spez. Forester Aktionen:
ja

referenziert durch:
TREE

Species ist ein Forester-Element-Typ mit 5 Feldern. Mit diesem Element können die Schaftform sowie die Kronen- und Stammtextur für Gruppen von TREE-Elementen angegeben werden. Die Felder ID (SFInt32) und Description (SFString) sind obligatorisch und dienen beide getrennt als Identifikationsmerkmal. Ein Species-Element kann von TREE-Elementen über die Description angesprochen werden.

Das Feld Shaftform (SFInt32) muss als Wert eine gültige Referenz auf ein Element des Typs ShaftForm enthalten.

CrownTexture (SFString) und StemTexture (SFString) sind Referenzen auf Grafikdateien mit Kronen- und Stammtexturen (siehe Kapitel 3.2.4.1).

3.2.4.3 Schaftformsysteme

```
ShaftSystem      {
        Id 102
        aVec [48.4 -151.5 -45 659.2 -799. 290]
        bVec [0.42 8.5 -29.8 48.2 -40.6 14.1 ]
}
```

Kurzcharakterisierung

Kategorie:
Forester Node Type

Pflichtelement:
nein

Vorkommen:
beliebig oft

Werte veränderlich:
ja

Angezeigt durch:
Nur Forester Client

spez. Forester Aktionen:
ja

referenziert durch:
TREE

ShaftSystem ist ein Forester-Element-Typ zur Definition erweiterter Möglichkeiten der Darstellung von Baumschaftformen. Das Feld ID (SFInt32) identifiziert das Schaftkurvensystem. Über die beiden Felder aVec (MFFloat) und bVec (MFFloat) mit jeweils 6 Werten wird ein Polynom 5ten Grades angegeben, welches die typische Schaftform einer Gruppe von Bäumen normiert beschreibt.

Dieses Polynom wird durch einen in den Forester Client implementierten Algorithmus zur Skalierung des VRML Shapes verwendet, welcher die Stammsektion eines TREE-Elementes darstellt. Die Skalierung erfolgt in Abhängigkeit vom Durchmesser an Position 1.3 und der relativen Position am Stamm (Abbildung 10).

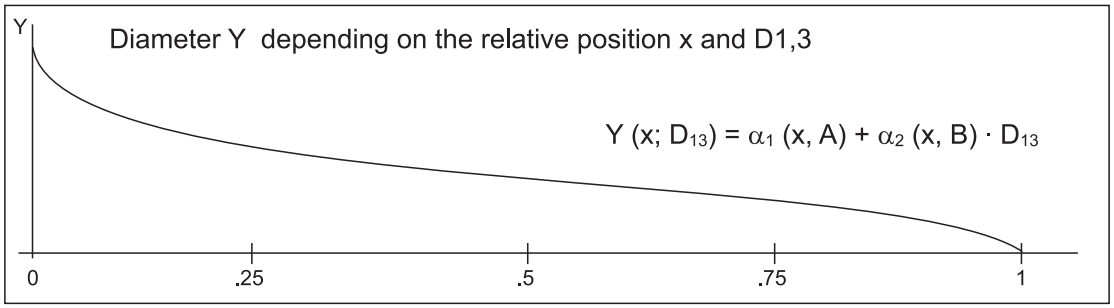


Abbildung 10: Beschreibung von bestandestypischen Baumschaftformen über ein Polynom 5ten Grades.

Grundlage dieser Form der Beschreibung der Schaftform ist eine Arbeit am Institut für Forstliche Biometrie und Informatik (Sloboda et al. 1998) zur Untersuchung von Schaftformen. Ziel der Arbeit war es, eine Methodik zu entwickeln, die es erlaubt, mit geringem Untersuchungsaufwand bestandestypische Schaftkurven mathematisch zu beschreiben.

3.2.5 Die Umgebung

3.2.5.1 Himmel

```
Background {
        skyColor [      0.0 0.0 0.0 ,
                        0.0 0.5 1.0 , 1.0 1.0 1.0 ]
        skyAngle [ 0.785, 1.571 ]
        groundColor [ 0.0 0.0 0.0,
                        0.2 0.1 0.0, 0.3 0.2 0.1 ]
        groundAngle [ 0.785, 1.57 ]
}
```

Kurzcharakterisierung

Kategorie:
VRML Standard

Pflichtelement:
nein

Vorkommen:
genau einmal

Werte veränderlich:
ja

Angezeigt durch:
alle VRML-Browser

spez. Forester Aktionen:
nein

referenziert durch:
-

`Background` ist ein VRML-Standardelement und beschreibt das Erscheinungsbild des Hintergrunds einer Szene. Die angegebenen Parameter sind nur eine Auswahl der zu Verfügung stehenden Möglichkeiten [vergl. Ames et al. 1997, Seite 451ff].

3.2.5.2 Terrain

```
Shape {
    appearance Appearance {
                material Material{}
          texture    ImageTexture {
                repeatT TRUE
                repeatS TRUE
                url "bewuchs1.jpg"
                }
    textureTransform TextureTransform {
                scale 30 30
                }
          }
    geometry ElevationGrid {
       creaseAngle 1
        xDimension 11
        zDimension 10
        xSpacing 5.000
        zSpacing 5.000
        solid TRUE
        height [
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1
0.0,0.0,0.3,0.3,0.3,0.3,0.3,0.3,2.3,2.3,0.5
0.0,0.3,0.3,0.6,0.6,0.6,0.3,0.3,3.0,3.3,0.7
0.0,0.3,0.2,0.2,0.2,0.3,0.2,0.3,3.3,3.6,1.0
0.0,0.3,0.2,0.3,0.3,0.4,0.4,0.4,3.4,2.0,1.0
0.0,0.0,0.2,0.3,0.4,0.4,0.5,0.5,2.5,3.0,0.8
0.0,0.0,0.2,0.3,0.3,0.4,0.4,0.3,3.8,4.0,0.5
0.0,0.0,0.2,0.3,0.3,0.2,0.3,0.3,2.0,3.6,0.3
0.0,0.3,0.0,0.0,0.0,0.0,0.0,0.0,1.5,2.7,0.4
0.0,0.0,0.2,0.2,0.3,0.3,0.3,0.3,0.0,0.4,0.2
      ]
    }
}
```

Kurzcharakterisierung

Kategorie:
VRML Standard

Pflichtelement:
nein

Vorkommen:
genau einmal

Werte veränderlich:
ja

Angezeigt durch:
alle VRML-Browser

spez. Forester Aktionen:
ja

referenziert durch:
-

Das Terrain wird durch ein VRML-Shape mit VRML-Unterelementen definiert. Die verwendeten Elemente gehören zum VRML-Standard und werden durch alle VRML-Browser dargestellt.

Besonders zu beachten ist hier die Möglichkeit der Anpassung der Bodentextur im Element `texture ImageTexture` mit den Feld `URL`. Hier kann eine Grafikdatei des Formates JPEG [ITU-T: JPEG 1992] oder GIF [W3C:GIF 1989] eingebunden werden. Zum Anzeigen der Textur muss die entsprechende Datei im Programmunterordner "Textures" vorhanden sein.

Kurz erwähnt werden soll auch das VRML-Element `geometry ElevationGrid` mit seinen Attributefeldern. Mit dem Feld `height` wird eine Matrix von Höhenwerten an-

gegeben, die den Untergrund definiert. Ausgehend vom Ursprung und der Ausrichtung der Szene laufen die Werte innerhalb einer Zeile entlang der x-Achse und die verschiedenen Zeilen bestimmen die z-Achse.

Für eine vollständige Beschreibung aller Möglichkeiten eines ElevationGrid-Elements siehe Ames et al. [1997], Seite 241ff.

3.2.6 Der Wachstumsgenerator

```
GGInfo {
        Name "GROIMP Model 2"
        Scriptpath
           "http://local:58080/open?model2.gs"
        IP "http://134.76.194.32"
        minAge 0
        maxAge 20
}
```

Kurzcharakterisierung
Kategorie: Forester Node Type
Pflichtelement: ja
Vorkommen: beliebig oft
Werte veränderlich: ja
Angezeigt durch: Nur Forester Client
spez. Forester Aktionen: ja
referenziert durch: -

`GGInfo` ist ein Forester-Element-Typ, mit dem Wachstumsgeneratoren für Prognoserechnungen in der Szene angegeben werden können. Auf Basis der `GGInfo`-Elemente wird im Netzwerkbetrieb (siehe Kapitel 3.1.1) das Auswahlmenü für den Wachstumsgenerator im Virtual Forester Client erstellt. Alle 5 Felder sind Pflichtfelder. `Name` (`SFString`) enthält den sichtbaren Menüeintrag. `Scripthpath` (`SFString`) enthält die vollständige URL (Berners-Lee et al. 2005) des Wachstumsgenerators; `IP` (`SFString`) die IP Adresse (mit http-Protokollvortrag des Wachstumsgenerators). `minAge` (`SFInt32`) und `maxAge` (`SFInt32`) definieren eine Spanne, die im Forester Client-Dialog (siehe Kapitel 3.3.5) für den Prognosezeitraum angegeben werden kann. Die Interpretation des Eingabewertes liegt aber im Ermessen des Wachstumsmodells (siehe Kapitel 3.7.5.1).

3.2.7 Kontrollelemente und Defaultwerte

3.2.7.1 Kontrolle des Szenenaufbaus

```
DEF BuildInfo Script {
        field SFBool useTaperSystem TRUE
        field SFBool adjustYPosition TRUE
}
```

Kurzcharakterisierung
Kategorie: Forester Node Type
Pflichtelement: ja
Vorkommen: genau 1 mal
Werte veränderlich: ja
Angezeigt durch: -
spez. Forester Aktionen: ja
referenziert durch: -

`BuildInfo` ist ein definierter Name eines VRML-Script-Elementes. Es dient zur Definition von Feldern, deren Werte durch den Virtual Forester Client interpretiert werden. Ziel dieser Felder ist es, dem Autor der Szene Zugriff auf die im Virtual Forester Client integrierten Programmfunktionen zu geben. Eine Aktion des Nutzers ist hierfür nicht notwendig.

VRML-Standardbrowser verwenden diese Felder nicht, was dem Verhalten des Virtual Forester Clients bei gesetzten FALSE-Werten entspricht.

Die zurzeit verwendeten Felder sind:

`useShaftSystem` (SFBool) entscheidet über die Nutzung der unter Kapitel 3.2.4.3 beschriebenen Methodik zur Berechnung der Stammform. Beim Wert FALSE werden alle definierten Elemente des Typs `ShaftSystem` ignoriert und die Stammform wird über eine in den Prototypen definierte Standardform dargestellt.

`adjustYPosition` (SFBool) kontrolliert die Berechnung der Höhenwerte (y-Werte) der Baumbasisposition in der Szene.

Ist der Wert TRUE, so gilt:

Die im `TREE`:`position(x,y,z)` angegebenen y-Werte werden relativ zum y-Wert des Terrains an der entsprechenden x,z-Position der Szene interpretiert. Das Terrain wird durch das `elevationGrid`-Element definiert (siehe Kapitel 3.2.5.2).

Zweck: Eine externe Berechnung der Höhenwerte der einzelnen Baumbasispunkte ist nicht notwendig. Werden die y-Werte des `position`-Feldes auf 0 gesetzt, wird eine bequeme Anpassung an das definierte Terrain automatisch durchgeführt.

Ist der Wert FALSE, so gilt:

Die in `TREE`:`position(x,y,z)` angegeben y-Werte werden vom Ursprung der Szene aus gerechnet. Eine besondere Anpassung an das Terrain erfolgt nicht.

Zweck: Sind die Höhenwerte der Baumbasispunkte vorhanden, kann auf eine Anpassung an das Terrain verzichtet werden. Dies beschleunigt erheblich die Erstdarstellung einer Bestandesszene.

3.2.7.2 Defaultdefinition der Stammform

```
DEF DefaultSystem TaperSystem    {
        Id -1,
        aVec [48.4 -151.5 -45 659.2 -799. 290]
        bVec [0.42 8.5 -29.8 48.2 -40.6 14.1 ]
}
```

Kurzcharakterisierung

Kategorie:
Forester Node Type

Pflichtelement:
ja

Vorkommen:
genau 1 mal

Werte veränderlich:
teilweise

Angezeigt durch:
nur Forester Client

spez. Forester Aktionen:
ja

referenziert durch:
-

`DefaultSystem` ist ein definierter Name für ein Element vom Typ `ShaftSystem` (siehe Kapitel 3.2.4.3) mit dem obligatorischen Wert -1 für das Feld `ID`. Das so definierte Schaftformsystem wird immer dann verwendet, wenn weder direkt im Baumelement (`TREE`) noch über eine Artendefinition (`Species`) ein `ShaftSystem` angegeben ist.

3.2.8 Kopfteil

Wie im Kapitel 3.2.2 beschrieben enthält eine Bestandesbeschreibung verschiedene Elemente, die vom Autor einer Szene nicht verändert werden dürfen und in jede Bestandesbeschreibung kopiert werden müssen. Aus Platzgründen werden diese Teile an dieser Stelle nicht detailliert beschrieben. Sie finden den vollständigen Kopfteil im Anhang 10. Hier nur eine kurze Übersicht der Bestandteile und ihrer Bedeutung:

- VRML Head: Mit dieser Zeile muss jede VRML-Szene beginnen. Der Head enthält die verwendete VRML-Version [Web3D:VRML 1997] und die Zeichenkodierung UTF8 [Yergeau 2003].
- Prototypes: VRML-Prototypes enthalten die Definitionen eigener Element-Typen. In diesem Fall sind dies die oben beschriebenen Forester Element-Typen. Ihre Prototypen dürfen nicht verändert werden, da sonst die Verwendung im Virtual Forester gefährdet wird.
- Variablendefinitionen: Die in einem Script-Element definieren Felder werden im Virtual Forester benötigt.

3.2.9 Weitere VRML-Objekte

Eine Bestandesszene kann weitere VRML 2.0-konforme Elemente enthalten. Zu beachten ist aber, dass dynamische Elemente, Skripte, Sensor- und Eventknoten nur auf dem Einzelarbeitsplatz angezeigt werden können (siehe Kapitel 3.3.5).

3.3 Virtual Forester Client (VRC)

Der Virtual Forester Client Version 0.9 (VRC) wurde am Institut für Forstliche Biometrie und Informatik nach Maßgaben des Autors von Marc Eilhardt (Graphische Programmierung und GUI) und Arkadius Rusin (Netzwerkprogrammierung) programmiert. Der Code unterliegt der GNU General Public License [GNU:GPL 1991]. Die verwendeten Avatare wurden bereitgestellt von Paul McLain.

3.3.1 Aufgaben des Clients

Wie in der in Kapitel 3.1 gegebenen Übersicht der Systemkomponenten beschrieben, ist der Virtual Forester Client das Programm, welches das Graphische User Interface (GUI) bereitstellt. Im Groben soll das GUI den Nutzerinnen folgende 3 Funktionen bereitstellen:

- Visualisierung der virtuellen, begehbaren Waldszene mit allen in der Bestandesbeschreibung enthaltenen Eigenschaften des Waldes. Dies beinhaltet die freie Navigation innerhalb der VRML-Welt, Dialoge zum Anzeigen von Baum- und Bestandesinformationen, sowie Möglichkeiten zur Veränderung der Ansicht, wie sie in VRML-Browsern üblich sind.
- Manipulation der Bestandesbeschreibung gemäß der in Kapitel 1.4.3 genannten Anforderungen zur Durchführung von virtuellen Durchforstungen. Dies beinhaltet das Auszeichnen und Fällen von Bäumen.
- Schnittstellen und Dialoge zur Ansprache weiterer Systemkomponenten. Dies umfasst die Dialoge zum Lesen und Speichern von geänderten Bestandesbeschreibungen im Einzelplatzbetrieb bis hin zum Starten von Simulationsläufen und zur Kommunikation zwischen den Clientprogrammen im Serverbetrieb.

In den folgenden Kapiteln werden die technischen Grundlagen und Nutzungsmöglichkeiten des Virtual Forester Clients beschrieben. Wie in Kapitel 3.1.1 erwähnt wird zwischen Einzelplatz- und Serverbetrieb unterschieden. Für den Virtual Forester Client gilt, dass die Basisfunktionalität des Einzelplatzbetriebes auch im Serverbetrieb vollständig zu Verfügung steht und um weitere Funktionalitäten erweitert wird.

3.3.2 Technische Grundlagen

Die Programmierung des Virtual Forester Clients erfolgte in C++ unter Verwendung der Bibliothek Coin3D-Version 2.1.1 von System in Motion für die graphische 3D-Darstellung.

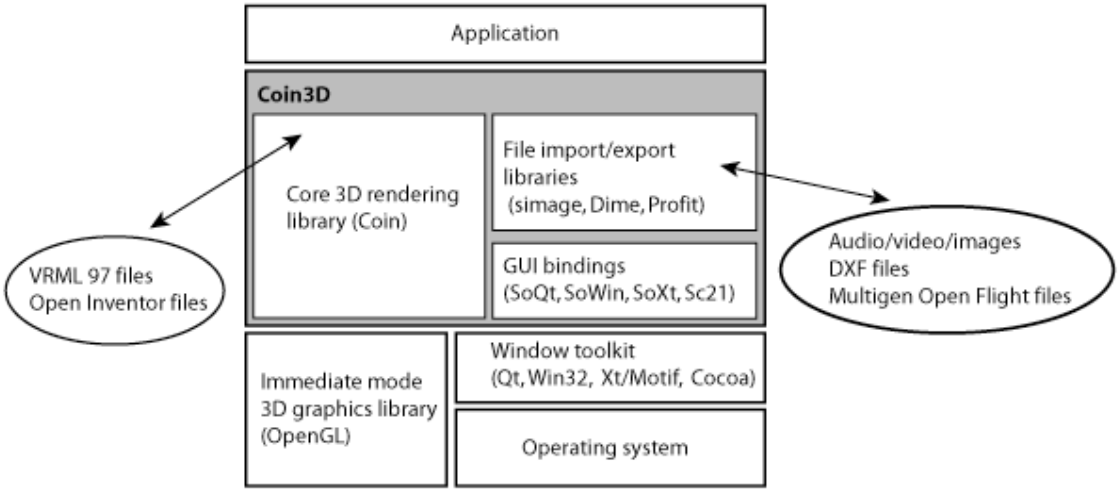
Die Eigenschaften von Coin3D werden von System in Motion [System in Motion: Coin3D] wie folgt beschreiben:

"Coin3D is a high-level 3D graphics toolkit for developing cross-platform real-time 3D visualization and visual simulation software.

*Coin3D is portable over a wide range of platforms: any UNIX / Linux / *BSD platform, all Microsoft Windows operating systems, and Mac OS X.*

Coin3D is built on OpenGL and uses scene graph data structures to render 3D graphics in real-time. Basic import, rendering, and interaction with a 3D object can be implemented in just a few lines of code, and programmer efficiency is greatly increased compared with programming directly with OpenGL. OpenGL code and Coin3D code can coexist in the same application, which makes gradual migration from OpenGL to Coin3D possible.

Coin3D is fully compatible with SGI Open Inventor 2.1, the de facto standard for 3D visualization and visual simulation software in the scientific and engineering community. Additional features in Coin3D include VRML97 support, 3D sound, 3D textures, and parallel rendering on multiple processors. (…)



The core 3D rendering library Coin uses scene graph data structures to store and render graphics. Its data-driven design (redraws are only scheduled if the scene data changes) makes Coin3D well suited for user interface-based applications (e.g. scientific and engineering visualization applications), since it does not claim excessive CPU resources.

Being only a 3D rendering library, Coin needs a user interface binding to be able to open windows, handle user input etc. GUI Bindings are available for the native Microsoft Windows GUI (SoWin), Trolltech's QT (SoQt), Xt/Motif on X Windows (SoXt), and the native Mac OS X GUI (Sc21).”

Zur Wahrung der Plattformunabhängigkeit wurde für das Graphische User Interface die Bilbiothek QT Version 2 von Trolltech verwendet. Hierzu schreibt System in Motion [System in Motion: SoQt]:

"Qt is a C++ toolkit for multiplatform development of 2D user interfaces, and also includes other functionality to help programmers write multiplatform applications. Qt is currently available on X11-based systems (UNIX, Linux and BSDs), MSWindows, Mac OS X and embedded systems. Development is done with Qt version 2, but SoQt is also compatible with Qt version 3".

Für die Anbindung von coin3D an Qt empfiehlt System in Motion die Bibliothek SoQt [System in Motion: SoQt].

"SoQt is a library which provides the glue between Systems in Motion's Coin high-level 3D visualization library and Troll Tech's Qt 2D user interface library. (…)

By using the combination of Coin, Qt and SoQt for your 3D applications, you have a framework for writing completely portable software across the whole range of UNIX, Linux, Microsoft Windows and Mac OS X operating systems. Coin, Qt and SoQt makes this possible from a 100% common codebase, which means there is a minimum of hassle for developers when working on multiplatform software, with the resulting large gains in productivity.

SoQt, like Coin and Qt, provides the programmer with a high-level application programmer's interface (API) in C++. The library primarily includes a class-hierarchy of viewer components of varying functionality and complexity, with various modes for the end-user to control the 3D-scene camera interaction."

Im Virtual Forester Client wurde für Windows die Version 1.0.3 verwendet.

Für alle drei Bibliotheken wurde die unter GNU General Public License [GNU: GPL 1991] stehende Version genommen [siehe System in Motion: Coin Free Edition, Trolltech: Qt und System in Motion: SoQt].

3.3.3 Forstliche Tasks

Bei der Definition der Grundprämissen einer Bestandesbeschreibung (vergl. Kapitel 3.2.1) wurde bereits erwähnt, dass aus Gründen der Übersichtlichkeit auf Sensorknoten, Eventhandling und dynamische Skriptprogrammierung in der Bestandesszene verzichtet wurde. Vielmehr wurden bestimme Funktionalitäten bei der Darstellung und der Behandlung von VRML-Elementen bei der Programmierung des Virtual Forester Clients in einer Form umgesetzt, die sich von einem VRML-Standardbrowser unterscheidet. Die so erreichte einfache Les- und Schreibbarkeit der Bestandesbeschreibung erfordert im Gegenzug die Existenz der aufgeführten Pflichtelemente (siehe Kapitel 3.2.2ff) in der Beschreibung, um die Funktionstüchtigkeit des Forester zu gewährleisten. Zudem stehen bestimmte Funktionen nur im Virtual Forester Client zur Verfügung. An dieser Stelle erfolgt eine kurze Auflistung der entsprechenden spezifischen Virtual Forester-Funktionen. Eine detaillierte Beschreibung folgt in den nächsten Kapiteln.

Bei den Basisfunktionalitäten sind dies:

- Markieren der Bäume gemäß des TREE:status
- Fällen aller Bäume einer TREE:status – Gruppe
- Anzeigen aller Bäume einer TREE:status – Gruppe

- Darstellen der Schaftform über ShaftSystem
- Angabe von Texturen im "texture" Ordner
- Automatisches Anpassen der TREE:position y-Koordinaten auf den Bestandesuntergrund (definiert durch ein elevationGrid-Element)

Im Client-Server Betrieb (vergl. Kapitel 3.3.5) sind dies:

- Szenen-/Sessionmanagement
- Auslieferung von Texturen
- Weitergabe vom TREE:status Änderungen
- Weitergabe der Avatarposition
- Kommunikationskomponenten zwischen den Clientrechnern
- Auslesen der GGInfo-Elemente

3.3.4 Basis-Funktionalität

Hauptfunktionalität des Virtual Forester Clients ist die eines VRML-Browsers, d. h. die Visualisierung einer begehbaren VRML-Welt. Als spezielles GUI für Forester-Bestandesbeschreibungen hat das Clientprogramm die oben aufgeführten besonderen Methoden implementiert. Sie dienen zur Abbildung von in der Forstwirtschaft üblichen Situationen. Abbildung 11 zeigt die Darstellung eines Nadelwaldbestandes mit markierten, unmarkierten und gefällten Bäumen. Der Zustand kann interaktiv durch Mausklick geändert werden. Neben der Visualisierung erfolgt eine Manipulation des status-Feldes des jeweiligen TREE-Elements (vergl. 3.2.4.1: Waldbaulicher Status). Über ein spezielles Menü kann die Anzeige der Bäume für die 4 Hauptstatusgruppen (normal, rot markiert, grün markiert, gefällt) einzeln an- und abgeschaltet werden. Alle Bäume – auch gefällte und abtransportierte – werden in der Szene belassen. So kann in der Bestandesbeschreibung jederzeit der Ausgangszustand wiederhergestellt werden. In Abbildung 12 werden die Bäume mit der im TREE-Prototypen definierten Standardform für den Schaft gezeigt. Abbildung 12 zeigt Bäume mit stark überakzentuiertem Stammfuß. Hierzu wurde ein ShaftSystem-Element verwendet, welches einen Parametersatz für eine **Stamm-Durchmesserfunktion** definiert (siehe Kapitel 3.2.4.3). Die im Virtual Forester Client integrierte Durchmesserfunktion errechnet Skalierungswerte für das Shape-Element, welches den Baumstamm darstellt.

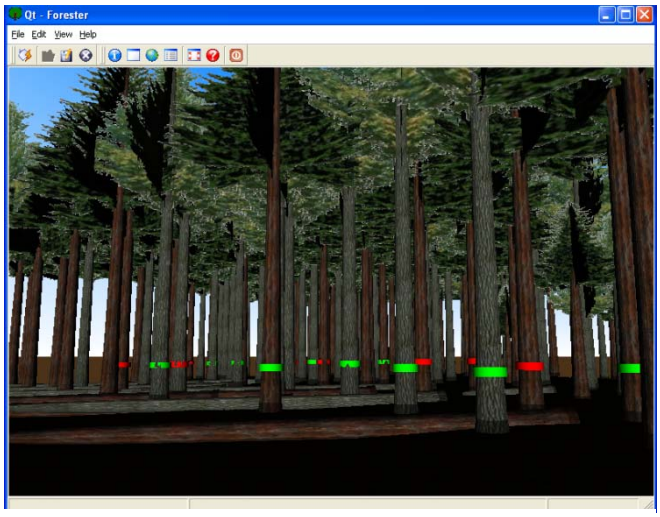


Abbildung 11: Markierte und gefällte Bäume im Virtual Forester Client. Der Zustand des Baumes wird im status-Feld festgehalten.

Die **Texturen** für den Stamm und die Baumkrone bilden im Wesentlichen die visuellen Charakteristika der Bäume. Alle Texturen werden auf die Oberfläche der tragenden Shape-Elemente gemappt und mit ihnen skaliert. Über verschiedene Texturen werden vielfältige Kronenformen dargestellt, ohne den Grundaufbau eines TREE-Elementes zu verändern. Abbildung 13 zeigt eine asymmetrische Ausbildung der Kronen zweier eng stehender Bäume, wie sie durch die Skalierung der 6 Textur tragenden Kronen-Shapes erzeugt wird. Die entsprechende Methode ist programm-integriert und wird durch die individuellen Werte des TREE:crown_value-Feldes parametrisiert (siehe Kapitel 3.2.4.1: Komplexe Kronenform).

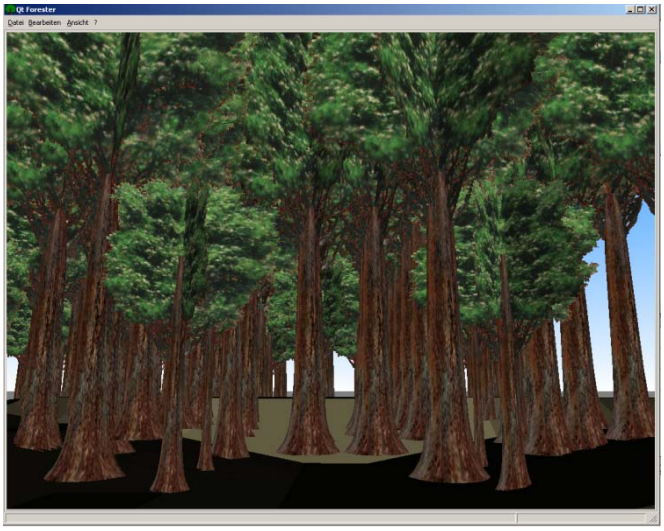


Abbildung 12: Beispielszene: Bäume mit überakzentuiertem Stammfuß. Die Schaftform kann über TaperSystem definiert werden.

Der **transparente Hintergrund** der Kronentexturen erlaubt die Durchsicht durch die Baumkronen auf dahinterliegende Objekte. Alle Texturen müssen im Unterordner "Textures" innerhalb des Programmordners liegen.

Im Virtual Forester Client ist die **Objektkollision** ausgeschaltet, d. h. die Navigation in der Szene wird durch Stamm- und Kronen-Shapes nicht behindert.

Wie in Kapitel 3.2.7.1 beschrieben kann wahlweise die **vertikale Position der Bäume** automatisch dem Terrain angepasst werden oder direkt angegeben werden. Im Falle der automatischen Anpassung wird der y-Baumkoordinate durch das Client-Programm automatisch derjenige Höhenwert zuaddiert, welcher sich an der jeweiligen Baumposition durch das definierte elevationGrid errechnet. Werden die vertikalen Baumpositionswerte auf Null gesetzt, wird der gesamte Bestand dem Terrain automatisch angepasst.

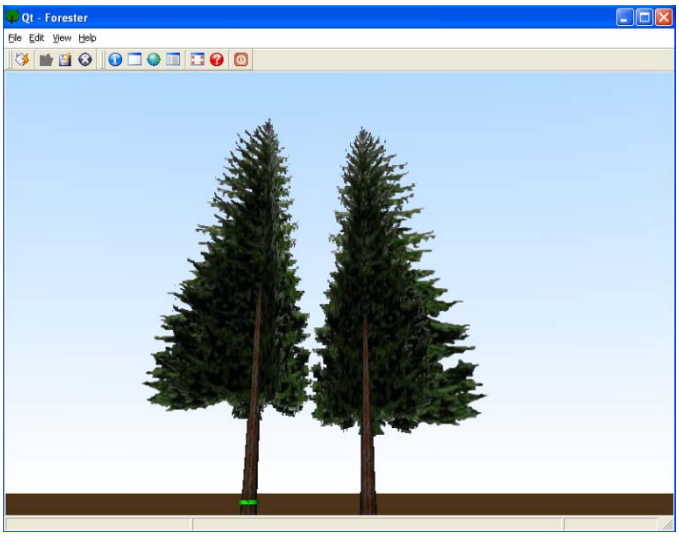


Abbildung 13: Beispielszene: Kronendeformation zweier eng stehender Bäume. Die Skalierung der Krone erfolgt über crown_values [1 1 .7 .5 .7 1] für den linken und crown_values [.5 0.7 1 1 1 .7] für den rechten Baum.

Im anderen Fall werden die Baumpositionswerte direkt bezogen auf den Ursprung der Szene umgesetzt, d. h. die Baumpositionen müssen bereits bei der Erstellung der Bestandesbeschreibung mit dem Terrain verrechnet werden, da sonst Bäume im Untergrund versunken oder in der Luft schwebend dargestellt werden.

3.3.5 Client-Server-Betrieb

Zusätzlich zu den Basisfunktionalitäten stellt der Virtual Forester Client Funktionen zur Verfügung, die nur im gemeinsamen Betrieb mit der Serverkomponente genutzt werden. Unter Verwendung des Elan Sim Servers können mehrere Virtual Forester Clients über das Netzwerk gleichzeitig eine Waldszene nutzen.

Die Clientprogramme sind so miteinander verbunden, dass alle Aktionen – wie z. B. das Fällen eines Baumes – in allen Ansichten gleichzeitig ausgeführt werden. Die technische Realisierung der Verbindung wird im Kapitel 3.4 erläutert.

Im Virtual Forester Client stehen verschieden Dialoge und Methoden für das kollaborative Arbeiten zu Verfügung.

Jedes Forester Client-Programm muss sich beim Server anmelden. Hierzu werden über einen Dialog der gewünschte Server sowie ein Username und ein Passwort nachgefragt. Jedem angemeldeten Forester wird eine Liste der vom Server bereitgestellten Szenen und Sessions angezeigt. Jeweils der erste Forester, der eine Bestandesbeschreibung vom Server wählt, eröffnet als Master eine neue Session. Andere Forester können dieser Session beitreten.

Jeder Forester in einer Session wird in den beigetretenen Clientprogrammen durch einen Avatar dargestellt, der bei Eintritt in eine Session durch die Nutzerin gewählt wird. Positionsänderungen der einzelnen Teilnehmer werden beständig in allen beigetretenen Programmen aktualisiert. Dasselbe gilt für Statusänderungen der Bäume. Die Avatare (z. B. Abbildung 14) dienen neben der besseren Orientierung, auch als Vergleichsmaßstab zur Einschätzung von Größen.

Abbildung 14: Avatar im Virtual Forster Client (©Paul McLain)

Zur Kommunikation der Studierenden untereinander ist eine Chat-Komponente integriert. So können die Teilnehmerinnen an einer Session sich über die aktuelle Situation und Aufgabenstellung austauschen. Bei Bedarf kann der Empfang von Chat-Nachrichten durch die Nutzerin geblockt werden.

Das Speichern einer lokalen Kopie der aktuellen Bestandesbeschreibung ist von jedem teilnehmenden Clientprogramm möglich. Der Dialog zum Ansprechen des Wachstumsgenerators über den Elan Sim Server ist nur für den Session Master aktiviert. Ansprechbar sind alle mit einem GGInfo-Element in der Bestandesbeschreibung angegebenen Wachstumsgeneratoren (siehe Kapitel 3.2.6). Die Arbeit der anderen Virtual Forester Clientprogramme wird davon nicht betroffen.

Dieses Vorgehen hat 3 Gründe:

Es können nur Wachstumsgeneratoren angesprochen werden, welche die Erstellerinnen der Bestandesszene für diese freigegeben haben. Es obliegt somit diesen, sicherzustellen, dass die Bestandesszene alle Informationen enthält, die das angesprochene Wachstumsmodell benötigt (vergl. Kapitel 3.2.4.1: Zusätzliche Informationen).

Das Berechnen einer Prognose der Bestandesentwicklung kann je nach Komplexität des verwendeten Modells einige Zeit benötigen. Die Prognoseberechnung ist daher von der Visualisierung der Bestandesszene entkoppelt. Nach Anforderung einer Wachstumsprognose durch den Session Master können die Session-Teilnehmerinnen mit der Arbeit in der Ursprungsszene fortfahren. Liegt irgendwann das Ergebnis einer Prognoseberechnung vor, wird diese in die Serverliste als neue Szene integriert. Durch dieses Vorgehen ist es möglich – ausgehend von einer Bestandesszene – mehrere Prognoseszenarien zu berechnen.

Das vorliegende System wurde dafür entwickelt, das kollaborative Arbeiten von mehreren Studierenden zu unterstützen. Zur Lösung von Aufgabenstellungen bedarf es hierzu der Diskussion und Abstimmung zwischen den Teilnehmerinnen. In bestimmten Lernszenarien ist auch der Einsatz einer Tutorin denkbar. Die Begrenzung des Dialogs zur Prognoseberechnung auf den Master soll ein mögliches tutorielles Szenario unterstützen. So kann z. B. eine Tutorin sicherstellen, dass als Grundlage für eine spätere Interpretation der Prognoseergebnisse eine genaue und sinnvolle Definition der Ausgangssituation erfolgt, bevor die Prognoseberechnung gestartet wird.

3.4 Elan Sim Server (ESS)

Der Elan Sim Server (ESS) Version 0.9 wurde am Institut für Forstliche Biometrie und Informatik unter Leitung des Autors von Arkadius Rusin und Marc Eilhardt programmiert. Der Code unterliegt der GNU General Public License [GNU:GPL 1991].

3.4.1 Aufgaben des Servers

Wie in der in Kapitel 3.1 gegebenen Übersicht der Systemkomponenten beschrieben, ist der Elan Sim Server die Programmkomponente, die mehrere Virtual Forester Clients zur kollaborativen Arbeit in einer Bestandesszene miteinander verbindet. Das Servermodul erfüllt dabei folgende Funktionen:

- Die Bereitstellung und Verwaltung von Bestandesszenen für die Nutzung durch die Clientprogramme. Hierzu gehören der Anmeldeprozess sowie das Ausliefern der Bestandesbeschreibung mit allen benötigten Texturen, die auf dem Clientprogramm nicht vorhanden sind.
- Die Verwaltung von Sessions, an denen mehrere Clientprogramme beteiligt sind. Hierzu gehören die An- und Abmeldung von Clientprogrammen in der Session, die Übertragung von Aktionen und Bewegungen zwischen den Clients sowie die Übertragung der Kommunikation über die Chat-Komponente.
- Die Anbindung von externen Wachstumsgeneratoren und die Verwaltung der angeforderten Prognoseberechnungen. Hierzu gehören das Versenden der Bestandesbeschreibung an die Wachstumsgeneratoren, die Abfrage der Prognoseergebnisse sowie deren Integration in die Szenenliste.

In den folgenden Kapiteln werden die technischen Grundlagen des Elan Sim Servers sowie die Kommunikation mit den Virtual Forester Clientprogrammen beschrieben.

3.4.2 Technische Grundlagen & Konfiguration

Die Serverkomponente wurde unter SuSE Linux 7.3 in C++ erstellt und zusätzlich unter SuSE Linux 8.1 kompiliert und getestet. Die Netzwerkfunktionalität wurde unter Verwendung der Socket API verwirklicht.

Die Konfiguration des Servers erfolgt über eine Konfigurationsdatei mit Namen "elansimconf", die im Programmordner des ESS vorliegen muss. In dieser Datei werden für den Server notwendige Parameter gespeichert. Die Struktur dieser Datei ist festgelegt. Es dürfen bis zum Ende des Parameterblocks keine Leerzeilen vorkommen. Eine Beispielkonfiguration sieht wie folgt aus [siehe Rusin 2004]:

Bsp. 3.4-1

```
//this is the configfile for the Elan Sim Server
//each value must be terminated by a '#'
//followed by the value description name
//for example 15#max_number_of_clients
//changes take effect after the server is restarted
//comments can be written in C++-style
// no empty lines are allowed
2048#buffer_size                //don't decrease this value!!!
15#max_number_of_clients
50#max_number_of_sessions
100#max_number_of_scenes
100#max_no_of_pending_clients
10#queue_length
```

`2048#buffer_size` gibt die Größe des Buffers für die Eventschleife an und darf 2048 nicht unterschreiten (vergl. 3.4.3).

`15#max_number_of_clients` setzt die Anzahl der erlaubten Clients in einer Session auf 15.

`50#max_number_of_sessions` setzt die maximale Anzahl der Sessions, die eröffnet werden können, auf 50.

`100#max_number_of_scenes` definiert die maximale Anzahl von Szenen in der Liste auf dem Server.

`100#max_no_of_pending_clients` setzt die maximale Zahl von Pending Clients. Ein Pending Client ist ein angemeldetes Clientprogramm, welches noch keine Session eröffnet hat bzw. keiner Session beigetreten ist.

`10#queue_length` definiert die Länge der Eventschlange (vergl. Kapitel 3.4.3).

Die Liste der zu Verfügung stehenden Szenen wird über die Datei "elansimscenes" angegeben, die ebenfalls im Programmordner des ESS vorliegen muss. In ihr werden zeilenweise die Namen der VRML-Dateien angegeben, die vom Server den Clients zur Verfügung gestellt werden sollen [siehe Rusin 2004]. Alle angegebenen Dateien müssen im Ordner "Scenes" unterhalb des Programmordners vorhanden sein.

Das Programm auf dem Server muss von der Kommandozeile aus mit Schreibrechten auf das Programmverzeichnis (mit allen Unterverzeichnissen und Dateien) gestartet werden. Das Fehlen einer der Konfigurationsdateien beim Start des ESS-Programms führt zur Ausgabe einer entsprechenden Fehlermeldung und zum Abbruch.

3.4.3 Forester Event-Behandlung

Kernstück des Elan Sim Servers ist das von den Programmierern entwickelte Flagsystem zur Übertragung und Behandlung von Forester Events. Ausgangspunkt war die Forderung, die in Kapitel 3.3.3 beschriebene Statusänderung der virtuellen Bäume – z. B. Markieren oder Fällen eines Baumes – auf allen angeschlossenen Clientprogrammen simultan auszuführen – auch bei geringeren Bandbreiten wie bei ISDN-Verbindungen.

Auf ein beständiges Austauschen der gesamten VRML-Szene wurde daher verzichtet. Vielmehr wurde das – bereits bei der Bestandesbeschreibung und dem Virtual Forester Client verfolgte – Konzept der Definition typischer forstlicher Tasks (vergl. Kapitel 3.2.1 und 3.3.3) aufgegriffen und in ein spezielles Flagsystem umgesetzt.

Der Ablauf ist wie folgt:

1. Ausgelöst durch eine spezielle Aktion im Clientprogramm wird ein Flag an die Eventbehandlung zum Server gesendet.
2. Der Server arbeitet die Events entlang ihrer Reihung in der Eventbehandlungsschlange ab. Gemäß der Kodierung des Events werden die Eventdaten ausgelesen.
3. Je nach Eventtyp sind jetzt mehrere Schritte möglich:
    a. Sollte für das Event eine serverseitige Aktion erforderlich sein, wird diese zuerst ausgeführt. Eventuell resultierende Flags werden erzeugt und an den Anfang der Eventbehandlung gesetzt. So kann ein auslösendes Event eine Reihe von resultierenden Events bedingen.
    b. Ist für das Event eine clientseitige Aktion erforderlich, bekommen die zuständigen Clientprogramme das entsprechende Flag mit den Eventdaten vom Server gesendet und führen die entsprechende Aktion aus. Je nach Eventtyp wird das Flag an den Masterclient, das auslösende Clientprogramm oder an alle gesendet.

Die Eventbehandlungsschlange wird aus den von den Clientprogrammen gesendeten oder vom Server erzeugten Flags mit Ihren Eventdaten gebildet. Jedes der definierten Flags folgt dem Schema:

2 Byte für die Flagkennung

4 Byte für die Größe des Datenbereichs

Flexibler Bereich für die Eventdaten

Folgende Flags sind definiert:

Name	Kennung	Datenbereich
logout	1	null
login	2	username,#,passwort
listsrequest	4	null
lists	8	4byte sceneId, 4byte sessionId, <name>, #, ., #

scenerequest	16	4 byte sceneId, 4 byte sessionId, <name>
scene	32	<scene>
cur_scenerequest	64	Null
movement	128	4byte x-pos, 4byte y-pos, 4byte z-pos, 4byte orientation
event	256	4byte treeId, 4byte partId
chat	512	<chatnachricht>
master	1024	<id>
error	2048	<errormessage>
new_client	4096	id, avatar_nr, pos, <name>

Tabelle 3: Die für die Forester-Event-Behandlung definierten Flags zur Kommunikation zwischen dem Serverprogramm und den Client-Programmen.

3.4.4 Verwaltung von Szenen

Die Verwaltung der bereitgestellten Waldszenen und der daraus abgeleiteten Sessions erfolgt durch den Server. Eine Waldszene besteht aus der Bestandesbeschreibung sowie den in der Beschreibung verwendeten Texturbildern.

Hierbei unterscheiden wir 3 Typen von Szenen:

- Die Ursprungsliste enthält diejenigen Szenen, welche im Konfigurationsfile des Servers aufgeführt sind. Diese Szenen stehen jedem Client zur Bildung einer neuen Session zur Verfügung. Die Bestandesbeschreibungen und die Texturen müssen in den Unterordnern des Programmordners "Scenes" bzw. "Textures" liegen.
- Die Sessionszenen werden gebildet, wenn ein Clientprogramm eine Session eröffnet und eine der Ursprungsszenen bearbeitet. Die aktuelle Szene der Session wird nicht vom Server gespeichert, sondern vom Master Client bereitgestellt. Kommt ein neuer Client hinzu, wird die aktuelle Szene über den Server vom Masterclient abgefragt und an das anfragende Clientprogramm ausgeliefert.
- Die Prognoseszenen werden gebildet, wenn auf Anforderung eines Master-Clients der ESS eine aktuelle Bestandesbeschreibung an einen Wachstumsgenerator schickt und als Antwort eine Bestandesbeschreibung mit der Prognoseberechnung enthält.

Wählt die Userin mit der clientseitigen Dialogbox eine der Szenen aus, wird die Bestandesbeschreibung vom ESS ausgeliefert. Außerdem werden alle zu der Szene gehörigen Texturdateien ausgeliefert, die auf dem Clientrechner nicht bereits vorhanden sind. Da Sessionszenen und Prognoseszenen von den Ursprungsszenen abgeleitet sind, kann so über das Servermodul sichergestellt werden, dass ihre Darstellung durch die Clientprogramme gewährleistet ist.

3.4.5 Anbindung von Wachstumsgeneratoren

Wichtigste Zielstellung für die Anbindung des Wachstumsgenerators war es, eine offene Schnittstelle zu entwickeln, die es erlaubt, unterschiedliche Programme und Modellansätze anzubinden. Dieses Ziel basiert auf dem Hauptgedanken dieser Arbeit, den Studierenden verwendete Modellansätze des Waldwachstums bewusst, transparent und in ihren Auswirkungen erfahrbar zu machen. Bedingt durch die teilweise – je nach Modellansatz – erhebliche Rechenleistung, die für eine Wachstumsprognose nötig ist, wurde ein netzbasiertes verteiltes System geplant. Für die Kommunikation zwischen dem ESS und dem Wachstumsgenerator wurde HTTP [Fielding et al.1999] vorgegeben.

Die für diese Arbeit entwickelte Schnittstelle arbeitet in 2 getrennten Schritten:

Schritt 1. Ausgelöst wird die Prognoseberechnung durch den Masterclient einer Session (siehe Kapitel 3.3.5). Der Elan Sim Server versendet nun die aktuelle Bestandesbeschreibung an den durch die Userin im Dialog ausgewählten Wachstumsgenerator. Hierzu wird die Bestandesbeschreibung via HTTP-POST als HTML 4.01 multipart Formdata [Nebel & Masinter 1995] an die im GGinfo (siehe Kapitel 3.2.6) angegebene Url [Berners-Lee et al. 2005] gesendet.

Hat der Filetransfer geklappt, so erhält der ESS als Antwort eine Statusmeldung und die Url zum Abholen der Prognoserechnung in zwei Textzeilen zurück:

```
status=ok
url=http://www.url_der_growthengine.de/outscript.cfm
```

Bei fehlerhafter Übermittlung der Bestandesbeschreibung wird nur die Statusmeldung zurückgegeben:

```
status=wrong
```

Schritt 2. In regelmäßigen Abständen ruft der ESS die erhaltene URL zur Abfrage der Prognoseberechnung via HTTP ab. Die Antwort wird als plain/text übermittelt. 3 Antworten sind möglich:

1. `status=wait`
   wenn die Berechnung noch nicht fertiggestellt ist. Der ESS wiederholt die Anfrage nach dem eingestellten Zeitintervall.
2. `status=wrong`
   wenn die Berechnung fehlgeschlagen ist. Der ESS stellt die Anfrage ein.
3. `status=ready`
   `#VRML V2.0 utf8 …… die Bestandesbeschreibung ……`
   wenn die Prognoseberechnung abgeschlossen ist. Zeile 2ff enthält die neue Bestandesbeschreibung mit der Wachstumsprognose für den gesendeten Bestand.

Die Prognoseberechnung wird in die Szenenliste in einen Unterbaum der Ursprungsszene eingegliedert und kann anschließend durch die Clientprogramme ausgewählt werden.

3.4.6 Integration neuer Wachstumsgeneratoren

Durch die offen gestaltete Schnittstelle des ESS zur Anbindung der Wachstumsgeneratoren und der Verwendung von HTTP als Transferprotokoll ist zur Anbindung weiterer

Wachstumsgeneratoren keine Anpassung des Programmcodes des ESS und des Virtual Forester Clients erforderlich. Für die Einbindung eines neuen Programms für die Wachstumsprognoseberechnung ist erforderlich:

Im Wachstumsgenerator muss eine Schnittstelle integriert werden, die den Empfang von HTML-Formulardaten (multipart/formdata) via HTTP-Post und zum Versenden der im Kapitel 3.4.5 definierten Antworten als plain/text-Seiten erlaubt.

Eine Bestandesbeschreibung muss erstellt werden, welche die URL der oben genannten Schnittstelle des Wachstumsgenerators als GGinfo-Element (Kapitel 3.2.6) enthält und alle für das verwendete Wachstumsmodell erforderlichen Bestandesdaten als WorldInfo-Element (Kapitel 3.2.3) und Baumdaten als TREE-Element (Kapitel 3.2.4.1) bereitstellt.

Die Bestandesbeschreibung muss in die Serverkonfigurationsdatei (Kapitel 3.4.2) aufgenommen werden und im "Scenes"-Ordner auf dem Server vorliegen. Weiterhin müssen alle verwendeten Texturen in den "Textures"-Ordner kopiert werden.

3.5 Quality Identifier

Der Quality Identifier (QID) ist ein System zur vereinfachten Darstellung von forstwirtschaftlich relevanten Qualitätskriterien von Bäumen im Forester. Als forstwirtschaftlich relevant werden in diesem Zusammenhang Baumeigenschaften angesehen, welche die Bestandesbehandlung – insbesondere die Ansprache des Baumes bei der Durchforstung – beeinflussen. Neben den Lage- und Größenparametern sind dies Anzeiger der Vitalität – wie Nadelvergilbung und Nadelverlust – sowie Anzeiger zur potenziellen Nutzung des Stammholzes – wie Zwieselbildung und Astigkeit.

Ziel des von Ortlepp [2006] im Rahmen seiner vom Autor betreuten Masterarbeit konzipierten Quality Identifier ist es, anhand weniger angegebener Schlüsselbegriffe zur Qualität der Bäume die Darstellung der Bäume im Forester so zu verändern, dass ihre forstwirtschaftliche Qualität durch die Anwenderin angesprochen werden kann. Die folgende Zusammenfassung basiert auf der Arbeit von Ortlepp [2006].

Basis des QID ist eine umfangreiche Sammlung von Kronen- und Stammtexturen.

3.5.1 Kronentexturen

Für die Krone benutzt Ortlepp [2006] für jede Baumart 21 Texturen, die eine Kombination aus Kronenstruktur und Kronenschadklasse darstellen. Für die Kronenstruktur sind 5 Klassen eingeteilt mit folgenden Ausprägungen:

- normal
- Solitärbaum
- Krone gebrochen
- Zwiesel
- tot

Für die Schadklasse benutzt Ortlepp ein System analog zu der, von Meining et al. [2005] bei der Waldzustandserhebung in Baden-Württemberg verwendeten Kombination von Nadel-/Blattverluststufe und Vergilbungsstufe.

Kombinationsschadstufen				
Nadel-/Blatt-verluststufe	**Vergilbungsstufe**			
	0	**1**	**2**	**3**
0	0	0	1	2
1	1	1	2	2
2	2	2	3	3
3	3	3	3	3
4	4			
Schadstufe 0: ungeschädigt				
Schadstufe 1: schwach geschädigt			**Warnstufe**	
Schadstufe 2: mittelstark geschädigt			**deutlich geschädigt**	
Schadstufe 3: stark geschädigt				
Schadstufe 4: abgestorben				

Tabelle 4: Kombinations-Schadstufen nach Meining et al. [2005]

Zur Einordnung der Nadel-/Blattverluststufen und Verfärbungsstufen wird auf Hanisch [1990] verwiesen.

Stufen	NBV-Prozent	Vergilbungsprozent
0	0%-10%	0-10%
1	11%-25%	11-25%
2	26%-60%	26-60%
3	61%-99%	>60%
4	100%	

Tabelle 5: Nadelblattverluststufen (NBV) und Verfärbungsstufen nach Hanisch [1990]

Die Verweise auf die genannten Werte zur Bildung der Schadklassen dienen der Anleitung zur Auswahl und Interpretation der verwendeten Textur. Letztendlich liegt die Verwendung der Schadklassen aber in der Verantwortung des Autors der Bestandesbeschreibung.

Aus der Kombinationen der beiden Merkmale Kronenstruktur und Kronenschadklasse ergeben sich für jede Baumart die 21 verschiedenen Texturen zur Darstellung der Baumkrone (für die Darstellung eines Totbaumes wird nur eine Textur verwendet).

3.5.2 Stammtextur

Für den Stamm verwendet Ortlepp eine Skala von Qualitätskriterien abnehmender Güte, die er von den Heckschen Schaftformklassen von 1897 [Heck 1931 S. 332] ableitet. Folgende Ausprägungen sind erlaubt [vergleiche Ortlepp 2006, S. 18f]:

- schöner Nutzstamm
- wenig astig
- sehr astig
- schwacher Rindenschaden
- starker Rindenschaden
- Totholz

3.5.3 Angabe der Qualitätskriterien

Für die Angabe der Qualitätskriterien für Baumkrone und Stamm und die Verwendung des Quality Identifiers wurde im Virtual Forester-Projekt ein Namensraum für die `labels/data`-Felder des TREE-Elementes (siehe Kapitel 3.2.4.1) reserviert. Alle in diesem Namensraum gültigen Schlüsselbegriffe beginnen mit "QC_".

Die wichtigsten Schlüsselbegriffe für den QID werden von Ortlepp wie folgt angegeben (vergleiche Ortlepp 2006. S. 28ff):

`"Qc_Tree_Species"` zur Angabe der Baumart. Zurzeit mit vollständigen Texturensätzen hinterlegte Baumarten sind `"Kiefer"`, `"Buche"` und `"Fichte"`. Der Standardwert ist `"Fichte"`.

`"Qc_Crown_structure"` zur Angabe der Kronenstruktur.

Mögliche Ausprägungen sind: `"Solitärbaum"`, `"Krone_gebrochen"`, `"Zwiesel"`, `"normal"`, `"tot"`. Der Standardwert ist `"normal"`.

`"Qc_Crown_damage_class"` zur Angabe der Kronenschadklasse.
Mögliche Ausprägungen sind: `"0"`, `"1"`, `"2"`, `"3"`, `"4"`. Der Standardwert ist `"0"`.

`"Qc_Stem_damage_class"` zur Angabe der Stammschadklasse.
Mögliche Ausprägungen sind: `"schoener_Nutzstamm"`, `"wenig_astig"`, `"sehr_astig"`, `"schwacher_Rindenschaden"`,`"starker_Rindenschaden"`, `"Totholz"`. Der Standardwert ist "schoener_Nutzstamm".

`"QC_Overwrite"` zur Genehmigung bzw. zum Verbot des Überschreibens von vorhandenen Datenfeldern. Mögliche Werte sind `"yes"` und `"no"`. Standardwert ist `"no"`.

3.5.4 Anwendung des Quality Identifiers

Das Quality Identifier-System besteht aus einen Java-Programm zur Bearbeitung der Bestandesbeschreibung, zwei Konfigurationsdateien mit der Zuordnung der Qualitätsschlüsselbegriffe zu der jeweiligen Kronen- bzw. Stammtextur und dem Satz von 21 Kronentexturen und 6 Stammtexturen pro Baumart als Grafikdateien.

Das in Java geschriebene Programm kann durch Kommandozeilenaufruf mit einem Eingabefile und einem Ausgabefile als Aufrufparameter gestartet werden. Die als Eingabefile genannte Bestandesbeschreibung wird gelesen, für jedes TREE-Element werden die in den `labels/data`-Feldern genannten Schlüsselbegriffe für Qualitätskriterien der Bäume ausgewertet und die entsprechenden Kronen- und Stammtexturen mit

den crown_texture- und stem_texture-Feldern gesetzt. Anschließend wird die Szene mit dem als Ausgabefile angegebenen Namen gespeichert.

Um Kompatibilitätsprobleme zu vermeiden (vergleiche Kapitel 3.2.4.1: Referenz auf artenspezifische Definitionen) wird ein vorhandenes Species-Feld entfernt. Der Wert wird in die labels/data-Felderkombination unter dem Label OLD_Species als Sicherheitskopie eingetragen.

Weitere in der Ausgangsbeschreibung vorhandene Felder in einem TREE-Element (z. B crown_texture und stem_texture) werden nur dann überschieben, wenn QC_Overwrite den Wert "yes" gesetzt hat (vergleiche Ortlepp 2006, S. 30). Wird ein Datenfeld überschieben, so wird es in der labels/data-Feldkombination mit OLD_Feldname gesichert.

Der Quality Identifier kann zum gegenwärtigen Zeitpunkt nur manuell ausgeführt werden und ist noch nicht in die Elan Sim Serverkomponente integriert. Die Nutzung im Einzelplatzbetrieb ist daher mit wenigen Arbeitsschritten einfach möglich, die Integration in den ESS für den Client-Server Betrieb ist konzeptionell gelöst, muss aber noch umgesetzt werden.

3.6 Der Wachstumsgenerator GroIMP

In Kapitel 2 wurden verschiedene Ansätze und Programme zur Wachstumssimulation von Waldbeständen kurz vorgestellt. Aufgabe des Wachstumssimulators ist in allen Fällen, die Walddynamik der virtuellen Bestände zu simulieren und eine Prognose der Bestandesentwicklung für die Zukunft zu erstellen. Dabei sollen sich die Studierenden die zugrundeliegenden Modellannahmen aus dem Zusammenhang zwischen den Durchforstungsmaßnahmen und der Wachstumsprognose erschließen können.

Für diese Arbeit wurde deshalb mit GroIMP der weitestgehende Ansatz als Wachstumsgenerator ausgewählt: Weitestgehend in dem Sinne, dass GroIMP nicht auf ein spezifisches Wachstumsmodell festgelegt ist, sondern sich für eine Vielzahl von regelbasierten Modellen eignet. Das für diese Arbeit verwendete Modell wird in Kapitel 3.7 genauer erläutert.

Bevor in den folgenden Kapiteln die Grundlagen der Relationalen Wachstumsgrammatiken, der Modellsprache XL und der Modellschale GroIMP genauer ausgeführt werden, scheint eine Beleuchtung der allgemeinen Gründe für die Verwendung regelbasierter Programmierung in der Lehre sinnvoll.

Wie in Kapitel 2.2.6 zu GROGRA ausgeführt, sind das Modell und die Modellschale bei der regelbasierten Pflanzenmodellierung voneinander getrennt. Lindenmayer-Systeme und Extended Lindemayer-Systeme sind Beispiele für höhere Modellsprachen zur Modellierung von Pflanzen. Sie sind für die Strukturmodellierung von Pflanzen ein besonders geeigneter Ansatz: "their underlying rule-based paradigm matches the observed plant growth at a macroscopic level" [Kniemeyer et al. 2006].

Abbildung 15 zeigt darüber hinaus die unterschiedlichen Architekturen bei klassischer Programmierung von Simulationsmodellen und bei regelbasierter Programmierung in einer generischen Modellschale. Im Gegensatz zur reinen prozeduralen oder objektorientierten Programmierung in einer der klassischen Programmiersprachen, liegt das Modell beim regelbasierten Ansatz offen und bedarf keiner kompletten Neukompilierung der Software. Die Modelle werden daher offen für informierte Userinnen, die keine Informatikerinnen sind.

Im Rahmen des Lernprozesses zur Wachstumsmodellierung können die Studierenden deshalb direkt in das Modell eingreifen und die Konsequenzen der Änderungen beobachten und analysieren.

In Kapitel 2.2.6 wurden bereits die Einschränkungen angesprochen, die sich aus den bisherigen Ansätzen zu Lindenmayer-Systemen ergeben. So können für GROGRA zwar jederzeit weitere Erweiterungsfunktionen – z. B. für die Berechnung der beschattenden Nadelmasse – geschrieben werden, der oben genannte Vorteil der Trennung von Modellschale und Modell geht dabei aber verloren.

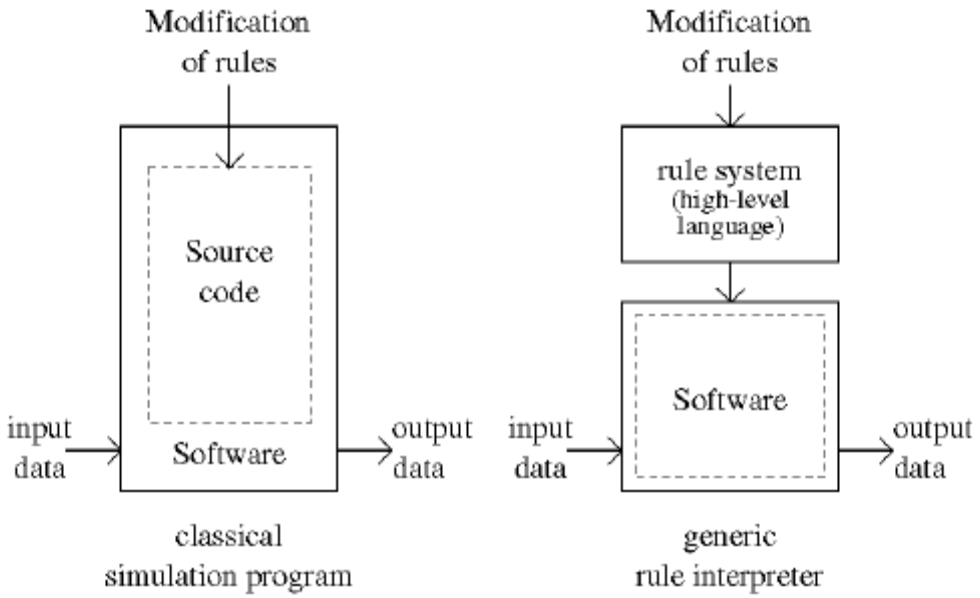


Abbildung 15: Gegenüberstellung der Architektur von klassischer Modellprogrammierung (links), die für jede Modelländerung ein Umschreiben des Software-Quellcodes verlangt, und regelbasierter Modellprogrammierung in einer generischen Software-Umgebung. [Kurth & Sloboda 2001 S. 2].

Durch die Erweiterung des regelbasierten Ansatzes von Strings auf Graphen haben Kurth und Kniemeyer einen Weg eröffnet, der diese Probleme in der Pflanzenmodellierung überwindet. In den folgenden Kapiteln wird eine kurze Zusammenfassung des grundlegenden Konzepts der Relationalen Wachstumsgrammatiken (RGG), der Modellsprache XL und der Modellschale GroIMP gegeben. Für eine ausführlichere Darlegung wird auf folgende Publikationen der Entwickler des Ansatzes verwiesen: [Kniemeyer 2004, Kurth et al. 2005, Kniemeyer et al. 2006, Kurth et al. 2006, LGS: XL Language Specification]. Die Ausführungen in den nächsten Kapiteln folgen im Wesentlichen den genannten Publikationen.

3.6.1 Relationale Wachstumsgrammatiken (RGG)

Relational Growth Grammars bzw. Relationale Wachstumsgrammatiken (RGG) gehören zu den Graph-Grammatiken und sind das Konzept einer Formalisierung der Kombination von L-Systemen – erweitert von Zeichenketten auf netzwerkartige Datenstrukturen (Graphen im Sinne der mathematischen Theorie) – mit einer allgemeinen höheren Programmiersprache.

Relationale Wachstumsgrammatiken wurden entwickelt, um das Konzept der regelbasierten Pflanzenmodellierung durch L-Systeme mit Techniken der prozessorientierten Modellierung zu kombinieren.

Mit Hilfe von Graphen kann die 3-dimensionale Struktur von Pflanzen wesentlich direkter abgebildet werden, als mit einer Kette von Symbolen, die erst einer Übersetzung – z. B. durch Turtle-Geometrien – in eine räumliche Struktur bedürfen.

Ein Graph besteht aus Knoten und gerichteten Kanten (Relationen), die die Knoten miteinander verbinden. Es gibt verschiedene Arten von Kanten. Betrachten wir die Knoten als eine verallgemeinerte Form der Symbole in L-Systemen und die Kanten als verallgemeinerte Reihung der Symbole – Kantentyp "ist Nachfolger von" –, haben wir eine Entsprechung zum String eines L-Systems. Durch Einführung weiterer Kantentypen können neben solchen Sequenzen auf einfache Weise auch Baumstrukturen – zusätzlicher Kantentyp "ist Tochter von" – und Netzwerkstrukturen – weiterer Kantentyp "ist Vorgänger von" – beschrieben werden.

Die Kantentypen sind frei definierbar. "Auch komplexe Beziehungen, wie die zwischen Genotyp und Phänotyp, können somit prinzipiell mit derselben formalen Einfachheit beschrieben werden wie die topologische Nachbarschaft pflanzlicher Module in klassischen L-Systemen" [Kurth et al. 2006].

Graphen können – ähnlich wie L-Systeme – durch Regeln verändert werden. Eine relationale Wachstumsgrammatik ist ein Satz von RGG-Regeln, die auf einen Graphen angewendet werden. Nach dem bei L-Systemen bewährten Prinzip der Ersetzungsregeln haben auch diese die Form

linke Regelseite → rechte Regelseite,

wobei es bei der Gestaltung der Regelseiten und der Form der Ersetzung wesentlich mehr Möglichkeiten gibt. In einer detaillierteren Form kann eine Graphersetzungsregel wie folgt beschrieben werden:

(**K**ontext *), **L**inker Graph, (**B**edingungen) → **R**echter Graph { **P**rogramm}*

"**L**inker Graph" steht für eine Menge von Graphen, bestehend aus Knoten und Kanten, welche bei Ausführung der Regel durch eine andere Menge von Graphen – definiert durch "**R**echter Graph" – ersetzt wird.

"**K**ontext" steht für eine weitere bestimmte Kombination von Knoten und Kanten, von deren Vorhandensein im definierten Kontext zum "**L**inken Graph" die Ausführung der Regel abhängig gemacht wird. Im Gegensatz zu **L** wird der **K**ontext aber nicht durch die rechte Regelseite ersetzt.

"**B**edingungen" steht für einen Satz von logischen Ausdrücken, die als Bedingung für die Regelausführung definiert werden können. In diesen Ausdrücken können sowohl Eigenschaften von Knoten in **L** oder **K** verwendet als auch Funktionen aufgerufen werden.

"**P**rogramm" steht für optionalen prozeduralen Code, der nach Einsetzen des "**R**echten Graphen" ausgeführt wird [vergl. Kurth et al. 2005].

Auch bei der Ausführung der Ersetzungsregeln – abstrakt ausgedrückt durch → – gibt es Änderungen. Beibehalten wurde die Anwendung der Ersetzungsregel im L-System-Stil – bei dem die Knoten, auf welche die linke Regelseite passt, am selben Platz im Graphen durch die Knoten der rechten Regelseite ersetzt werden.

Hinzugekommen sind die echten Graphersetzungsregeln, bei denen der Ort des Einfügens der Knoten durch die rechte Regelseite explizit definiert werden muss. Das bedeutet, dass Knoten und all ihre Kanten zu anderen Knoten auf der rechten Regelseite angegeben werden müssen.

Die Semantik der Relationalen Wachstumsgrammatik-Regeln wurde so definiert, dass sie die GROGRA L-System-Regeln als einen Spezialfall abdeckt, und die Syntax wurde weitgehend ähnlich gestaltet. Auch werden die RGG-Regeln ebenfalls parallel auf die gesamte Struktur angewandt, um die Nähe zu den Wachstumsprozessen von Pflanzen zu behalten.

"Somit basieren relationale Wachstumsgrammatiken auf dem etablierten regelorientierten Modellierungs-Formalismus der L-Systeme, erweitern aber die zugrundeliegenden Datenstrukturen von Zeichenketten auf Graphen, die Zeichenkettenersetzung auf Graph-Ersetzung und stellen eine Verbindung zum prozeduralen und zum objektorientierten Programmierparadigma her, indem eine konventionelle objektorientierte Programmiersprache (Java) mit eingebettet wird" [Kurth et al. 2006].

3.6.2 Die Modellierungssprache XL

XL ist die konkrete Implementierung des Konzeptes der Relationalen Wachstumsgrammatiken und somit die erste Modellierungssprache, die dieses Konzept umsetzt. Sie ist definiert als eine Erweiterung der objektorientierten Programmiersprache Java und macht somit den vollen Umfang von Java für die Integration in die regelbasierte Modellierung verfügbar.

Bei der Umsetzung von XL konnte die bekannte Syntax der GROGRA-L-Systeme weitgehend beibehalten werden, wenngleich die Ersetzungsregeln in eine Methode eingebettet werden müssen, um ein gültiges XL-Programm zu bilden. Dies scheint auf dem ersten Blick etwas umständlich, zeigt bei umfangreicheren Modellen aber schnell den Vorteil, dass diese besser strukturiert werden können.

Die Stelle der Symbole in L-Systemen nehmen in XL Objekte, d.h. Instanzen von Java-Klassen, ein. Der jeweilige Graph hat solche Objekte als Knoten.

Am deutlichsten wird der Unterschied zwischen der Anwendung von Symbolen und Objekten wohl an der Behandlung von Syntaxelementen wie Klammerzeichen "[" und "]" in den Regeln. Konventionelle L-Systeme behandeln die Klammerzeichen als einfaches Symbol und nehmen sie in den String mit auf. Eine Verzweigung der Struktur wird erst bei der Interpretation des Strings z.B. durch Turtle-Geometrie angezeigt. XL hingegen betrachtet die Klammerzeichen als Syntaxelemente, die einen echten Tochtergraphen – d. h. eine Verzweigung in der Datenstruktur – anzeigen. Sie drücken sich im Graph daher als Relation zwischen Knoten aus und nicht als Knoten selbst.

Die Objektklassen können in XL frei definiert werden. Auch das Überladen von bereits bestehenden Klassen ist möglich. Wenngleich XL als Graph-Ersetzungssprache nicht auf die Anwendung im Bereich der Pflanzenmodellierung begrenzt ist, hat die Verwendung von XL in Verbindung mit GroIMP (vergl. 3.6.3) hier einen Schwerpunkt. So stehen in GroIMP Klassenbeschreibungen für die aus L-Systemen bekannten Symbole wie Turtle-Geometrie für einfache Raumelemente sowie umfangreiche weitere Klassen unter anderem für komplexe räumliche Objekte wie Box, Sphere, Cone, Cylinder, Frustum oder Nurbs zur Verfügung.

Um die extern definierten Objekte nutzen zu können, ist es notwendig, die entsprechenden Java-Klassen zu importieren. XL bietet hier eine spezielle Variante an, welche die gebräuchlichsten Klassen automatisch importiert. Dieser "RGG Dialekt von XL" wird durch die Endung .rgg der Quellcodedatei erkannt. Im Folgenden wird nur auf diesen Dialekt Bezug genommen. Für die weiteren Vereinfachungen, die sich durch die Verwendung des Dialekts ergeben, wird auf das in GroIMP eingebaute Tutorium (Kapitel 3.3 im Tutorium) verwiesen.

Für die Definition eigener Objektklassen sind verschiedene Wege möglich:

Die `module`-Definition bietet eine Kurzschreibweise, bei der nur der Name des Symbols und die Parameter angegeben werden müssen. Alles weitere übernimmt XL. Hierfür sind keine tieferen Kenntnisse von Java nötig.

Die explizite Programmierung eigener, beliebig komplexer Java-Klassen stellt eine zweite Möglichkeit dar. Dieser Weg richtet sich an erfahrenere Userinnen mit Kenntnissen in der objektorientierten Programmierung. Es müssen alle Methoden und Klassen zur Anbindung in XL vollständig selbst implementiert werden.

Eine Mischform bietet sich insbesondere bei der Verwendung von eigenen Objekten mit Parametern auf der linken Regelseite an. Über die `module`-Definition können Objekte mit Parametern definiert werden, die eine eigene – in Java geschriebene – Objektklasse erweitern. Die aufgeführten Parameter werden durch XL für die linke Regelseite verfügbar gemacht. Über super.Attributname kann auf Eigenschaften der erweiterten Klasse zugegriffen werden. Das folgende Beispiel (Bsp. 3.6-1):

Bsp. 3.6-1: Module Definition in XL

```
module Segment(int bnr, int snr, super.length, float base) extends
Segmentklasse(length);
```

definiert das Modul Segment als Erweiterung der Segmentklasse. Die Parameter "bnr", "snr", "length" und "base" können so für Bedingungen auf der linken Regelseite verwendet werden. Der Parameter "length" wird an den Constructor der Segmentklasse weitergegeben. Alle Eigenschaften, sowohl die 4 genannten der Klasse "Segment", wie auch die geerbten öffentlichen Eigenschaften aus der Klasse "Segmentklasse", sind über den Attributeditor des Objektes erreichbar.

Eine umfangreiche Beschreibung von XL kann an dieser Stelle nicht erfolgen. Wie bereits erwähnt, wird auf die in Kapitel 3.6 genannten Veröffentlichungen und die zahlreichen Anwendungsbeispiele verwiesen. Einige für das später beschriebene Wachstumsmodell wichtige Eigenschaften von XL sollen an dieser Stelle aber kurz charakterisiert werden. Hierzu werden Beispiele aus dem Modell verwendet. Um die Beispiele anschaulich zu gestalten, wird im Folgenden die XL/Java-Implementation in GroIMP verwendet.

Regeltypen und die Benennung von Nodes:

Ein sehr wichtiges Element in der Sprache XL/Java ist die Benennung von Knoten innerhalb einer Regel. Gezeigt wird dies an einem Beispiel (Bsp. 3.6-2) mit einer Ersetzungsregel im **L-System-Stil**, welche durch das Syntaxelement ==> angegeben wird:

Bsp. 3.6-2: Regel im L-System-Stil mit benannten Knoten

```
//Modul-Definition:
module Bestand(Point3d referenzpunkt, float xseite, float yseite,
       boolean aktiv) extends Null(){setTransform(referenzpunkt);}
module Referenzelement;
module Pfosten() extends Null;
//Ersetzungsregel:
best:Bestand(referenzpunkt, xseite, yseite, aktiv) ==> best
 [ Referenz
      [ p1:Pfosten ]    { p1.setTransform(0,0,0);}
      [ p2:Pfosten ]    { p2.setTransform(0,yseite,0);}
      [ p3:Pfosten ]    { p3.setTransform(xseite,0,0);}
      [ p4:Pfosten ]    { p4.setTransform(xseite,yseite,0);}

… Einfügen von Bäumen …
 ]
```

Das Beispiel enthält drei Objektklassen. Die Ersetzungsregel ist anwendbar auf Objekte der Klasse Bestand. Durch best:Bestand werden die Objekte der Trefferliste der linken Regelseite benannt und können auf der rechten Regelseite direkt angesprochen werden. In der Ausführung heißt das: jede Instanz der Klasse Bestand wird im Graphen durch sich selber plus weitere Objekte ersetzt. Würde die Regel hingegen folgendermaßen (Bsp. 3.6-3) anfangen:

Bsp. 3.6-3: Regel im L-System-Stil ohne benannten Knoten

```
…
best:Bestand(referenzpunkt, xseite, yseite, aktiv) ==>
       Bestand(referenzpunkt, xseite, yseite, aktiv)
…
```

so würde jede Instanz der Klasse Bestand durch eine neue Instanz derselben Klasse ersetzt. Als weiterer Knoten wird dem Graphen in Bsp. 3.4-1 mit einer Tochterrelation ein Objekt der Klasse "Referenz" angefügt, welches zwei Tochterobjekte der Klasse "Pfosten" bekommt. Auch die Pfosten werden in dieser Regel benannt, diesmal auf der rechten Seite, damit sie anschließend durch den in {} eingefassten Java-Code angesprochen werden können. Angesprochen wird die Standardmethode "`setTransform`()" mit Werten, die aus den Parametern des Bestandes-Objektes aus der linken Regelseite bezogen werden. Mit dieser Methode werden die Raumkoordinaten der Pfostenobjekte relativ zur Referenz verändert. Abbildung 16 zeigt den visuellen Output und die entsprechende Graphen-Struktur.

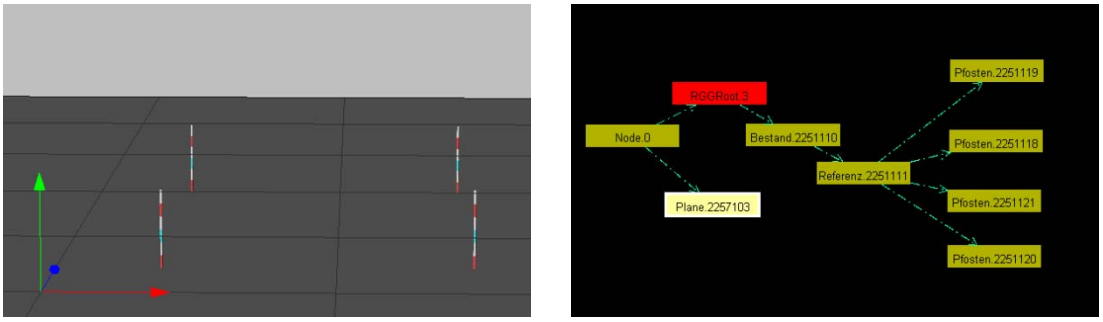


Abbildung 16: Szene des Modellbeispiels (Bsp. 3.6-2) in der GroIMP 3D-Ansicht (links) und der Graphen-Ansicht (rechts). Ausgehend vom Szene-Basisknoten (Node 0) wird das Ebene-Objekt (Instanz der Klasse Plane, erzeugt durch die Userin) und der Wurzelknoten des Modells (Instanz der Klasse RGGRoot, erzeugt durch das XL Plug-In) gezeigt. Unterhalb von RGGRoot ist die Graphen-Struktur des Modells sichtbar. Alle gezeigten Relationen sind Tochterrelationen.

Da die Klasse "Referenz" keine bekannte Objektklasse aus GroIMP erweitert, wird das Objekt in der GroIMP 3D-Ansicht auch nicht gezeichnet. Es wird vielmehr im später präsentierten Modell verwendet, um alle Objekte eines Bestandes in einem Subgraphen zusammenzufassen. Das folgende Beispiel (Bsp. 3.6-4) verwendet eine Regelanwendung im Stil der klassischen **Graphersetzungsregeln,** um das Objekt Referenz mit allen Relationen zu löschen. Hierdurch wird dann auch der gesamte Sub-Graph (alle Bäume und Pfosten) entfernt.

Bsp. 3.6-4: Regel im Graphersetzungsstil zum Löschen von Knoten und Relationen

```
[
      Referenz ==>> ;

];
```

Der dritte in XL definierte Regeltyp – die **Aktualisierungsregel** – wird am Bsp. 3.6-5 zur Berechnung des Nettoassimilatespeichers und der Produktionseffizienz der Bäume gezeigt.

Bsp. 3.6-5: Aktualisierungsregel

```
// Moduldefinition
…
module Baum(int bnr, super.baumAlter, super.baumHoehe, super.d13,
super.baumArt, super.kronenAnsatz) extends Baumklasse(baumAlter,
baumHoehe, d13, baumArt, kronenAnsatz);
…
// Aktualisierungsregel
[
      b:Baum ::>
      {
            b[nettoAssimilateSpeicher] =
                  b.bruttoAssimilateSpeicher - b.gesamtRespiration;
            b.kalkuliereProduktionsEffizienz();
            b.kalkuliereRelativeProduktionsEffizienz();
      }
]
…
```

Die Klasse "Baum" erweitert die komplexere Java-Klasse "Baumklasse". Diese enthält unter anderem die Eigenschaften "nettoAssimilateSpeicher", "bruttoAssimilateSpeicher" und "gesamtRespiration" und die Methoden "kalkuliereProduktionsEffizienz()" und "kalkuliereRelativeProduktionsEffizienz()". Die Aktualisierung – dargestellt durch das Syntaxelement ::> – erlaubt es, die Eigenschaften der benannten Objekte zu aktualisieren und Objektmethoden aufzurufen.

Queries

Das wohl mit Abstand mächtigste Werkzeug in der XL-Sprache sind Queries. Queries sind Abfragen über die Graphen-Datenstruktur und ergeben als Rückgabewert eine Liste von Knoten, welche die Abfragebedingungen erfüllen. Queries können sowohl im normalen XL/Java-Code wie auch in RGG-Regeln verwendet werden und eröffnen damit einen direkten und effizienten Weg des Zugriffs auf die Objekte des Graphen. Sie werden mit folgender Notation erstellt:

(Kombination von Knoten und Kanten, (optionaleBedingung) *)*

Das folgende Beispiel (Bsp. 3.6-6) zeigt die Verwendung einer Query (rot markiert) in einer XL/Java-for-Schleife. Mit der Query werden alle Objekte (Knoten) der Klasse "Baum" aus der Datenstruktur herausgesucht und für jedes Objekt wird die Eigenschaft "nadelMasseG" mit der objekteigenen Methode "addiereNadelJahrgangMassen()" aktualisiert.

Bsp. 3.6-6: Einfache Query-Definition

```
for ((* b:Baum *)) {
      b[nadelMasseG] = b.addiereNadelJahrgangMassen();
}
```

Ein weiteres Beispiel (Bsp. 3.6-7) zeigt eine deutlich komplexere Anwendung einer Query, da hier sowohl eine Relation explizit angegeben wird als auch mehrere Bedingungen in die Query eingeführt werden:

Bsp. 3.6-7: Query-Definition mit Bedingungen

```
"gegeben sei ein Baumsegment s:Segment für welches alle beschattenden
Segmente anderer Bäume ermittelt werden sollen:"
…
for ((* b:Baum (-->)* a:Segment ,
   (s.getParent() != b && b.getLastChild() in cone(s, false, winkel)),
   ( a.nadelDichteGesamt > 0.0f && a in cone(s, false, winkel))*))
…
```

Im Einzelnen wird die Query wie folgt gelesen:

Die Übereinstimmung für den Graphen (rot markiert) sucht alle Tupel eines Objektes (Knoten) der Klasse "Baum" mit einem in Kantenrichtung liegenden Objekt der Klasse "Segment". Die Kantenrichtung wird durch das Element `(-->)*` definiert, die Art der Relation ist aber beliebig, d. h. es kann z. B. eine Nachfolgerkante oder eine Tochterkante sein.

In die Liste mit aufgenommen werden aber nur Objektpaare, welche die 4 Bedingungen erfüllen.

Bedingung 1 (`s.getParent() != b`) stellt sicher, dass der Baum, zu dem s gehört, ausgeschlossen wird. Die Methode "`s.getParent()`" gibt bei entsprechender Datenstruktur des Modells (siehe Kapitel 3.7.3) ein Baumobjekt zurück.

Bedingung 2 (`b.getLastChild() in cone(s, false, winkel)`) überprüft, ob das Gipfelsegment des jeweiligen Baumes b in einen Lichtkonus von s liegt. Diese beiden Bedingungen sind mit einer Klammer gekapselt und werden zuerst überprüft, um alle für die Beschattung unbedeutenden Bäume auszuschließen. Erst auf die verbleibende Anzahl von Baumobjekten wird nun das zweite Paar von Bedingungen angewandt.

Bedingung 3 (`a.nadelDichteGesamt > 0.0f`) schließt alle Segmente aus, die keine Nadelmasse besitzen und

Bedingung 4 (`a in cone(s, false, winkel)`) lässt nur Elemente zu, die im Lichtkonus von s liegen.

Die gezeigte Möglichkeit mehrerer Bedingungsblöcke erlaubt dem XL-Compiler, die Suchoperationen auf den Graphen zu optimieren, und dies schlägt sich erheblich auf die Rechenzeit nieder.

3.6.3 Das Programm GroIMP

Hauptziel des Programms GoIMP (Growth-Grammar related Interactive Modelling Platform) ist die Umsetzung von in XL geschriebenen, auf relationalen Wachstumsgramatiken (RGG) beruhenden Modellen [vergl. Kurth et al. 2005]. Eingedenk des primären Anwendungsfeldes im Forschungsbereich der Funktionalen-/Strukturalen Pflanzen-Modellierung [siehe FSPM 2004] bietet GroIMP zahlreiche Features, die es zu einem vielseitigen und komfortablen Graphischen Userinterface für die Entwicklung und Ausführung von Pflanzenmodellen macht.

Grundlage der Modellierung mit GroIMP ist die Sprache XL. Sie ist vollständig in das Programm integriert. Neben dem XL-Kompiler bietet GroIMP eine Fehlerausgabe mit direkten Sprungmarken in den Modellcode, jEdit [Pestov et al.] als integrierten Code-Editor sowie eine XL-Konsole (Abbildung 18 u. r.) zur Direkteingabe von XL Befehlen.

Für die Modellierung bietet GroIMP eine vollständige Klassenbibliothek für räumliche Objekte an. Dies umschließt Basiselemente wie Kugel, Zylinder, Kegel, Rechteck, die Turtle-Geometrien – wie sie von den L-Systemen bekannt sind – sowie über Spline-Funktionen definierte Oberflächen (Nurbs) [Piegl & Tiller 1997, zitiert in Kniemeyer 2004].

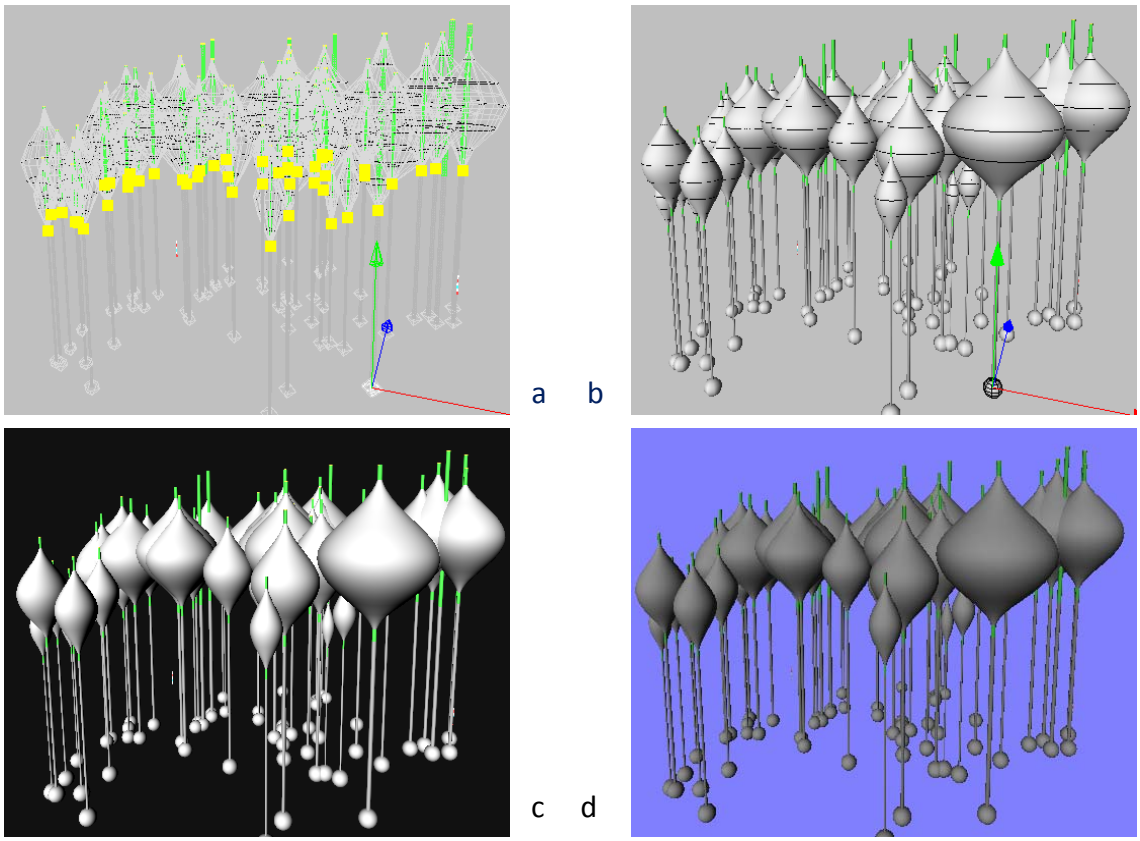


Abbildung 17: Die 4 Optionen zur Visualisierung von GroIMP im Vergleich: wireframe (a) und OpenGL® (b) als realtime Darstellung und die Rendering-Programme Twilight (c) – in GroIMP integriert – und Pov-Ray® (d) als externes Programm.

Wie Abbildung 17 zeigt, ist die Visualisierung außer in der Gitternetzansicht auch in einer realtime OpenGL® [SGI]-Darstellung möglich. Für ein anspruchsvolleres Rendering können der eingebaute Raytracer Twilight [Tauer 2006, Kniemeyer 2007] und eine Schnittstelle zu Pov-Ray® [Persistence of Vision Raytracer] verwendet werden.

Für die Gestaltung und Bearbeitung von Objektoberflächen sowie die Einbindung von externen Grafiken als Textur ist mit dem Material-Editor ein komfortabler Dialog im üblichen Stil bewährter 3-D-Modelliersoftware vorhanden. Materialmuster können im Projekt abgespeichert werden und stehen zur Nutzung im Code zur Verfügung.

Alle sichtbaren Knoten des Graphen können direkt über das GUI (Abbildung 18 o. l.) angesprochen und manipuliert werden. Dies beinhaltet das Verschieben, Skalieren, Entfernen vorhandener und das Hinzufügen neuer Objekte. Der Attribute-Editor (Abbildung 18 o. r.) gibt zusätzlich direkten Zugriff auf alle definierten Eigenschaften des ausgewählten Objektes.

Abhängig von den im Modell enthaltenen öffentlichen Methoden erstellt GroIMP selbstständig ein Menü zur schrittweisen oder automatischen Ausführung der Simulation. Beim Beenden der Simulation kann der jeweils aktuelle Graph gespeichert werden. Der aktuelle Stand einer Simulation wird so festgehalten und kann später als Grundlage einer weiteren Simulation genutzt werden. Auch die Oberfläche der Entwicklungsumgebung – mit den jeweils benötigten Dialogen – kann projektspezifisch angepasst und gespeichert werden. Abbildung 18 zeigt eine typische Simulationsoberfläche von GroIMP.

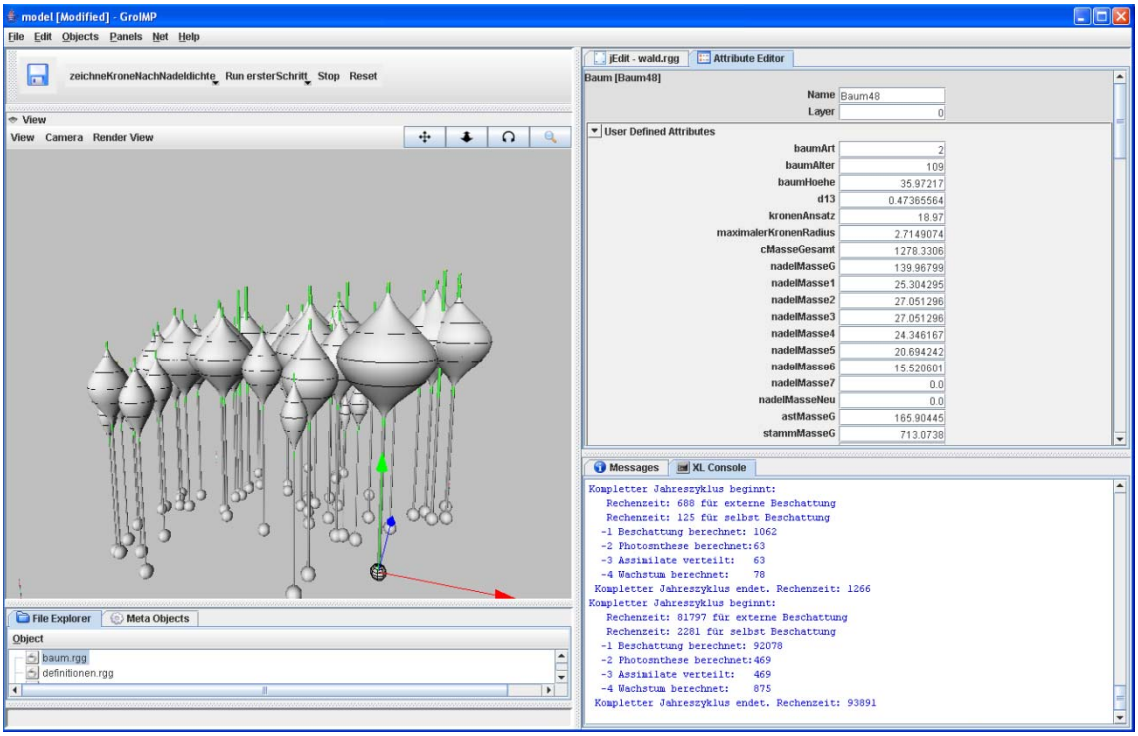


Abbildung 18: Die Simulationsumgebung von GroIMP mit dem Visualisierungsfenster in der OpenGL Ansicht (o. l.), dem Attribute-Editor (o. r.), dem Message Panel (u. r.) und dem ExplorerFenster mit aktiviertem File Explorer (u. l.). Im Attribute-Editor (o. r.) sind die Eigenschaften des ausgewählten Baumobjektes (o. l. erkennbar durch die Richtungsvektoren und die Wireframe-Ansicht) erkennbar.

GroIMP stellt darüber hinaus noch weitere umfangreiche Toolboxen und Userdialoge bereit, die eine komfortable und effiziente Modellierumgebung schaffen. So ist z. B. die in Abbildung 16 verwendete Ansicht zur Darstellung der Graphen-Struktur mit dem Attribute-Editor verbunden. Über das "Objects"-Menü können Primitivobjekte direkt in die Szene eingefügt werden. Die Toolbar stellt graphische Werkzeuge für das Skalieren, Verschieben und Rotieren von Objekten bereit.

Ein wichtiges Element für E-Learning-Szenarien ist der eingebaute HTML-Viewer. Er eröffnet die Möglichkeit, modellbezogenes Lehrmaterial in die GroIMP Oberfläche direkt zu integrieren. Über eine spezielle Linksyntax

Bsp. 3.6-8

```
<a href="command:/rggtutorial/selectfiles?pfs:Koch.rgg">
<a href="command:/workbench/rgg/toolbar/apply/derivation">
```

kann aus dem Tutorium (Abbildung 19 rechts) heraus die Durchführung einer Simulation detailliert gesteuert werden. Dies bildet eine gute Grundlage für modellbezogenes Lernen und ist ein wesentlicher Faktor auf dem Weg, ein tieferes Verständnis der verwendeten Modelle zu vermitteln.

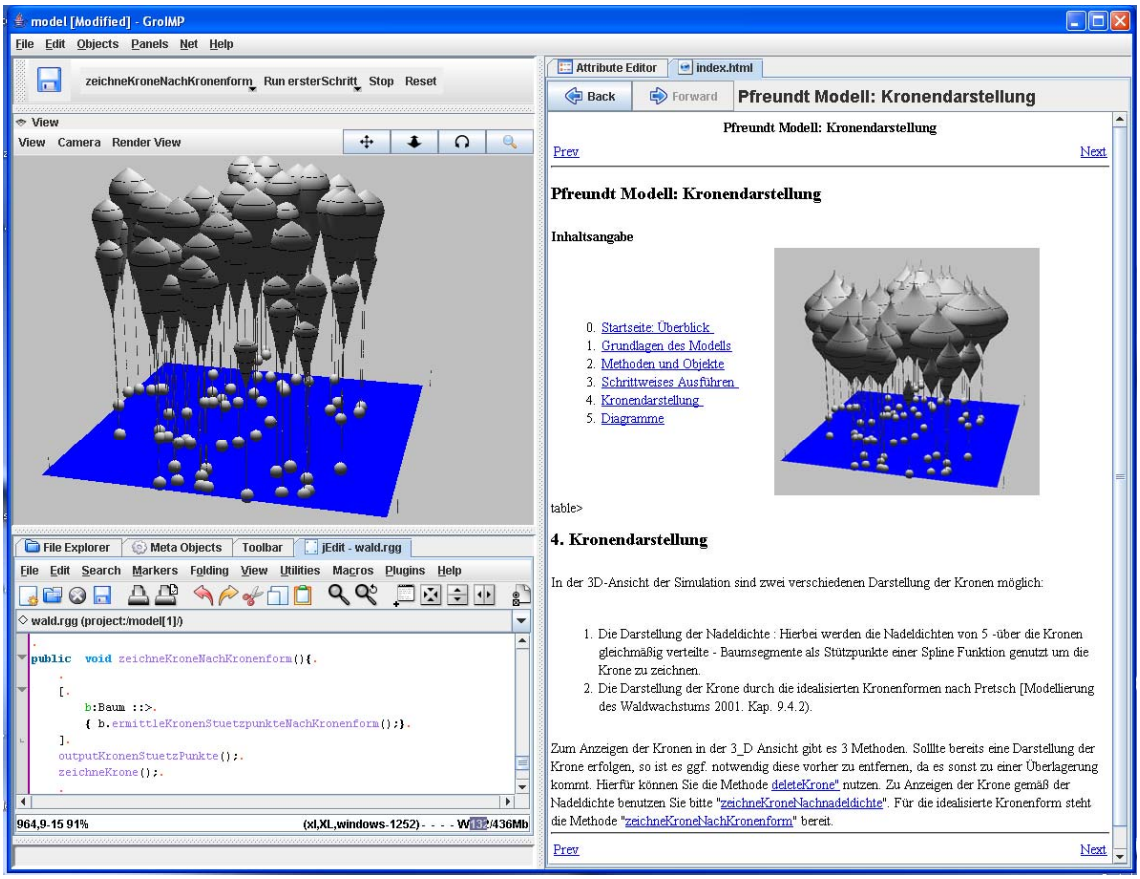


Abbildung 19: Der in GroIMP eingebaute HTML-Viewer erlaubt die Verzahnung von Tutorium und Modell. Modelleditor, Tutorium und 3D-Ansicht bilden eine gemeinsame UserOberfläche. Die Simulation kann über spezielle Hyperlinks aus dem Tutorium heraus gesteuert werden.

In der Standardinstallation von GroIMP enthalten sind ein RGG-Tutorium und zahlreiche Modellbeispiele zur Verdeutlichung des RGG/XL-Paradigmas. Die Erstellung von Tutorien einzelner Modelle kann von den Modell-Autorinnen mit geringen HTML-Kenntnissen selbst erfolgen.

Von besonderer Bedeutung für die in dieser Arbeit beschriebene Anwendung ist der integrierte HTTP-Server. Er wurde von Ole Kniemeyer in GroIMP für das Hyper-Text Transfer-Protokoll 1.1 [Fielding et al. 1999] implementiert, um dem Wunsch des Autors zu entsprechen, GroIMP per HTTP an den Elan Sim-Server über das Internet anbinden

zu können. Der abgehörte Port und das Server-Rootverzeichnis kann über die Programmvoreinstellungen (Abbildung 20) frei definiert werden. Die notwendigen Schritte zur Integration in ein Wachstumsmodell werden in Kapitel 3.7.5.1 beschrieben.

Für die vorliegende Arbeit wurde die Version 0.9.4 von GroIMP verwendet. GroIMP ist eine open-source-Software und wird unter den Bedingungen der GNU General Public Licence [GNU:GPL 1991] zur Nutzung freigegeben. Die jeweils aktuelle Version von GroIMP liegt unter http://www.grogra.de. Dort befindet sich auch die aktuelle Javadoc [Sun:Javadoc]-generierte Klassendokumentation. Für die Verwendung von GroIMP wird Java benötigt. Gegenwärtig empfohlen wird die Version 1.5.11 [Sun:Java]. Für die Verwendung der OpenGL®-Ansicht wird die jogl [Java.net:jogl]-Erweiterung von Java benötigt.

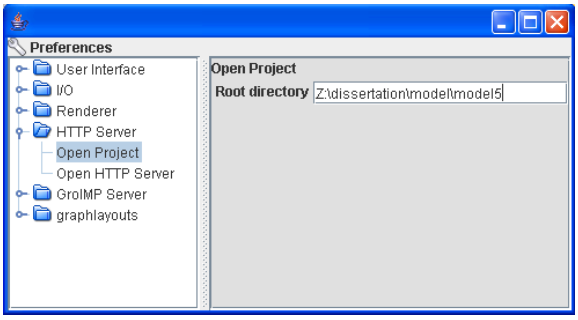


Abbildung 20: Dialog zur Spezifizierung des Root-Verzeichnisses des integrierten http-Servers.

3.7 Das Wachstumsmodell

3.7.1 Grundlagen des Modells

Das für diese Arbeit verwendete Wachstumsmodell basiert auf einem Ansatz, der von Pfreundt und Sloboda für die Fichte entwickelt wurde [Pfreundt 1988, Sloboda & Pfreundt 1989, Pfreundt & Sloboda 1996] und gehört in die Klasse der einfachen ökophysiologischen Modelle [vergl. Pretzsch 2001, S. 172ff]. Sloboda und Pfreundt schlugen vor, die Nettophotosyntheseleistung von Einzelbäumen in einem Bestand in Jahresschritten zu berechnen. Basis dieser Berechnung ist die – mit einer Betafunktion vertikal entlang der Stammachse verteilte – Nadelmasse. Diese wird genutzt, um für jeden Baum an mehreren Punkten die beschattende Nadelmasse innerhalb eines nach oben offenen Lichtkegels zu ermitteln. Als Eingangsgröße einer Beer-Lambert-Funktion [Monsi & Saeki 1953] wird mit diesem Wert die maximale photosynthetische Kapazität an der Kronenspitze den Beschattungsverhältnissen an den Berechnungspunkten angepasst und so mittels linearer Interpolation zwischen diesen Punkten die Bruttoassimilation des Baumes berechnet.

Die Respiration wird getrennt für die Baumkompartimente Nadeln, Äste, Stamm, Grob- und Feinwurzeln berechnet. Es wird zwischen Erhaltungsrespiration und Wachstumsrespiration unterschieden [vergl. Pretzsch 2002, S. 225].

Der aus Bruttophotosyntheseleistung – Erhaltungsrespiration resultierende Nettoassimilatspeicher bildet die Grundlage für das Wachstum. Zuvor werden für Wurzeln, Äste und Nadeln Sterberaten berücksichtigt. Nach Abzug der Assimilatmengen für Nadel-, Ast-, Wurzel- und Höhenwachstum bestimmt dann die verbleibende Menge im Assimilatspeicher das Dickenwachstum des Stamms.

Nach Abschluss des Wachstums werden dann die neuen Nadeldichten für das nächste Jahr berechnet. Die Berechnung der Nadeldichten erfolgt getrennt für die einzelnen Nadeljahrgänge. Hierbei werden die neue Baumhöhe, die Sterberaten der alten Nadeljahrgänge, die neu gebildete Nadelmasse sowie eine Verschiebung des Kronenansatzpunktes berücksichtigt.

Das im Rahmen dieser Arbeit entwickelte Modell folgt im Groben dem von Sloboda und Pfreundt beschriebenen Ansatz, weicht im Detail aber teilweise erheblich davon ab. So werden insbesondere bei der Berechnung der Respiration und der Assimilatverteilung für die einzelnen Kompartimente andere Berechnungsansätze verwendet. Weitere wesentliche Unterschiede ergeben sich durch die Implementierung des Modells mit XL/Java und die objektorientierte Umsetzung (siehe Kapitel 3.7.2ff).

3.7.2 Grundlagen der Umsetzung in XL

Die im Kapitel 3.6 zum Wachstumsgenerator aufgeführten Eigenschaften der Relationalen Wachstumsgrammatiken und der regelbasierten Sprache XL bestimmen wesentlich die Form der Umsetzung eines ökophysiologischen Modellansatzes für Bestandeswachstum, wie er im vorherigen Kapitel kurz beschrieben wurde. An dieser Stelle sollen daher kurz die Prämissen definiert werden, welche dem in den folgenden Kapiteln beschriebenen Modell zugrunde liegen.

Das vorliegende Modell folgt einem weitestgehend objektorientierten Ansatz. So besteht ein Baum aus einer Reihe von Segmenten, deren Eigenschaften – wie z.B. die Nadelmassen – als Initialwerte zugewiesen oder im Laufe der Simulation berechnet werden. Alle Berechnungen, wie Beschattung und Photosyntheseleistung, beruhen auf diesen Objekten. Die Ergebnisse werden in den Objekteigenschaften gespeichert. Die räumliche Beschaffenheit dieser Objekte beeinflusst somit auch die Rechengenauigkeit.

Alle Objekte sind in Graphen organisiert. Die Graphen sind innerhalb dieses Modells nicht zyklisch. Die Objektstruktur des Modells wird dabei analog einer botanischen Objektstruktur gehalten (Bestand -> Baum -> Jahrestrieb).

Die Berechnungsschritte innerhalb eines Simulationslaufes werden soweit wie möglich durch Anwendung Relationalen Wachstumsgrammatiken auf die Graphenstruktur des Modells organisiert und durchgeführt. Dies soll den Code einfach, verständlich und nahe an der "botanischen Struktur" halten.

3.7.3 Objektstruktur des Modells

Das Modell setzt sich aus 3 Hauptmodulen und 4 Hilfsmodulen zusammen:

Das Hilfsmodul **Nullpunkt** definiert den räumlichen Ursprung der gesamten Szene. Das Objekt wird nicht gezeichnet.

Die Hauptmodule **Bestand**, **Baum** und **Segmente** definieren die für den Simulationslauf notwendigen Objekte.

Die Hilfsmodule **Pfosten** und **Krone** dienen der besseren visuellen Darstellung des Modells und der Simulationsergebnisse. Für die Modellrechnung an sich werden Objekte dieses Typs nicht benötigt.

Das Hilfsmodul **Referenz** definiert die linke untere Ecke eines Bestandes. Es dient ausschließlich dem Zweck, die Bäume entsprechend Ihrer VRML-Koordinaten zu positionieren.

Als für die Modellberechnung nicht relevant werden die Hilfsmodule Referenz, Pfosten und Krone in diesem Kapitel nicht weiter beschrieben.

Die Module sind wie folgt aufgebaut.

Nullpunkt

Definition

Bsp. 3.7-1
```
module Nullpunkt() extends NULL
```

Das Modul Nullpunkt erweitert die Klasse Null aus de.grogra.imp3d.objects. Es verfügt über keine Aufrufparameter. Das Modul wird nur durch ein Objekt instanziiert und setzt einen Referenzpunkt in der Mitte der Modellfläche. Dieser Referenzpunkt dient als Mutterelement für die Bestandesobjekte.

Darstellung

Eine Darstellung erfolgt nicht.

Bestand

Definition

Bsp. 3.7-2
```
module Bestand( Point3d mittelpunkt, float xseite,
        float yseite,  boolean aktiv)  extends  Null()
        {setTransform(mittelpunkt); }
```

Das Modul Bestand erweitert ebenfalls die Klasse `Null` aus `de.grogra.imp3d.objects`. Es verfügt über 4 Aufrufparameter:

`Point3d mittelpunkt`	definiert den Bestandesursprung (linke untere Ecke) über X,Y,Z-Koordinaten relativ zum Mutterelement.
`float xseite`	definiert die Ausdehnung der Bestandesgrundfläche in Richtung der X-Achse je Richtung.
`float yseite`	definiert die Ausdehnung der Bestandesgrundfläche in Richtung der Y-Achse je Richtung.
`Boolean aktiv`	unterscheidet den Hauptbestand von kopierten Beständen zur Lösung von Randeffekten.
`{setTransform(referenzpunkt);}`	führt eine relative Koordinatenverschiebung des Bestandesbezugspunktes bei der Erzeugung des Objektes durch.

Es gibt eine Instanz des Moduls Bestand (s. Anhang 10.2.2), die bei Modellstart angelegt wird. Während der Simulationsläufe werden weitere temporäre Instanzen zur Lösung der Randeffekte angelegt und wieder gelöscht.

Bsp. 3.7-3
```
module Baum(int bnr, super.baumAlter, super.baumHoehe,
        super.d13, super.baumArt, super.kronenAnsatz,
        super.biomasse) extends  Baumklasse(baumAlter,
        baumHoehe, d13, baumArt, kronenAnsatz, biomasse);
```

Darstellung

Die Objekte der Klasse Bestand sind nicht sichtbar. Die Eckpunkte der Bestandesfläche werden aber durch die Objekte des Moduls Pfosten angezeigt (s. Anhang 10.2.2).

Baum

Definition

Das Modul "Baum" erweitert die Klasse "Baumklasse" aus baum.rgg. Diese Klasse wurde als Teil des vorliegenden Projektes geschrieben und beinhaltet alle baumspezifischen Eigenschaften und Methoden (siehe. Anhang 10.2.3).

Das Modul besitzt 7 Aufrufparameter, von denen 6 an den Konstruktor der Mutterklasse weitergegeben werden. Diese Aufrufparameter bestimmen die individuellen Eigenschaften der Bäume.

`int bnr`	erlaubt die Angabe einer ID für jede Instanz des Moduls Baum in dem Bestand.
`super.baumAlter`	definiert das Baumalter ganzzahlig in Jahren (j).
`super.Baumhöhe`	definiert die Baumhöhe in Metern (m).
`super.d13`	definiert den Stammdurchmesser in 1.3 Meter Baumhöhe in Metern (m).
`super.BaumArt`	definiert über einen ganzzahligen Wert die Baumart. Die Wertedefinitionen für die Baumarten sind in der statischen Klasse Definitionen (siehe. Anhang 10.2.1) festgelegt.
`super.kronenAnsatz`	definiert den Kronenansatzpunkt in Metern (m).
`super.biomasse`	definiert die Biomasse des Baumes in Kilogramm Kohlenstoff (kgC).

Die einzelnen Instanzen des Moduls Baum werden beim Start des Modells als Töchter eines Bestandesobjektes erzeugt. Sie bekommen durch diese Regel auch ihre Position im Bestand zugewiesen. Die Position ist relativ zum Bestandesreferenzpunkt.

Weitere Baumobjekte werden erzeugt, wenn zur Lösung der Randeffekte das Bestandesobjekt mit allen Subgraphen temporär kopiert wird. Die Baumobjekte werden mit ihren Mutterobjekten auch wieder gelöscht.

Darstellung

Die Bäume sind Objekte der "Baumklasse", welche eine Erweiterung der de.grogra.imp3D-Klasse "Sphere" ist.

Segment

Definition

Bsp. 3.7-4:
```
module Segment(int bnr, int snr, super.length, float base)
                extends  Segmentklasse(length);
```

Das Modul "Segment" erweitert die Klasse Segmentklasse aus der Datei segment.rgg. Auch diese Klasse wurde für dieses Projekt geschrieben und enthält alle segmentspezifischen Eigenschaften und Methoden (vergl. Anhang 10.2.4).

Das Modul enthält 4 Aufrufparameter, von dem der Parameter length an den Konstruktor der Mutterklasse weitergegeben wird. Ein Segment findet seine Analogie in der Botanik in einem Astquirl oder einem Jahrestrieb. Diese Analogie wird aber dadurch gebrochen, dass aus Gründen der Genauigkeit eine maximale Länge (length) für Seg-

mente definiert wird. Überschreitet das Längenwachstum eines Jahres diesen Wert, wird das Segment so lange in gleich große Segmente aufgeteilt, bis die Maximallänge unterschritten wird (siehe Kapitel 3.7.5.4).

`int bnr`	definiert den Identifier des Baum-Mutterobjektes.
`int snr`	erlaubt die Definition eines ganzzahligen Identifiers für das jeweilige Segment in der Reihenfolge der Segmenttochterelemente eines Baumelementes. Er wird gezählt von der Wurzel zur Baumspitze.
`super.length`	definiert die Länge des Segmentes in Metern (m).
`float base`	definiert die Höhe der Basis des Segmentes in Metern (m) relativ zum Stammfuß entlang der Z-Achse des Baumes.

Die Instanziierung von Segmentobjekten erfolgt als Reihe von Tochterelementen eines Baumelements. Sie definieren die räumliche Struktur der Bäume – wenngleich diese nur aus einer Achse besteht. Die Zahl der Segmente ist Abhängig von der definierten maximalen Segmentlänge, der Baumhöhe und der Anzahl der Bäume pro Bestand. Für die kopierten Bestände zur Lösung von Randeffekten gelten dieselben Aussagen wie für Baumobjekte.

Darstellung

Segmente sind Objekte der Klasse Segmentklasse und bilden eine Erweiterung der Klasse de.grogra.lsystem.F. Als solche werden sie als einfache Zylinder dargestellt.

3.7.4 Ablauf der Simulation

Wie in Kapitel 3.7.1 beschrieben, erfolgen in dem hier verwendeten Modellansatz die Berechnungen in Jahresschritten. Für einen Simulationslauf gibt es also 3 Phasen:

Phase 1 umfasst den Aufbau und die Parametrisierung der notwendigen grundlegenden Bestandesstruktur. Hierbei werden die Baum- und Bestandesparameter aus der VRML-Bestandesszene in eine genormte Forester-Description-Objektstruktur eingelesenen und anschließend die entsprechenden Objekte für das Modell erzeugt und mit den notwendigen Parametern initialisiert. Die in der Bestandesszene enthaltenen Parameter werden hierfür mittels der Literatur entnommener baumartenspezifischer Methoden auf die Baumkompartimente und die Segmente herunter gebrochen.

Die Phase 1 der Simulation umfasst folgende Arbeitsschritte:

1. Erzeugen der Struktur
    a. Auslesen der Bestandesbeschreibung
    b. Anlegen der Bäume des Bestandes
    c. Initialisieren der Bäume
    d. Anlegen der Segmente der Bäume
    e. Erste Initialisierung der Segmente

Phase 2 umfasst die Berechnungen, die für den vorgegebenen Simulationszeitraum in Jahreszyklen wiederholt werden. Im Groben sind dies 4 Schritte, die so lange wiederholt werden, bis der vorgegebene Simulationszeitraum berechnet ist. Die baumartenspezifischen Methoden und Parameter für diese Berechnungen wurden ebenfalls der Literatur entnommen.

Die Phase 2 umfasst folgende Arbeitsschritte:

2. Kalkulieren der Beschattung
    a. Kalkulieren der externen Beschattung
    b. Kalkulieren der Selbstbeschattung
3. Kalkulieren der Nettophotosynthese
    a. Kalkulieren der Bruttophotosynthese
    b. Kalkulieren der Respiration
4. Verteilen der Assimilate auf die Kompartimente
5. Berechnen des Wachstums
    a. Höhenwachstum
    b. Neuer Kronenansatzpunkt
    c. Update der Baumdaten
    d. Reinitialisieren der Segmente

Phase 3 umfasst den Abschluss einer Simulation. Der simulierte Bestand wird in die genormte Forester-Description-Objektstruktur übertragen und abschließend als VRML Bestandesszene abgespeichert.

Die Phase 3 umfasst folgende Arbeitsschritte:

6. Erstellen der neuen Bestandesbeschreibung

In den folgenden Kapiteln wird das Modell gemäß dieser Gliederung erläutert. Hierzu wird erst auf die theoretischen Grundlagen eingegangen und anschließend die Implementierung aufgezeigt.

3.7.5 Erzeugen der Struktur

3.7.5.1 Auslesen der Bestandesszene

Das Auslesen der VRML-Bestandeszene erfolgt über speziell für diesen Zweck geschriebene Java-Klassen, die als Erweiterung von GroIMP in das Modell eingebunden werden. Da diese Klassen einen normierten Weg zum Import/Export von Forester-Bestandesbeschreibungen in XL/Java-Modelle darstellen und im engeren Sinne kein Teil des Modells sind, werden sie im Kapitel über die Kommunikation Forester – GroIMP – Modell (Kapitel 3.8) behandelt.

3.7.5.2 Anlegen der Bäume des Bestandes

Auf Basis der ausgelesenen Bestandesszene wird für das Modell ein Bestandes-Objekt mit der entsprechenden Anzahl von Baum-Objekten erzeugt. Die Aufrufparameter der Baum-Objekte: ***Baumnummer***, ***Alter***, ***Baumart, Höhe***, ***BHD***, ***Kronenansatzpunkt*** sowie die relativen ***x, y, z-Koordinaten*** zum Bestandesmittelpunkt werden der externen Bestandesszene entnommen.

3.7.5.3 Initialisieren der Bäume

Die Initialisierung der Baumeigenschaften erfolgt durch den Konstruktor der Klasse Baumklasse. Um die Bäume und im späteren Verlauf die Baumsegmente und die ver-

schiedenen Berechnungsmethoden der Simulation zu parametrisieren, werden neben den übergebenen Aufrufparametern aus der Bestandesszene (vergl. Kapitel 3.7.5.2) im Modell zahlreiche baumartenspezifische Parameter verwendet.

Alle modell- und baumartenspezifischen Parameter sind in einer Klasse Definitionen als Konstanten definiert und zusammengefasst. Bei der Konstruktion des Baum-Objektes werden diese Parameter auf Baum-Eigenschaften übertragen und können ab dort individuell geändert werden. Das Modell ist auf diesem Weg grundsätzlich so angelegt, dass eine beliebige Anzahl von Baumarten verwendetet werden kann. Der im Anhang beigefügte Quellcode der Klasse Definitionen zeigt die Anwendung für die drei Baumarten Douglasie (Feldindex 0), Kiefer (Feldindex 1) und Fichte (Feldindex 2). Durch eine entsprechende Anpassung der Klasse Definitionen können leicht weitere Baumarten hinzugefügt werden.

Wichtig ist es aber zu bedenken, dass der zugrunde liegende Modellansatz von Sloboda & Pfreundt [1989] für einen Fichtenreinbestand konzipiert wurde. Das vorliegende Modell versucht, mit Hilfe verschiedener Quellen aus der Literatur die Parametrisierung für die Fichte (Picea abies (L.) Karst) zu vervollständigen und für die Kiefer (Pinus silvestris L.) zu ergänzen. Ein Teil der Parameter – soweit keine validierten Daten vorlagen – musste aber aus dem Modellverlauf heraus geschätzt werden. Der Parametersatz für die Douglasie ist völlig fiktiv und dient der Möglichkeit, im Rahmen des Lernprozesses mit den Modellparametern zu experimentieren.

Wesentlich für die Initialisierung der Baum-Objekte ist die Berechnung der Baumbiomasse und die Aufteilung der Biomasse auf die 5 Kompartimente. Alle in den Modellrechnungen verwendeten Biomassen beziehen sich auf den C-Anteil (kgC) des Trockengewichts. Zuerst werden hierzu die Stammparameter berechnet.

Stammparameter

Ausgangsgröße für die Berechnung der Stammparameter sind die dem Baum-Objekt übergebenen – und der Forester-Bestandesszene entnommenen – Eigenschaften Höhe h (m) und Brusthöhendurchmesser $d_{1,3}$ (m). Daraus abgeleitet werden die Stammbiomasse M_{stamm} (kgC) und das Stammvolumen v_{stamm} (m^3). Hierzu werden die baumartenspezifischen Parameter für die Holzdichte ρ_{holz}, der C-Anteil an der Biomasse cQuote (kgC/kgBiomasse) und die unechte Schaftholzformzahl $f_{1,3}$ nach Kennel [1969, zitiert in Pretzsch 2002 S. 171] verwendet.

Die unechte Schaftholzformzahl stellt ein relatives Maß dar, inwieweit das Volumen des Stammes von dem Volumen eines Zylinders mit dem Durchmesser $d_{1,3}$ und der Höhe h abweicht.

Es definieren sich daraus folgende Beziehungen:

(6) $$v_{stamm} = \pi \cdot \frac{d_{1,3}^{\,2}}{4} \cdot h \cdot f_{1,3}$$ und

(7) $$M_{stamm} = v_{stamm} \cdot cQuote \cdot \rho_{Holz}$$

mit $f_{1,3}$ als der unechten Schaftholzformzahl. Sie wird mit einem baumartenspezifischen Parametersatz $k_1..k_7$ und mit Bezug zum Stammdurchmesser d (in cm; d=$d_{1,3}$*100) und der Baumhöhe h (m) wie folgt berechnet:

(8) $$f_{1,3} = k_1 + k_2 / d + k_3 / h + k_4 / d^2 + k_5 /(d \cdot h) + k_6 /(d \cdot h)^2 + k_7 \cdot \ln^2(d)$$

Die Werte für die Koeffizienten $k_1..k_7$ für die drei Baumarten Douglasie (Pseudotsuga menziesii Mirb.), Kiefer (Pinus silvestris L.) und Fichte (Picea abies (L.) Karst) lauten [Pretzsch 2002, S. 171]:

	k1	k2	k3	k4	k5	k6	k7
Douglasie	0.3398	3.9392	0.0000	-16.5646	-17.8652	117.1410	0.0000
Kiefer	0.5858	-0.2693	-0.4593	0.000	5.5060	-5.1390	0.0116
Fichte	0.5073	1.5685	-0.4237	-9.1576	3.5585	17.3395	-0.0068

Tabelle 6: Koeffizienten zur Berechnung der unechten Schaftholzformzahl nach Kennel für Douglasie (Pseudottsuga menziesii Mirb.), Kiefer (Pinus silvestris L.) und Fichte (Picea abies (L.) Karst) [Kennel 1969 zitiert von Pretzsch 2002].

Wenn eine Funktion zur Beschreibung der Stammform vorhanden ist, kann die Stammoberfläche über ein Integral der Kreisumfänge beschrieben werden mit

(9) $$o_{stamm} = \int_0^h 2 \cdot \pi \cdot f(x) \cdot \sqrt{1 + f'(x)^2} \cdot dx$$

wobei $f(x)$ den Stammradius an der Stelle x angibt. Im vorliegenden Modell wird die Stammoberfläche aber über den Durchmesser, die Baumhöhe und die Schaftholzformzahl näherungsweise berechnet, mit:

(10) $$O_{stamm} = \pi \cdot d_{1,3} \cdot h \cdot \sqrt{f_{1,3}}$$

Die Stammoberfläche wird nur für die Berechnung der Stammrespiration verwendet. Da die Respiration der Holzteile im Verhältnis zur Gesamtrespiration des Baumes von geringer Bedeutung ist, werden die Fehler dieser Methode der Stammoberflächenberechnung akzeptiert.

Den in Formel (6) und (7) beschriebenen Beziehungen kommt im vorliegenden Modell insofern eine zentrale Bedeutung zu, als über den Umweg der Stammbiomasse auf die Gesamtbiomasse des Baumes zurück gerechnet wird (Bsp. 3.7-5) und somit auch die Massen der übrigen Kompartimente festgelegt werden.

Bsp. 3.7-5: Berechnung der Gesamtbiomasse über die Stammmasse

```
stammMasseG = kalkuliereStammCMasse();
cMasseGesamt = stammMasseG
       / this.kompartimentAnteile[Definitionen.STAMM];
nadelMasseG = cMasseGesamt
       * this.kompartimentAnteile[Definitionen.NADEL];
astMasseG = cMasseGesamt * this.kompartimentAnteile[Definitionen.AST];
```

Die Biomasse wiederum ist die wesentliche Grundlage für die später ausgeführten Berechnungen der Photosynthese, der Respiration und des Wachstums und beeinflusst somit den gesamten Verlauf der Simulation.

Biomassen der Kompartimente

Die Aufteilung auf die 5 Kompartimente (Tabelle 7) wurde für Fichte und Kiefer einer Untersuchung in Finnland entnommen [Muukkonen et al. 2004]. Als vereinfachende Annahme werden die Anteile der Kompartimente als konstant angesehen. Dies erfolgt unter der Prämisse, dass der vorliegende Modellansatz für sehr junge Bäume wenig Relevanz besitzt.

	Blätter	*Äste*	*Stamm*	*Feinwurzeln*	*Grobwurzeln*
Kiefer	6%	13%	62%	2%	1%
Fichte	11%	16%	53%	3%	18%

Tabelle 7: Prozentuale Anteile der 5 Kompartimente an der Gesamtbiomasse des Baumes [abgeleitet aus Muukkonen et al. 2004].

Die Kompartimente Fein- und Grobwurzeln, Stamm und Äste werden ausschließlich über ihre Biomassewerte (kgC) als Eigenschaft des Baumes repräsentiert und im Baum-Objekt gespeichert. Das Kompartiment der Nadeln wird zuerst auf die einzelnen Nadeljahrgänge aufgeteilt, jahrgangsweise im Baum-Objekt gespeichert und später zur Initialisierung der Baumsegmente genutzt.

Für das Kompartiment Nadelmasse wird eine vertikale Verteilung entlang der senkrechten Stammachse angenommen. Die Nadelmasse wird hierbei ausschließlich durch Punkte auf der Stammachse repräsentiert. Eine horizontale Ausdehnung wird nicht berechnet. Die Zahl der Nadeljahrgänge und der Aufteilungsschlüssel der Masse auf die Nadeljahrgänge wird durch baumartenspezifische Überlebensraten der Nadeljahrgänge spezifiziert (siehe Tabelle 9: Kapitel 3.7.11).

Eine weitere Zustandsgröße für das Nadelkompartiment ist die vertikale Ausdehnung des Kronenraums. Hierzu wird für jeden Nadeljahrgang die untere und obere Benadelungsgrenze gespeichert. Beim Anlegen der Struktur wird für alle Nadeljahrgänge der Kronenansatzpunkt des Baumes als untere Benadelungsgrenze festgesetzt. Die obere Benadelungsgrenze wird mit der Kronenspitze des jeweiligen Jahres der Nadelbildung festgelegt. Zur Rückberechnung der Baumhöhen der letzten Jahre wird ein lineares Wachstum angenommen.

(11) $$\text{Baumhöhe}_j = \text{Baumhöhe}_1 - ((j-1) * (\text{Baumhöhe}_1 / \text{Baumalter}))$$

mit $j = 1$ für den aktuellen Nadeljahrgang und $j = 2$ bis zum maximalen Nadeljahrgang für die vorjährigen Nadeljahrgänge (siehe auch Kapitel 3.7.5.4). Die im Baum-Element gespeicherten Werte des Nadelkompartiments bilden die Grundlage für die Berechnung der vertikalen Verteilung der Nadelmassen.

Kronendurchmesser

Die Kronendurchmesser-Eigenschaft der Baum-Objekte wird mit aus der Forester-Bestandesbeschreibung ausgelesenen Werten beim Anlegen der Bäume initialisiert. Ist im Forester-Description-Objekt keine entsprechender Wert angegeben, wird der Vorgabewert KRONENDURCHMESSERDEFAULTWERT – aktuell 3.16 Meter für alle Baumarten – aus der Definitionen-Klasse genommen. Der Kronendurchmesser wird ausschließlich als Korrekturgröße zur Nadelmassenverteilung bei der Gewichtung der Beschattungswirkung der Nadelmasse über die Distanz zum Berechnungssegment genutzt.

3.7.5.4 Anlegen der Segmente der Bäume

Die Segmente bilden die vertikale Struktur des Baumes. Sie konstruieren, als eine 1-dimensionale Reihe von Tochterelementen eines Baumelements, die senkrechte Stammachse des Baumes. Sie werden definiert durch einen Basispunkt und eine Länge. Die Segmente sind die Träger der Nadelmasse. Ihre Länge bildet die Schrittweite bei der Diskretisierung der Nadelverteilung.

Des Weiteren sind die Segmente Träger der Berechnungen für Beschattung, Photosynthese und Nadelrespiration. Eine Übersicht der Eigenschaften der Segmente findet sich im Code der Klasse Segment (Anhang 10.2.4).

Die Zahl und Länge der Segmente wird bestimmt durch eine vordefinierte Maximallänge pro Segment, das Baumalter und die Baumhöhe. Hierzu wird die Segmentanzahl als das kleinste Vielfache des Baumalters gebildet, wobei der Quotient von Baumhöhe / Segmentzahl eine Segmentlänge ≤ der definierten Maximallänge ergibt (s. Formel (12)).

Dieses Verfahren stellt sicher, dass die durch die Annahme des linearen Wachstums ermittelten oberen Benadelungsgrenzen der Nadeljahrgänge (vergl. 3.7.5.3) immer mit einem Segmentende zusammenfallen. Das vereinfacht im späteren Verlauf verschiedene Berechnungen.

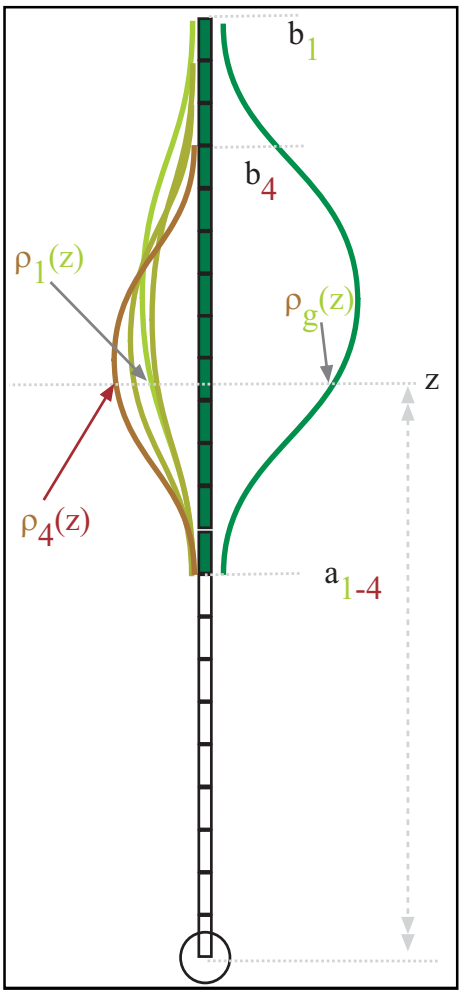


Abbildung 21: Betaverteilungen zur Beschreibung der Nadeldichte ρ für die Nadeljahrgänge 1-4.

(12) $\text{length(Seg)} = \text{Baumhöhe} / (\text{Baumalter} * \text{i}) ; \text{bis length(Seg)} \leq \text{maxlength}; \text{i}++$

3.7.5.5 Erste Initialisierung der Segment-Nadelmassen

Bei der Initialisierung der Segmente wird die Baum-Nadelmasse auf die Segmente übertragen. Nach jedem Jahresdurchlauf werden die Segmente mit der neuen Baum-Nadelmasse neu initialisiert (vergl. Kapitel 3.7.11.2).

Die Berechnung der Nadelmassen der Segmente erfolgt mit einer Verteilungsfunktion der Nadeldichten ρ_j (kgC/m) entlang der vertikalen Stammachse. Kellomäki et al. [1980] schlagen zur Beschreibung der vertikalen Verteilung der relativen Nadelbiomasse bei Kiefern eine Beta-Verteilung vor. Basierend auf Beobachtungen zur vertikalen Verteilung von Nadeloberflächen [Pfreundt 1988 S. 15] verwenden Sloboda und Pfreundt [1989] eine Verteilung des selben Typs zur Beschreibung der Nadelbiomasseverteilung bei Fichten.

Für das vorliegende Modell wird daher zur Beschreibung der Nadeldichten entlang der Stammachse eine Betaverteilung vom Typ:

(13) $$\rho_j(z) = M_j \cdot \frac{1}{B(p,q)} \cdot (z - a_j)^p \cdot (b_j - z)^q$$

verwendet, wobei gemäß Abbildung 21 z die jeweilige Höhe ist, M_j die Nadelmasse des Baumes für den Nadeljahrgang j, a_j der untere Kronenpunkt für den Nadeljahrgang j, mit $(a_j \leq z)$, b_j der obere Kronenpunkt für den Nadeljahrgang j, mit $(z \leq b_j)$, p und q die Lageparameter der Betaverteilung sind, mit $(p,q > 0)$ und $B(p,q)$ die Betafunktion

(14) $$B(p,q) = \int_{a_j}^{b_j} (z - a_j)^p (b_j - z)^q \, dz$$

zur Normierung.

Die Berechnung der Nadelmasse eines Segmentes m_{total} erfolgt getrennt für die einzelnen Nadeljahrgänge *j*. Hierzu wird die Nadeldichteverteilung mit den Segmenten als Intervall interpoliert. Zwischen den Werten für den Segmentbasispunkt und der Segmentspitze wird gemittelt.

Die Gesamtnadelmasse (kgC) eines Segmentes ergibt sich durch Aufsummierung der Jahrgangsmassen wie folgt:

(15) $$m_{total}(Seg) = \sum_{j=1}^{\max} \rho_j(basis(Seg)) + \rho_j(top(Seg)) / 2 * length(Seg)$$

Die Jahrgangsnadelmassen und die Gesamtnadelmasse werden als entsprechende Eigenschaften in den Segment-Objekten gespeichert.

Die Parametrisierung der Betaverteilungen erfolgt unter Verwendung der von Sloboda & Pfreundt vorgeschlagenen Werte von 2,5 für die Lageparameter p und q für alle Nadeljahrgänge und Baumarten. Untersuchungen haben aber gezeigt, dass die Nadelmasseverteilung zwischen den Nadeljahrgängen unterschiedlich sein kann [Kellomäki et al. 1980] und auch von der Klasse des Baumes beeinflusst wird [Kellomäki & Hari 1980]. Eine Analyse und eventuelle Verfeinerung des Modells an dieser Stelle ist ein weiterer möglicher Ansatzpunkt im Rahmen des Lernprozesses.

3.7.6 Grundlagen zur Beschattung und Photosyntheseberechnung

Sloboda und Pfreundt stellen fest: *"Der wichtigste Punkt in allen Photosynthese-orientierten Wachstumsmodellen ist die räumliche Verteilung der Lichtintensitäten innerhalb des Bestandes und die davon abhängigen Photosynthese-Kapazitäten"* [Sloboda & Pfreundt 1989 Seite 8]. Im Folgenden wird daher ein Verfahren beschrieben, mit dem diese räumliche Verteilung in das Modell einbezogen wird.

Die Photosyntheseleistung im Punkt $\vec{x}$ ist abhängig von der Lichtintensität I am Punkt $\vec{x}$. Gerechnet für einen Zeitraum t_0 -> t_1 ergibt sich:

(16) $$P(\vec{x}) = \int_{t_o}^{t_1} p(I(\vec{x},t))dt \;\; ; \vec{x} \in \Re^3$$

für einen Punkt $\vec{x}$ in der Baumkrone und

(17) $$P_0 = \int_{t_o}^{t_1} p(I_0(t))dt$$

für unbeschattete Verhältnisse an der Baumspitze.

Um die mit der im Bestand abnehmenden Lichtintensität ebenfalls abnehmende photosynthetische Kapazität über längere Zeiträume zu beschreiben, kann das Konzept der beschattenden Biomasse verwendet werden [vergl. Kellomäki et al. 1980, Mäkelä 1982].

Hierbei wird eine dimensionslose Produktionsrate q für den Punkt $\vec{x}$ im Bestand und dem Zeitraum t_0 -> t_1, relativ zur maximalen Photosynthesekapazität $p(I_0)$ bei unbeschatteten Verhältnissen I_0 definiert:

(18) $$q(\vec{x},t_0,t_1) := \frac{\int_{t_o}^{t_1} p(I(\vec{x},t))dt}{\int_{t_o}^{t_1} p(I_0(t))dt}; \vec{x} \in \Re^3$$ oder **(19)** $$q(\vec{x},t_0,t_1) := \frac{P(\vec{x})}{P_0}; \vec{x} \in \Re^3$$

Gehen wir davon aus, dass die Produktionsrate $q(\vec{x},t_0,t_1)$ eine stetige, monoton fallende Funktion der Blattmasse über $\vec{x}$ ist, ergibt sich

(20) $$q(\vec{x},t_o,t_1) = F(M(\vec{x}))\,,$$

wobei $M(\vec{x})$ die Nadelbiomasse über $\vec{x}$ ist (kgC). Dies setzt aber voraus, dass sich M im Zeitraum t_0 -> t_1 nicht ändert.

Die Annahme einer stetigen, monoton fallenden Funktion kann hierbei nur über längere Betrachtungszeiträume als gültig betrachtet werden. Kurzfristig sind Photosyntheseleistungen möglich, die in der Schattenkrone höher liegen als in der Lichtkrone. Dies kann z. B. durch hohe Photorespirationsraten bei sehr starker Sonneneinstrahlung begründet sein.

Im nächsten Schritt muss dieses Konzept auf ein räumliches Modell übertragen werden. Wir folgen wieder dem Ansatz von Sloboda und Pfreundt [1989]. Um die beschattende Biomasse $M(\vec{x})$ für einen Punkt $\vec{x}$ zu ermitteln, denken wir uns einen Kegel mit der Spitze in $\vec{x}$ und einer Weite α. Die innerhalb dieses Kegels liegende Biomasse sei die beschattende Biomasse von $\vec{x}$. Je nach Entfernung des Auftretens der Biomasse ist die beschattende Wirkung auf $\vec{x}$ unterschiedlich groß. Um ihren beschattenden Einfluss zu bestimmen, wird sie über ihre Projektionsfläche normiert. Flächenstücke werden daher mit dem reziproken Quadrat ihrer Entfernung zu $\vec{x}$ bewertet.

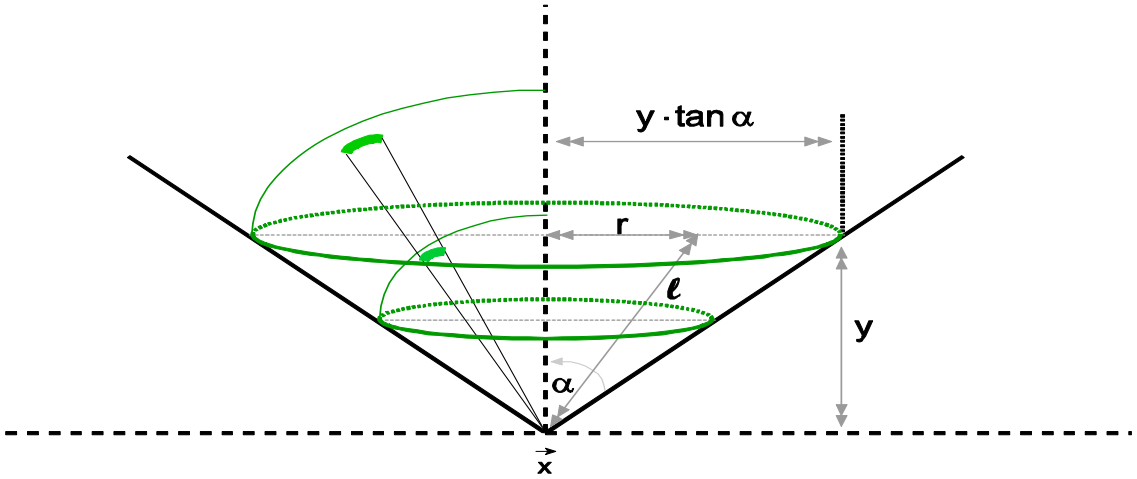


Abbildung 22: Aufbau eines Lichtkegels über Punkt x zur Ermittlung und Gewichtung der Beschattungswirkung von horizontal uniform verteilter Nadelbiomasse.

Bei einer horizontal uniformen Nadeldichte (kgC·m^{-2}·m^{-1}) ergibt sich aus der in Abbildung 22 skizzierten Darstellung folgende räumliche Betrachtung:

(21) $$B(z) = \int_{z}^{\infty} \int_{0}^{2\pi y \tan\alpha} \int_{0}^{} \frac{p(y)}{l^2} r \, \mathrm{d}r \, \mathrm{d}\varphi \, \mathrm{d}y \quad \textbf{oder} \quad B(z) = \int_{z}^{\infty} \int_{0}^{2\pi y \tan\alpha} \int_{0}^{} \frac{p(y)}{y^2 + r^2} r \, \mathrm{d}r \, \mathrm{d}\varphi \, \mathrm{d}y$$

Da in dem vorliegenden Modell – statt eine horizontal uniforme Nadeldichte anzunehmen – die Nadelmasse den einzelnen Bäumen in Segmenten zugeordnet ist, muss die Berechnung entsprechend angepasst werden.

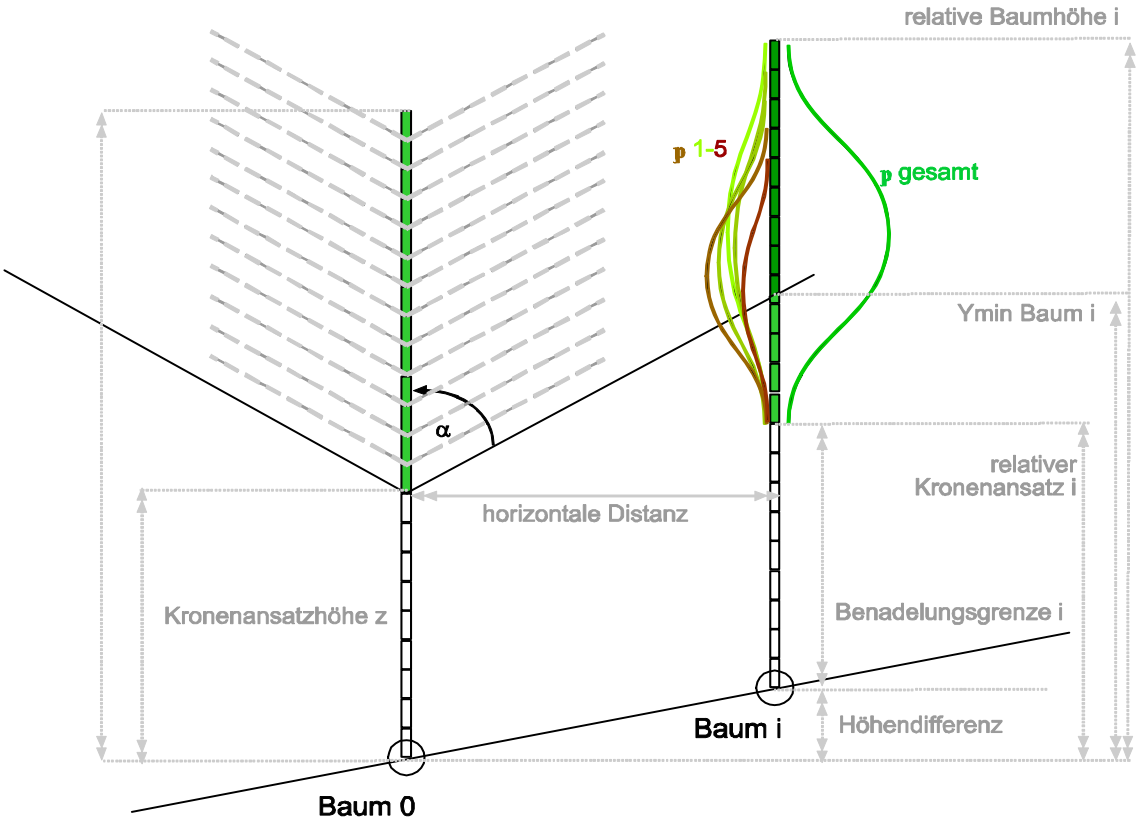


Abbildung 23: Berechnung der beschattenden Nadelbiomasse eines benachbarten Baumes i über dem Basispunkt z eines Baumsegmentes.

Abbildung 23 zeigt zwei beliebige, benachbarte Bäume. Für Punkt z auf der Stammachse von $Baum_0$ wird für einen Kegel mit dem Öffnungswinkel α die beschattende Nadelmasse des Nachbarn $Baum_i$ berechnet. Der Schnittpunkt Y_{min} des Kegels mit der Stammachse von $Baum_i$ ergibt sich wie folgt:

(22) $$Y_{\min,i} = MAX(Kronenansatz\_relativ_i, z + d_i \cdot \tan\frac{\pi}{2} - \alpha),$$

wobei der $Kronenansatz\_relativ_i$ die untere Benadelungsgrenze von $Baum_i$ (plus einer eventuellen Höhendifferenz der Stammfußpunkte zwischen den Bäumen) und d_i die horizontale Distanz zwischen $Baum_0$ und $Baum_i$ ist.

Die gewichtete, beschattende Nadelmasse des Nachbarn $Baum_i$ für den Punkt $\vec{x}$ ergibt sich durch:

(23) $$B_i(\vec{x}) = \int_{y_{\min,i}}^{h_i} \frac{p_i(y)}{\overline{l^2}} \mathrm{dy},$$

wobei h_i die relative Höhe von $Baum_i$ ist und $\overline{l^2}$ die mittlere quadrierte Entfernung zur beschattenden Biomasse für jedes y.

Die direkte Distanz zwischen Punkt $\vec{x}$ und einem Punkt $\vec{y}$ auf der Stammachse des Nachbarn $Baum_i$ kann über

(24) $$l^2 = d_i^{\,2} + (y - z)^2$$

berechnet werden, wobei z der Höhenwert des Punktes $\vec{x}$ auf der Stammachse von $Baum_0$, y der Höhenwert des Punktes $\vec{y}$ auf der Stammachse von $Baum_i$ (inkl. Höhendifferenz $Baum_0 - Baum_i$) und d_I die horizontale Distanz zwischen den Bäumen ist.

Zur Herleitung der mittleren quadratischen Distanz zu $\vec{x}$ betrachten wir

Abbildung 24. Es gilt, dass $\vec{x}$ und $\vec{y}$ auf einer gemeinsamen, horizontalen Ebene (y=z) liegen und wir so die vertikale Komponente vernachlässigen können.

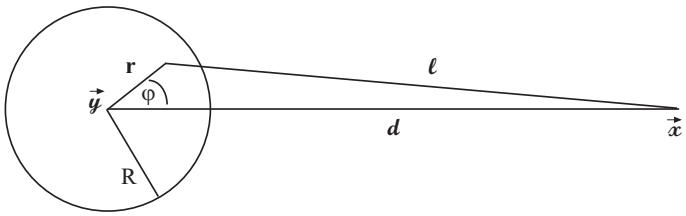


Abbildung 24: Beziehung von Kronenradius *(R)*, benadelter Fläche *(r)*, Distanz *(d)* und Länge *(l)* zur Berechnung der mittleren Länge *(l̄)* zu Punkt $\vec{x}$.

Für die Berechnung der mittleren quadratischen Entfernung unterscheiden wir nun drei Varianten:

Variante 1 geht davon aus, dass die Nadelmasse im Punkt $\vec{y}$ konzentriert ist. Es ergibt sich daraus:

(25) $\overline{l^2} = l^2 = d^2$; wenn (y=z) und Nadelmasse in $\vec{y}$ konzentriert.

Variante 2 geht davon aus, dass die Nadelmasse auf einer horizontalen Kreisfläche mit Radius R rund um $\vec{y}$ gleichförmig verteilt ist. Wir können nun die mittlere quadratische Entfernung von $\vec{x}$ mit dem Cosinus-Satz berechnen:

(26) $$\overline{l^2} = \int_0^{2\pi}\int_0^R l^2 \frac{1}{\pi R^2} \mathrm{r}\,\mathrm{dr}\,\mathrm{d}\varphi \qquad = \frac{1}{\pi R^2}\int_0^{2\pi}\int_0^R (d^2 + r^2 - 2rd\cos\varphi)\,\mathrm{r}\,\mathrm{dr}\,\mathrm{d}\varphi$$

$$= \frac{1}{\pi R^2}(\pi R^2 d^2 + 2\pi\frac{1}{4}R^4) \qquad = d^2 + R^2/2$$

Variante 3 geht davon aus, dass die Nadeln auf dem Kronenmantel, d. h auf dem Kreisumfang von R konzentriert sind. Man erhält dann:

(27) $$\overline{l^2} = \int_0^{2\pi} l^2 \frac{1}{2\pi}\,\mathrm{dr}\,\mathrm{d}\varphi \qquad = d^2 + R^2$$

Bei einer räumlichen Verteilung der Nadeln ist es also notwendig, die horizontale Distanz mit einem Additionsterm zu korrigieren, der vom Kronenradius R und der angenommenen Verteilung der Nadeln – hier ausgedrückt durch einen positiven Formfaktor – abhängig ist. Nehmen wir eine mögliche vertikale Distanz wieder in unsere Betrachtung hinein, ergibt sich für die Gewichtung der beschattenden Wirkung der Nadelmasse folgender Term:

(28) $$\frac{1}{(\mathrm{y}-\mathrm{z})^2 + d^2 + \frac{R^2}{cf}} \qquad \text{oder} \qquad \frac{1}{\mathrm{l}^2 + \frac{R^2}{cf}} \qquad 1 \le cf \le 2,$$

wobei $cf = 1$ für eine Konzentration auf dem Kronenmantel steht und $cf = 2$ eine gleichförmige Verteilung über R darstellt.

Setzen wir diesen Term nun in Gleichung (23) ein und erweitern die Betrachtung auf die gesamte beschattende Biomasse am Punkt $\vec{x}$, ergibt sich folgende Situation:

Gegeben sei $Baum_0$ mit n Nachbarn, deren Nadelmassen einen Punkt mit Höhe z auf der Stammachse von $Baum_0$ in einem Kegel mit dem Öffnungswinkel α beschatten. Zur Berechnung der mit der quadrierten Distanz gewichteten beschattenden Biomasse ergibt sich folgende Gleichung:

(29) $$B_0(\vec{x}) = \int_z^{h_0} \frac{p_0(y)}{(y-z)^2 + \frac{R_o^2}{cf}}\,\mathrm{dy} + \sum_{i=1}^{n}\int_{y_{\min,i}}^{h_i} \frac{p_i(y)}{(y-z)^2 + d_i^2 + \frac{R_i^2}{cf}}\,\mathrm{dy}$$

Y_{min} errechnet sich wie in Formel (22) angegeben.

Der erste Term steht hier für die Selbstbeschattung von $Baum_0$, Term 2 berechnet die aufsummierte gewichtete Beschattung durch die Nachbarn.

Analog zu Gleichung (20) wird für die photosynthetische Produktionsrate die Existenz einer Funktion F angenommen mit

(30) $$q(\vec{x}, t_o, t_1) = F(B(\vec{x})) \qquad \text{bzw.} \qquad q(\vec{x}) = F(B(\vec{x}))$$

mit einer Vegetationsperiode für t_0 -> t_1. Die Nadelmasse wird für die gesamte Vegetationsperiode wieder als konstant angesehen.

Sei P_0 nun die Photosyntheserate (kgC·kgC^{-1}·j^{-1}) für die gesamte Vegetationsperiode bei unbeschatteten Verhältnissen mit

(31) $$\mathrm{p}_0 = \mathrm{p}_0(\mathrm{t}_o, \mathrm{t}_1) = \int_{t_o}^{t_1} p(I_o(t))dt$$

so ergibt sich als jährliche Produktionsrate (kgC·kgC^{-1}·j^{-1}) für die 1-jährige Nadeln des Baumes am Punkt $\vec{x}$

(32) $$\mathrm{p}(\vec{\mathrm{x}}) = \int_{t_o}^{t_1} p(I(\vec{x}, t))dt \qquad = p_0 \cdot q(\vec{x}) \qquad = p_0 \cdot F(B(\vec{x}))$$

Die Funktion $F(B(\vec{x}))$ ist baumartenspezifisch und kann für Reinbestände mit empirischen Daten angepasst werden.

Für die Verwendung in einem Modell für Mischbestände bleibt bei dem beschriebenen Ansatz allerdings ein Problem ungelöst. Die in Formel (29) aufgezeigte Form der Berechnung der beschattenden Biomasse berücksichtigt als Korrekturterm der Beschattungswirkung nur die Distanz zum Punkt $\vec{x}$. Bereits Monsi & Saeki [1953] haben aber auf die unterschiedlichen Beschattungseigenschaften für verschiedene Pflanzengesellschaften hingewiesen. In einem Einzelbaummodell mit individuellen Lichtkegeln besteht also der Bedarf, die Beschattungswirkung der Nadelmasse unterschiedlicher Baumarten zu gewichten.

Anlehnend an den in SILVA verwendeten Konkurrenzindex KKL (beschrieben in Kapitel 2.2.2) wird vorgeschlagen, die Biomasse zusätzlich mit einem Lichttransmissionskoeffizienten TM [Pretzsch 2001 mit Bezug auf Ellenberg [1963] zu gewichten. Formel (29) wird also wie folgt erweitert.

(33) $$B_0(\vec{x}) = \int_{z}^{h_0} \frac{p_0(y) \cdot TM_o}{(y-z)^2 + \frac{R_o^2}{cf}} \mathrm{dy} + \sum_{i=1}^{n} \int_{y_{\min,i}}^{h_i} \frac{p_i(y) \cdot TM_n}{(y-z)^2 + d_i^2 + \frac{R_i^2}{cf}} \mathrm{dy}$$

TM_0 steht hierbei für den Lichttransmissionskoeffizienten von $Baum_0$, TM_j für den $Nachbarn_j$ von $Baum_0$. Pretzsch [2001] gibt Lichttransmissionskoeffizienten von 1.0 für Tanne (Abies alba Mill.); 0,8 für Fichte (Picea abies (L.) Karst) und 0,2 für Kiefer (Pinus silvestris L.) an.

3.7.7 Kalkulieren der Beschattung

3.7.7.1 Randeffekte

Randeffekte sind in dem vorliegenden Modell nur für die Berechnung der beschattenden Nadelbiomasse von Bedeutung. Zur Lösung dieser Randeffekte wird der Hauptbestand temporär 8-fach mit seinem gesamten Subgraphen kopiert und jeweils so verschoben, dass die Schattenbestände den Hauptbestand vollständig umgeben. Jede Kopie schließt sich hierbei mit seiner genau gegenüberliegenden Seite an den Hauptbestand an (Abbildung 25).

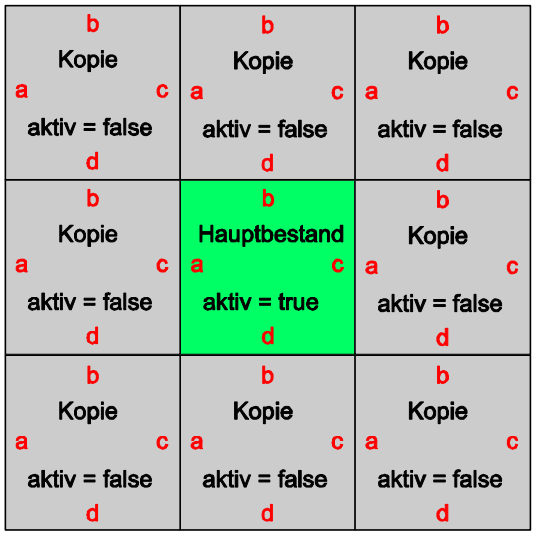


Abbildung 25: Temporäre Kopien (grau) des Hauptbestandes (grün) bilden einen Schattengürtel zur Lösung der Randproblematik. Die Bezeichnung der Seiten (a-d) zeigt das Anschlussmuster. Der boolsche Parameter aktiv wird in Ersetzungsregeln zur Identifizierung der Schattenkopien verwendet.

Nach der Berechnung der Beschattungswerte werden die Schattenbestände mit allen Tochterelementen wieder entfernt. Hierzu werden die Graphen der Schattenbestände gelöscht.

3.7.7.2 Beschattungswerte der Segmente

Die Beschattung wird für alle Segmente (*seg*) im Hauptbestand berechnet, für die gilt, dass die Nadelmasse des Segmentes > 0 ist. Die Nadelmasse ist den Segmenten während ihrer Initialisierung über die Betaverteilung zugewiesen worden und ist in kgC angeben (vergl. Kapitel 3.7.5.5). Eine Überprüfung der unteren und oberen Benadlungsgrenzen ist daher nicht mehr notwendig. Für jedes dieser Segmente (in Abbildung 23 angedeutet durch die grau gestrichelten Winkel) erfolgt eine Aufsummierung der gewichteten Nadelmassen aller Segmente – des Hauptbestandes und der temporären Kopien -, deren Basispunkt ($\overrightarrow{segbasis}_i$) innerhalb eines nach oben offenen Konus mit einem Öffnungswinkel α liegt.

Für jedes Segment gilt:

(34)
$$B(seg_0) = \sum_{i=1}^{n} \frac{M(seg_i) \cdot TM_{Baum(i)}}{Dist(\overrightarrow{segbasis}_0, \overrightarrow{segbasis}_i) + c_i} \quad \text{mit } c_i = \frac{R^2{}_{Baum(i)}}{cf_{Baum(i)}},$$

wobei Dist($\overrightarrow{segbasis}_0$, $\overrightarrow{segbasis}_i$) die Distanz zwischen den zwei Basispunkten ermittelt und c_i den spezifischen Korrekturterm des Baumelementes wiedergibt, dessen Tochterelement das Segment i ist. Mit $TM_{Baum(i)}$ ist der Lichttransmissionskoeffizient des Baumes angegeben, zu dessen Tochtergraphen das Segment-Objekt i gehört. Für Fichte wird ein Faktor von 0,8 und für Kiefer von 0,2 verwendet [Pretzsch 2001].

Die Beschattungswerte werden in den Eigenschaften externeBeschattung, selbstBeschattung und gesamtBeschattung in den Segment-Objekten gespeichert.

Die für die Beschattung durch die Nachbarbäume verwendete Query über den Graphen ist der rechenintensivste Teil der Simulation. Die im Bsp. 3.7-6 kursiv geschriebenen Bedingungen auf der linken Regelseite (Zeile 5) und die Query (Zeilen 9-11) stellen bereits eine Optimierung der Berechnung dar. Dennoch benötigt die Methode für einen kleinen Bestand mit 63 Bäumen mit 108 Segmenten ca. 80 Sekunden Rechenzeit auf einem aktuellen Windows XP-PC. Die Wahl der maximalen Segmentlänge (vergl. Kapitel 3.7.5.4) hat entscheidenden Einfluss auf die Berechnungsdauer der Beschattungswerte.

Bsp. 3.7-6: XL-Query zur Berechnung der beschattenden Biomasse durch externe Baumsegmente (Code mit eingefügten Zeilennummern).

```
1 private  void kalkuliereExterneBeschattung(float winkel)
2{
3  [
4       s:Segment -ancestor-> best:Bestand,
5        ( s.nadelMasseGesamt > 0.0f && best.aktiv == true) ::>
6  {
7       s.externeBeschattung = 0.0f;
8 // Ermitteln der beschattenden Segmente(a) für das aktuelle Segment (s)
9       for ((* b:Baum (-->)* a:Segment ,
10       ( s.getParent() != b && b.getLastChild() in cone(s, false, winkel)),
11        ( a.nadelMasseGesamt > 0.0f && a in cone(s, false, winkel)) *))

12      {
13           s[externeBeschattung] += ( a.nadelMasseGesamt
14              * b.lichtTransmissionsKoeffizient / (distanceSquared(a, s)
15              + b.kalkuliereNadelMasseVerteilungKorrekturTerm()));
16      };
17      if (s.externeBeschattung > 0.0f) s[color] = F_BESCHATTET;
18 }
19.]
20}
```

3.7.8 Kalkulieren der Nettophotosynthese

Die Nettophotosyntheseleistung eines Jahres errechnet sich aus der übers Jahr produzierten Menge von Assimilaten (Bruttophotosynthese) abzüglich der Erhaltungsatmung. Die Erhaltungsatmung wird als unabhängig von der Wachstumsrespiration und der Photosyntheseleistung angesehen. Die im Rahmen der Wachstumsrespiration enthaltenen Assimilate sind im Nettoassimilatespeicher also noch enthalten und werden erst im späteren Modellteil berücksichtigt. Die Dunkelatmung wird hingegen nicht weiter beachtet, da diese nur bei sehr kurzen Betrachtungszeiträumen eine Rolle spielt [vergl. Pretzsch 2001, Seite 182]. Sie ist bereits bei der Jahreskapazität implizit berücksichtigt.

Die Bruttophotosynthese und Nadelrespiration werden auf Basis der Baumsegment-Objekte berechnet und in den Eigenschaften assimilateSpeicher und nadelRespiration gespeichert. Die Erhaltungsrespiration der anderen Kompartimente wird für den Baum als ganzes im Baum-Objekt berechnet.

3.7.8.1 Kalkulieren der Bruttophotosynthese

Wie in Kapitel 3.7.6 eingeführt, wird die jährliche Produktionsrate p($\vec{x}$) (kgC kgC^{-1} j^{-1}) für 1-jährige Nadeln am Punkt $\vec{x}$ in Relation zur maximalen Produktionsrate p_0 bei unbeschatteten Verhältnissen ausgedrückt, wobei die Reduktion durch eine Funktion F in Abhängigkeit der gewichteten beschattenden Biomasse $B(\vec{x})$ am Punkt $\vec{x}$ bestimmt wird.

"*Es ist bekannt, dass Nadeln unterschiedlichen Alters nicht die gleiche Assimilationsrate besitzen. So assimilieren die 1-jährigen und die neuen Nadeln am stärksten, während ältere Nadeln stark in der Leistung abfallen*" [Sloboda & Pfreundt 1989]. Diese Eigenschaft zeigt sich bei verschiedenen Nadelbaumarten [vergl. Schulze et al. 1977, Woodman 1971, Künstle & Mitscherlich 1975]. Die Produktionsrate wird für die einzelnen Nadeljahrgänge daher mit einem Reduktionsfaktor μ_{jhg} in Bezug zur Leistung der 1-jährigen Nadeln korrigiert.

Um die Bruttophotosyntheseleistung eines Baumes zu berechnen ist es notwendig – getrennt für die einzelnen Nadeljahrgänge – das Produkt der Produktionsrate $p_{jhg}(\vec{x})$ und der Nadeldichte $\rho_{jhg}(\vec{x})$ über die Baumkrone zu integrieren und die Nadeljahrgangsassimilate zu addieren.

In dem vorliegenden Modell wird die Menge der durch Photosynthese produzierten Assimilate auf Basis der Segment-Objekte als Intervall interpoliert. Die Nadelmasse für die Nadeljahrgänge werden gemäß Formel (15) ermittelt. Die beschattende Biomasse fließt gemäß Formel (34) mit in die Berechnung ein. Unter Annahme einer in Formel (32) aufgezeigten Funktion F erfolgt die Berechnung der Bruttophotosynthese für einen Baum in kgC mit

(35) $$\mathrm{p}(Baum) = p_0 \cdot \sum_{Seg_I}^{n} F(B(Seg_i)) \sum_{jhg=0}^{n} M_{jhg} \cdot \mu_{jhg}$$

Für die Faktoren μ_{jhg} der Nadeljahrgänge geben Sloboda & Pfreundt für Fichte folgende Werte an, die ebenfalls für die Kiefer genommen werden:

0-1-jhg	*2-jhg*	*3-jhg*	*4-jhg*	*5-jhg*	*6-jhg*
2.0	0.65	0.6	0.55	0.5	0.45

Tabelle 8: Werte zur Reduktion der photosynthetischen Kapazität mit zunehmendem Nadelalter. Bezugsgröße ist die Kapazität 1-jähriger Nadeln. Nadeljahrgang 1 ist mit den neu gebildeten Nadeln des aktuellen Jahres zusammengefasst [Sloboda & Pfreundt 1989].

Für die maximale Photosynthesekapazität bei unbeschatteten Bedingungen wurde ein Wert von 2.8 kgC·kgC^{-1}·j^{-1} für Fichte [Sloboda & Pfreundt 1989] und 4.6 kgC·kgC^{-1}·j^{-1} für Kiefer angesetzt [Perttunen et al. 1996].

Wesentlich für die Photosyntheseberechnung ist aber die verwendete Funktion F für den Zusammenhang von beschattender Biomasse und Bruttoassimilation. Hierzu gibt es verschiedene Ansätze. Mäkelä [1979; zitiert von Kellomäki et al. 1980] benutzen folgende Funktion für Kiefer:

(36) $$F(M(\vec{x})) = \frac{c}{1 + e^{-2 \cdot \frac{\log M(\vec{x}) - a}{b}}} + 1 - c \quad ; \text{mit } a = 2.3,\ b = -1.2 \text{ und } c = 0.8.$$

Dieser Ansatz ist aber mit der in Formel (34) berechneten beschattenden Biomasse nicht ohne Weiteres zu verwenden. Zum Einen wird als Bezugsgröße $M(\vec{x})$ die gesamte beschattenden Biomasse oberhalb des Punktes $\vec{x}$ in kg/ha verwendet. Es wird kein Lichtkonus verwendet. Zum Anderen wird die Biomasse bezüglich ihrer Beschattungswirkung nicht mit der Distanz gewichtet. Die horizontale Verteilung der Biomasse wird nicht berücksichtigt.

In ihrem Ansatz schlagen Sloboda & Pfreundt [1989] daher eine Funktion der Form

(37) $$F(B(\vec{x})) = MAX(1 - a \cdot B(\vec{x})^b; 0)$$

mit sinnvoll angesetzten Parametern für a und b vor. Sloboda und Pfreundt nutzen diese Funktion für Fichten-Reinbestände. Konkrete Werte werden in ihrer Arbeit nicht genannt. Als initiale Werte werden in dem vorliegenden Modell 0,1 für a und 0,5 für b genutzt.

Monsi & Saeki [1953] benutzen in ihrer Arbeit den Ansatz einer Beer-Lambert-Funktion mit dem Blattflächenindex als Eingangsgröße. Die Blattfläche und somit auch der Blattflächenindex werden im vorliegenden Modell nicht berechnet. Um den Ansatz der Beer-Lambert-Funktion dennoch zu integrieren, wird als Berechnungsalternative eine Funktion auf Basis der Biomasse (kgC) angeboten:

(38) $$F(B(\vec{x})) = a \cdot e^{-k \cdot B(\vec{x})}$$

Als initiale Werte werden in dem vorliegenden Modell 1,0 für a und 0,03 für k genutzt.

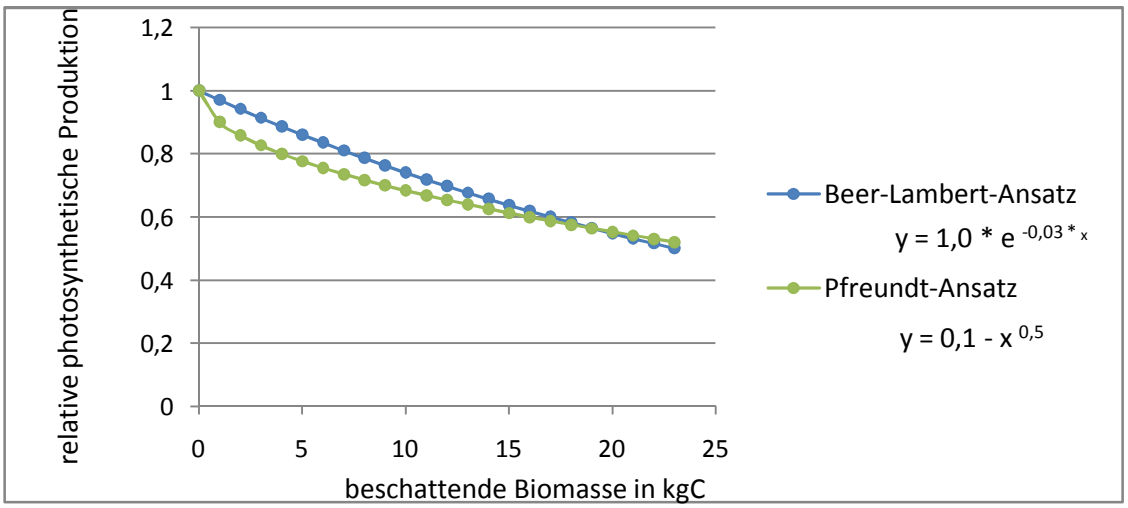


Abbildung 26: Verlauf der F-Funktionen des Modells mit der initialen Parametrisierung.

Abbildung 26 zeigt den Verlauf der Relation der Photosynthese am Punkt $\vec{x}$ zur maximalen photosynthetischen Kapazität bei unbeschatteten Verhältnissen, in Abhängigkeit zur beschattenden Biomasse $B(\vec{x})$ an Punkt $\vec{x}$. Zugrunde liegen die beiden Berechnungsansätze des Modells mit oben genannter Parametrisierung.

Keine der beiden Berechnungsfunktionen ist zurzeit empirisch abgesichert. Sie spiegeln vielmehr das grundlegende Denkmodell zum Wirkungsmechanismus von beschattender Biomasse auf die Photosyntheseleistung im Bestand wieder. Eine Anpassung der Funktionsverläufe mit Hilfe empirischer Daten oder die Integration von alternativen

Berechnungsmethoden aus der Literatur ist – im Rahmen der kritischen Auseinandersetzung mit dem Modell innerhalb des Lernprozesses – ein erreichbares Ziel. Der technische Aufbau des Modells erlaubt bereits jetzt baumweise eine Neuparametrisierung bzw. einen Methodenwechsel während der Simulation.

3.7.8.2 Kalkulieren der Respiration

Die Erhaltungsatmung wird für die 5 Kompartimente getrennt berechnet und anschließend auf Baumebene addiert. Für die Respiration liegen zum Teil keine vollständigen Parametersätze für die Fichte vor. In diesem Fall wird die Berechnung wie bei der Kiefer ausgeführt. Der Ansatz von Sloboda & Pfreundt [1989] für Äste, Stamm und Wurzeln der Fichte sah vor, mit einer Respirationsrate in Abhängigkeit vom maximalen Durchmesser und der Biomasse des jeweiligen Kompartiments zu arbeiten, wobei die maximalen Durchmesser der Kompartimente aus dem Brusthöhendurchmesser (d13) abgeleitet wurden. Leider konnte die angegebene Berechnungsmethode nicht reproduziert werden und führte zu Respirationswerten, welche die gesamte Photosyntheseleistung des Baumes um ein Mehrfaches überstieg. Für diese 4 Kompartimente wurden daher aus der Literatur andere Ansätze gewählt, mit entsprechenden Einschränkungen in der Parametrisierung.

Nadelrespiration

Die Respirationswerte für die Nadeln werden für alle Segment-Objekte einzeln berechnet und in der Segmenteigenschaft "nadelRespiration" gespeichert. Im späteren Verlauf des Simulationsdurchlaufes wird diese Eigenschaft zur Ermittlung des neuen Kronenansatzpunktes benötigt.

Die Nadelrespiration wird für alle Nadeljahrgänge mit einem gemeinsamen Faktor proportional zur aktuellen Nadelmasse berechnet (kgC kgC^{-1} j^{-1}). Für die Kiefer geben Perttunen et al. [1998] 0,2 kgC kgC^{-1} pro Jahr an. Für die Fichte nennen Sloboda und Pfreundt [1989] Werte von 0,5-1 kgC kgC^{-1} pro Jahr. Das vorliegende Modell wird mit 0,2 kgC $kgC^{-1}j^{-1}$ für Kiefer und mit 0,5 kgC $kgC^{-1}j^{-1}$ für Fichte parametrisiert.

Wurzelrespiration

Für die Wurzelrespiration geben Perttunen et al. [1996] eine massenbezogene Respirationsrate von 0,24 kgC kgC^{-1} j^{-1} für die Kiefer an. Dieselben Werte benutzen Lo et al. [2001] auch für Pinus banksiana Lamb. Zwischen Fein und Grobwurzeln wird nicht unterschieden. Da eine entsprechende Parametrisierung für Fichte nicht vorliegt, wird diese Respirationsrate im Modell für die Kompartimente der Fein- und Grobwurzeln für alle Baumarten angewandt.

Respiration Stamm

Für die Angabe von Respirationsraten der Holzteile werden in der Literatur sowohl das Splintholzvolumen (µmolC m^{-3} s^{-1}) [Bosc et al. 2003, Lo et al. 2001, Lavigne 1996, Lavigne et al. 1996], die Stammoberfläche (µmol m^{-2} s^{-1}) [Zah et al. 1994, Wieser & Bahn 2004, Künstle & Mitscherlich 1976], die Biomasse (µmolC $kgTg^{-1}$ s^{-1}) [McDowell et al. 2000] als auch der Stickstoffgehalt des Phloem und des Splintholzes (µmolCO_2 $(molN)^{-1}$ s^{-1}) [Bosc et al. 2003] als Bezugsgröße verwendet. Wenngleich die Verwendung der Biomasse als Bezugsgröße für die Äste üblicher ist als für den Stamm, hat Lavigne [1996] gezeigt, dass das Splintholzvolumen, die Oberfläche und der Stickstoff-

gehalt der lebenden Biomasse für die Stammrespiration gleichermaßen verwendbar sind. Im vorliegenden Modell wird die Oberfläche als Bezugsgröße verwendet, da Splintholzvolumen und Stickstoffgehalte im Modell nicht enthalten sind.

Respirationsraten sind temperaturabhänging. Sie werden daher in der Regel in einer normalisierten Form angegeben. Dies erfolgt über die Respirationsrate R_{15} bei 15° Celsius und einen Temperaturkoeffizienten Q_{10}, welcher die Veränderung der Respirationsrate bei einer Erhöhung der Stammtemperatur von 10°Celsius beschreibt. Die Berechnung der Respirationsrate R bei einer gegebenen Stammtemperatur T erfolgt über Lavigne [1996]

(39) $$R = R_{15} Q_{10}^{(T-15)/10}$$

Zah et al. [1994] geben bei Kiefer (Pinus silvestris L.) für die Erhaltungsrespiration des Stammes – bezogen auf dessen Oberfläche – Respirationsraten (R_{15}) von 0,72 – 0,87 µmol m^{-2} s^{-1} bei 15°C mit einem Q_{10}-Koeffizienten von 1,99 – 2,09 an.

Es bleibt die Aufgabe, die Respirationsraten auf das Jahr hochzurechnen. Hierzu wird zuerst der Bezug zwischen Lufttemperatur der Umgebung und der Stammtemperatur hergestellt. Zah et al. [1994] stellen hierfür eine enge Relation fest, in der Form einer linearen Gleichung mit

(40) $$T_{stamm} = 0{,}56 + 0{,}92 \cdot T_{luft} \qquad \text{mit } r^2 = 0.98$$

Im Weiteren werden nun die durchschnittlichen Tagesrespirationswerte für das Jahr berechnet und aufsummiert. Hierzu werden Klimadaten der Wetterstation 10315 Münster/Osnabrück des Deutschen Wetterdienstes [DWD 2006 a & b] für das Jahr 2005 verwendet. Gemäß der Formeln (39) und (40) errechnet sich die Jahresrespirationsrate durch

(41) $$R = \sum_{d=1}^{365} R_{15} Q_{10}^{((0{,}56+0{,}92\cdot\overline{T_d})-15)/10)} \cdot 0{,}0010368 \qquad \text{in (kgC m}^{-2}\text{ j}^{-1}\text{).}$$

wobei $\overline{Td}$ die Tagesmitteltemperatur in °C, und 0,0010368 einen Umrechnungsfaktor µmol s^{-1} auf kg j^{-1} bei einem molaren Gewicht von 12 g für Kohlenstoff angibt.

Mit dem Durchschnitt der angegebene Werte R_{15} von 0,795 und Q_{10} von 2,04 ergibt sich eine Jahresrespirationsrate von 0,24211 (kgC m^{-2} j^{-1}) mit Bezug auf die Stammoberfläche bei Kiefer. Da Stockfors und Linder [1998] ähnliche Q_{10}-Werte für Fichte angeben, werden dieselben Respirationsraten auch für Fichte genutzt.

Astrespiration

Perttunen et al. [1996] geben für ihr Modell für die Kiefer eine Respirationsrate von 0,024 kgC kgC^{-1} j^{-1} für Splintholz an. Vereinfachend davon ausgehend, dass die Äste durchschnittlich zu mindestens 50% aus lebender Biomasse bestehen, wird diese Respirationsrate (verdoppelt) für das Astkompartiment verwendet. Auch kommt hier zum Tragen, dass die Astrespiration deutlich geringer ausfällt als die des Nadel- und des Wurzelkompartimentes und dass Änderungen in der Berechnung die Simulationsergebnisse nur geringfügig verändern. Dennoch ist dieser Modellteil noch vorläufig und sollte in zukünftigen Versionen überarbeitet werden.

3.7.9 Verteilen der Assimilate auf die Kompartimente

Über die Verteilung der Assimilate auf die 5 Kompartimente wird das Baumwachstum determiniert. Die Allokationsfunktionen sind für Änderungen sehr sensibel und haben erhebliche Auswirkungen auf die Simulationsergebnisse.

Die im Folgenden verwendeten Methoden und Parameter sind experimenteller Natur und dienen der Verdeutlichung und der kritischen Analyse der Hypothesen zum Baumwachstum, die in der Literatur teilweise kontrovers diskutiert werden. Es wird zur Zeit derselbe initiale Parametersatz für Kiefer und Fichte verwandt.

Im folgenden Modell erfolgt die Aufteilung in 6 Schritten: Nach jedem der 6 Berechnungsschritte (i) wird die dabei verwandte Menge ($Verbrauch_i$) an Assimilaten (kgC) vom der zu Beginn des Rechenschrittes vorhandenen Assimilatmenge abgezogen und bildet damit den verfügbaren Speicher für den nächsten Schritt ($i+1$):

(42) $$Nettoassimilate_j(i+1) = Nettoassimilate_j(i) - Verbrauch_j(i)$$

1. Schritt: Wachstumsrepiration

Die Wachstumsrespiration ist definiert als die für das Wachstum zur Energiegewinnung verbrauchten Assimilate. Sloboda & Pfreundt haben hierfür 30 % des Nettoassimilatespeichers angesetzt. Diese wird vor der Aufteilung auf die 5 Kompartimente vom Nettoassimilatespeicher abgezogen, d. h sie wird gleichermaßen auf alle Kompartimente angewandt. Zu diskutieren ist, ob durch eine alters- und kompartimentsbezogene Anpassung der Verbrauchsrate eine größere Realitätsnähe des Modells erreicht werden kann. So haben Zah et al. [1994] für den Stamm eine Wachstumsrespirationsrate im Bezug zur Erhaltungsrespiration gesetzt und hierfür eine Aufteilung von 20 % zu 80 % der Gesamtrespiration angesetzt. Stockfors [1998] beobachtet einen Anstieg der Wachstumsrespiration des Stammes mit zunehmendem Alter, auf mehr als das Doppelte. Auch Cannell & Thornley weisen darauf hin, dass das "*growth-maintenance paradigm may be acceptable and useful for some purposes, but it should be realized that there is no rigorous division between growth and maintenance energy-requiring processes*" [Cannell & Thornley 2000, S. 49]. Auf der anderen Seite gehen Gielen et al. davon aus, dass "*Woody-tissue respiration (R_t) is generally separated into growth respiration (R_g) and maintenance respiration (R_m), in which R_g is related to the synthesis of new tissues, and R_m to the maintenance of existing cells (Penning de Vries 1975, Amthor 1989). Growth respiration per unit tissue produced is temperature-independent (Penning de Vries et al. 1974), while R_m changes with temperature (Penning de Vries 1975)*" [Gielen et al. 2003, S. 500].

2. Schritt: Feinwurzelwachstum

Soboda & Pfreundt schlagen vor, das Feinwurzelwachstum an die Bruttoassimilateleistung zu koppeln. Sie nehmen eine stark vereinfachte Beziehung zwischen Lichtgenuss, Photosyntheseleistung, Nährstoff- und Wasserbedarf und dem daraus resultierenden Feinwurzelwachstum – abhängig von der Güte des Standorts – an. Für das vorliegende Modell wird dieser Mechanismus übernommen, und mit einer starken Erneuerungsrate der Feinwurzel gekoppelt. Die Masse der Feinwurzeln für das Jahr $j+1$ entspricht daher

(43) $$FWMasse_{j+1} = FWMasse_j \cdot FWÜberlebensrate + FWAllokationsrate \cdot Bruttophoto_j$$

mit Werten von 0,05 für die Feinwurzelallokationsrate und einer Überlebensrate von 66 %. Die Eigenschaft für die Feinwurzelmasse im Baum-Objekt wird direkt aktualisiert.

3. Schritt: Nadelwachstum

Für die Nadeln schlagen Sloboda und Pfreundt [1989] vor, das Wachstum an die relative Effizienz der Photosyntheseleistung zu koppeln. Dies folgt der Hypothese, dass ein Baum, der stark unter Beschattung gerät, mehr in das Wachstum von Nadeln investiert als ein Baum mit guter Strahlungsversorgung. Für das vorliegende Modell wurde dieser Mechanismus zur Berechnung der Nadelmasse im Jahr $j+1$ in folgender Form übernommen:

(44) $$NadelMasse_{j+1} = Nettoassimilate_j(3) \cdot MIN(maximalRate; \frac{minimalRate}{relativeProduktionseffizienz_j})$$

mit Werten von 0,3 als Minimalrate und 0,5 als Maximalrate zur Deckelung des Nadelwachstums und der relativen Produktionseffizienz als

(45) $$relativeProduktionseffizienz_j = \frac{Bruttophotosyntheseleistung_j}{Nadelmasse_j \cdot Produktionsrate_{unbeschattet}}$$

Die neue Nadelmasse und die errechnete Produktionseffizienz werden im Baum-Objekt als Eigenschaften gespeichert. Die Fortschreibung der Nadelmasse für das nächste Jahr erfolgt später im Simulationslauf (siehe Kapitel 3.7.11.1).

4. Schritt: Astwachstum

Das Astwachstum sehen Sloboda & Pfreundt [1989] analog zur neu erzeugten Nadelmasse. Wenngleich sie auch gegenteilige Untersuchungen an Kiefer von Kellomäki [1981] anführen, vermuten sie für ihr Modell eine Korrelation zwischen neu erzeugter Nadelmasse und dem Wachstum der Nadelmasse als Träger dieser Nadeln. Das vorliegende Modell implementiert die Annahme des Ursprungsmodells in der Form:

(46) $$AstMasse_{j+1} = AstMasse_j \cdot Astüberlebensrate + NadelMasse_{j+1} \cdot Astwachstumsrate$$

mit 0,1 für die Astwachstumsrate und 0,8 für die Überlebensrate.

5. Schritt: Grobwurzelwachstum

Für das Grobwurzelwachstum postulieren Sloboda und Pfreundt [1989] einen dem Nadelwachstum gegenteiligen Effekt. Je höher die Photosyntheseleistung, desto höher der Bedarf an Transportwegen für Wasser und Nährstoffe. Einem pragmatischen Ansatz folgend werden die rechnerisch noch im Nettoassimilatespeicher befindlichen Mengen zwischen den verbleibenden Kompartimenten Grobwurzeln und Stamm aufgeteilt. Der Effekt der Beschattung geht demzufolge über die Reduktion des Assimilatespeichers in die Schritte 3 und 4 mit ein, die in inverser Abhängigkeit zur Produktionseffizienz größer oder kleiner ausfällt.

Grundlage für die Aufteilung zwischen den beiden verbleibenden Kompartimenten ist das Aufteilungsverhältnis bei der Initialisierung der Bäume. Um eine feste Aufteilungsquote zu gewährleisten, müssen aber zuvor Absterbeprozesse bei den Grobwurzeln berücksichtigt werden. Hierzu wird eine Überlebensrate integriert. Die Menge der auf das Wurzelwachstum verwandten Assimilate berechnet sich demzufolge mit:

(47) $GWneu = (Nettoassim il._j(5) - GWMasse_j \cdot (1 - GWÜLrate)) \cdot GWAnteilsr ate + (1 - GWÜLrate)$

Für die Grobwurzelmasse des nächsten Jahres ergibt sich daraus:

(48) $$GWMasse_{j+1} = GWMasse_j \cdot (1 - GWÜLsrate) + GWneu_j$$

6. Schritt: Stammwachstum

Die übrigen Assimilate werden dem Stammkompartiment zugeschlagen. Sie bilden die Grundlage für das Höhenwachstum und das Dickenwachstum des Baumes (siehe Kapitel 3.7.10).

3.7.10 Berechnen des Stammwachstums

3.7.10.1 Höhenwachstum

Das Höhenwachstum eines Baumes mit Alter t im aktuellen Jahr auf das Alter $t+1$ im nächsten Jahr setzt sich aus drei Komponenten zusammen:

1. dem Zuwachs, der sich aus Anwendung der Höhenkurve für die Bestandesmittelhöhe für das Alter t und $t+1$ ergibt
2. eine rang- und standortspezifische Komponente, welche sich aus der aktuellen Höhe des Baumes i im Alter t im Vergleich zur Mittelhöhe der Höhenkurve im Alter t ergibt
3. einer Zufallskomponente.

Die gesamte Zuwachsgleichung setzt sich aus 3 Termen zusammen:

(49) $$\Delta h(t) = \Delta \bar{h}(t) + \Delta h_2(t) + \varepsilon(t)$$

wobei Term 2 und 3 sowohl positive wie negative Werte ergeben können.

Term 1:

Eine Höhenkurve wird in der Regel durch eine oder mehrere quadratische Gleichungen beschrieben, welche die Beziehung zwischen dem Alter des Baumes und der Mittelhöhe des Bestandes an einem gegebenen Standort wiedergeben.

Für das vorliegende Modell werden zwei Gleichungen mit folgender, von Sloboda und Pfreundt [1989] nach der Fichten-Ertragstafel von Assmann & Franz [1963] vorgeschlagenen Parametrisierung verwandt:

(50)
$$\bar{h}(t) = 0{,}68 + 0{,}12 \cdot t + 0{,}0087 \cdot t^2; t < 25$$
$$\bar{h}(t) = 0 - 5{,}12 + 0{,}636 \cdot t + 0{,}00265 \cdot t^2; t \geq 25$$

Der mittlere Höhenzuwachs nach Höhenkurve ergibt sich aus

(51) $$\Delta \bar{h}(t) = \bar{h}(t+1) - \bar{h}(t)$$

Term 2:

Der mittlere Zuwachs und die mittlere Höhe im Alter t wird nun um einen zweiten Term mit einer rang- und standortspezifischen Komponente erweitert. Je nach Bonität des Standortes und der individuellen Situation kann die tatsächliche Höhe eines Baumes i deutlich von der Mittelhöhe abweichen. Es hat sich aber gezeigt, dass der Rang eines Baumes, d. h. seine Position in einer Reihung nach der Baumhöhe, sich nur wenig verändert [Sloboda 1988]. Je nach Richtung der Abweichung zur Mittelhöhe nach Höhenkurve, wird der folgende Term dazu genutzt, den Höhenzuwachs zu vergrößern oder zu vermindern:

(52) $$\Delta h_2(t) = r_{H(t),\Delta h(t)} \cdot \frac{s\Delta h(t)}{\sigma h(t)} \cdot (h_i(t) - \bar{h}(t))$$

$h_i(t)$ gibt dabei die Höhe des Baumes i zum Zeitpunkt t, $\sigma h(t)$ die errechnete Standardabweichung der Baumhöhen des simulierten Bestandes und $s\Delta h(t)$ die geschätzte Standardabweichung der Höhenzuwächse zum Zeitpunkt (t) an. $r_{h(t),\,\Delta h(t)}$ definiert einen Korrelationskoeffizienten zwischen der Baumhöhe im Alter t und dem Höhenzuwachs von t auf $t+1$ und gewichtet somit die Bedeutung des zweiten Zuwachsterms. Wie von Sloboda und Pfreundt vorgeschlagen, werden im vorliegenden Modell die Werte von Kobayashi [Kobayashi 1981, zitiert von Sloboda und Pfreundt 1989] als Konstante über alle t angenommen. Dieser hatte für Lärche den Korrelationskoeffizienten $r_{h(t),\,\Delta h(t)}$ mit 0,3 und einen Variationskoeffizienten für die Zuwächse $s\Delta h(t) / \Delta h(t)$ mit 0,2 angegeben. $\Delta h(t)$ wird im Modell berechnet.

Term 3:

Der dritte Term der Zuwachsgleichung beinhaltet eine reine Zufallskomponente zur Vergrößerung bzw. zur Verminderung des Höhenzuwachses. $\varepsilon(t)$ wird als Zufallszahl im Wertebereich von $-s\Delta h(t)$ bis $s\Delta h(t)$ durch eine Java-Funktion generiert:

(53) $$\varepsilon(t) = random(-s\Delta h(t), s\Delta h(t))$$

Wurde der Höhenzuwachs ermittelt, so wird ein Segment mit der entsprechenden Länge erzeugt und an den bestehenden Graphen angehängt. Sollte der Zuwachs größer als die maximal erlaubte Länge der Segmente sein, werden mehrere gleichlange Segmente angehängt. Die Zahl der Segmente wird so gewählt, dass die resultierende Segmentlänge unter dem definierten Maximalwert liegt (vergleichbar dem in Kapitel 3.7.5.4 beschriebenen Vorgehen).

3.7.10.2 Volumen- und Durchmesserzuwachs

Durch die Formeln (6) & (7) (siehe Kapitel 3.7.5.3: Stammparameter) wird die Berechnung des Stammvolumens und der Stammmasse unter Zuhilfenahme der unechten Formzahl definiert. Der neue Durchmesser für das Alter $t+1$ wird nun durch Umkehrung der Berechnung auf Basis der neuen Baumhöhe und Stammmasse für $t+1$ vorgenommen, mit:

(54)
$$d13_i(t+1) = \sqrt{\frac{M_{stamm}(t+1)}{\frac{\pi}{4} \cdot h(t+1) \cdot f_{1,3} \cdot cQuote \cdot \rho_{Holz}}}$$

Die Eigenschaften der Baum-Objekte für Durchmesser, Volumen und Oberflächen des Stammes werden unter Verwendung der Formeln (54), (6) & (10) aktualisiert.

3.7.10.3 Kronenradius

Die Funktion zur Dynamisierung des Kronendurchmessers ist vorläufiger Natur. Der Kronenradius wird zurzeit einfach linear fortgeschrieben mit

(55)
$$Kronendurchmesser(t+1) = \frac{Kronendurchmesser(t)}{t} \cdot (t+1)$$

Eine deutlich bessere Bezugsgröße für den Kronenzuwachs wäre die Kronenmantelfläche [vergl. Pretzsch 2001].

3.7.10.4 Kronenansatzpunkt

Der Kronenansatzpunkt ist die untere Benadelungsgrenze der Krone und somit der untere Kronenpunkt a gemäß Formel (13) für die Beta-Verteilung der Nadeldichten. Da im vorliegenden Modell die Segmente die Träger der Nadelmassen sind, ist der Kronenansatzpunkt mit dem Basispunkt des untersten Segmentes mit Nadelmasse > 0 identisch. Bei der Fortschreibung der Nadelmassen vom Alter t auf $t+1$ wird angenommen, dass bei denjenigen Segmenten, deren Nadelrespiration die Bruttophotosyntheseleistung überschreitet, die Nadeln nicht lebensfähig sind und absterben. Der Kronenansatzpunkt wird zum ersten höher liegenden Segment verschoben, bei dem die Bilanz > 0 ist.

3.7.11 Vorbereiten des nächste Simulationslaufes

Zur Vorbereitung des nächsten Simulationsdurchlaufs sind zwei Arbeitsschritte notwendig, die der einfachen Handhabung wegen am Ende eines Simulationsdurchlaufes ausgeführt werden und nicht am Beginn des Folgenden. Dies muss bei der Analyse von Zustandswerten unbedingt beachtet werden.

3.7.11.1 Aktualisieren der Nadelmassen

Am Ende des Simulationslaufes werden die Nadelmassen des aktuellen Jahres auf die Simulation der nächsten Periode übertragen. Tabelle 9 zeigt die im Modell verwendeten Überlebensraten, d. h. den prozentualen Anteil der Nadeln einer Altersstufe, die am Ende des Jahres überleben und in die nächste Altersstufe wechseln. Bezugsgröße ist die jeweilige im Baum-Objekt angegebene Nadelmasse der einzelnen Jahrgänge der aktuellen Periode.

	Jahr 0	*Jahr 1*	*Jahr 2*	*Jahr 3*	*Jahr 4*	*Jahr 5*	*Jahr 6*
Kiefer	90%	78%	71%	60%	0%		
Fichte	100%	95%	90%	90%	90%	85%	0%

Tabelle 9: Die im Modell verwendeten prozentualen Anteile der Nadeln eines Nadeljahrgangs, die am Ende des Jahres überleben und in die nächste Altersstufe wechseln.

Über die Lebensdauer von Nadeln der einzelnen Baumarten gibt es in der Literatur verschiedenen Aussagen. Sloboda & Pfreundt [1989] verwenden maximal 6 Nadeljahrgänge für ihr Fichtenmodell. Muukkonen et al. [2004] geben bis 9 Nadeljahrgänge an. Die verwendeten Parameter für Fichte sind abgeleitet von Muukkonnen et al. mit einem maximalen Alter von 6 Jahren. Die Überlebensraten für Kiefer sind abgeleitet aus Perttunen et al. [1996]. Das maximale Alter wird dort mit 4 Jahren angegeben.

3.7.11.2 Reinitialisierung der Segment-Objekte

Im zweiten Schritt werden die Segmente mit den geänderten Daten für die Nadelmassen, die Baumhöhe und dem Kronenansatz neu initialisiert. Hierfür wird wieder die Methode initialisiereSegmenteFuerJahresDurchlauf() verwandt, die bereits bei der ersten Initialisierung der Bäume Anwendung fand (siehe Kapitel 3.7.5.5).

3.8 Kommunikation Forester – GroIMP–Modell

3.8.1 Stufe 1: Empfangen der Daten

Wie in Kapitel 3.1.1 beschrieben, sind sowohl das Programm GroIMP als auch das vorliegende Modell Teil eines umfangreicheren Systems von Software-, Daten- und Modellkomponenten. Als Protokoll für das Austauschen von Forester-Bestandesbeschreibungen zwischen dem Wachstumsgenerator auf der einen Seite und dem Elan-Sim-Server im Netzwerkbetrieb bzw. dem Virtual Forester-Client-Rechner im Einzelplatzbetrieb wurde HTTP festgelegt und ein Austauschverfahren definiert (vergl. Kapitel 3.4.5). Auf der Seite des Programms GroIMP gewährleistet ein extra zu diesem Zweck eingebauter HTTP-Server (siehe Kapitel 3.6.3) die Möglichkeit, Daten als HTML 4.01-multipart-Formdata zu empfangen und in XL geschriebene Modelle von außen zu starten. Bsp. 3.8-1 zeigt die beiden Methoden startup() und run(), welche in der Haupt-RGG-Datei des Modells definiert sein müssen, um das Modell durch den HTTP-Server zu starten. Ist der GroIMP-HTTP-Server gestartet und wird das Modell per HTTP über eine URL in der Form: http://servername:port/open?modelname.gs angesprochen, wird ein Arbeitsbereich (workbench) von GorIMP geöffnet, ein Textbuffer (buf) mit einer Antwort erzeugt und versendet, anschließend die init()-Methode des Modells aufgerufen und zum Abschluss der Arbeitsbereich wieder geschlossen.

Bsp. 3.8-1

```
import de.grogra.imp.net.*
…
protected void startup()
// Startmethode für den HTTP-Server-Betrieb.
{
   super.startup();
   HttpResponse resp = HttpResponse.get(workbench());
   if (resp != null)
   {
        runLater(resp);
   }
}

// Methode run wird von runLater aus startup aufgerufen.
// Findet nur Verwendung beim Start der Simulation über den HTTP-Server
protected void run (Object info) {
   HttpResponse resp = (HttpResponse) info;   //enthält die gesendeten Daten
   StringBuffer buf = new StringBuffer();     // Textbuffer anlegen
   buf.append("url=" + server + "/open?" + modelout + ";" + filename + "\n");
   buf.append("status=ok\n");
   resp.setContent("text/text", "UTF-8", buf.toString()); //Format festlegen
   resp.send(true);       //Antwort senden

   init();                //Ausführen des Modells
   closeWorkbench();      // Arbeitsbereich schließen
}
…
```

Das Objekt resp der Klasse de.grogra.imp.net.HttpResponse enthält dabei neben den Methoden setContent() und send() für die Antwort auch einen Verweis auf ein Objekt der Klasse de.grogra.http.Request, welches den gesamten gesendeten Inhalt der HTML-Seitenanfrage enthält. Die Klasse Request verfügt über entsprechende Methoden wie getContent() und getHeaderField(). Für die vollständige Dokumentation der Klassen wird auf die GroIMP-API-Dokumentation unter www.grogra.de [Kniemeyer et al. 2007] verwiesen.

3.8.2 Stufe 2: Forester-Description in XL/Java

Wenngleich mit Stufe 1 das technische Rahmengerüst für das Empfangen und Versenden von Requestdaten und damit auch von Forester-Bestandesbeschreibungen gewährleistet ist, so muss die Bestandesbeschreibung noch in die jeweilige Modellstruktur überführt werden. Da der Vorgang des Auslesens der Bestandsbeschreibung und der Kommunikation mit dem Elan-Sim-Server bzw. der Nutzerin im engeren Sinne nicht Bestandteil des Modells ist und für unterschiedliche, mit GroIMP simulierte Modelle gleich verläuft, wurden 5 Java-Hilfsklassen entwickelt, die eine Forester-Bestandesbeschreibung in ein genormtes Forester-Description-Objekt überführen. Auf dieses Objekt kann dann aus dem Modell heraus beliebig zugegriffen werden. Entsprechende Zugriffsmethoden sind Teil der Objektklasse.

Im Folgenden werden die 5 Klassen in ihrer Funktion und mit ihren öffentlichen Datenfeldern und Methoden kurz beschrieben. Der Source-Code wird im Anhang in den Kapitel 10.4, 0 und 0 aufgeführt.

package forester.http.formdata

In dem Paket sind die beiden Klassen HttpMultipartFormData und HttpFormField zusammengefaßt.

HttpMultipartFormData (Bsp. 3.8-2) analysiert und zerlegt einen im HTML 4.01 multipart Formdata-Format aufgebauten String und extrahiert die Teilbereiche der einzelnen Formularfelder. Aufrufparameter des Konstruktors ist der Inhalt der Seitenanfrage als String und der boundary String [vergl. Nebel & Masinter 1995. Masinter 1988].

Für jedes gefundene HTML-Formularfeld wird ein httpFormField-Objekt angelegt und der von der boundary definierte Teilstring des Feldes übergeben. Die Referenzen auf diese Objekte sind in einem Eigenschaften-Array fields[] gespeichert. Weiterhin werden Methoden bereitgestellt, um auf die Eigenschaften eines Formularfeldes zuzugreifen, welches durch den Formularfeldnamen (key) identifiziert werden kann.

Abweichend von der theoretischen Definition des HTML 4.01-multipart-Formdata-Formates [Nebel & Masinter 1995] ist die gegenwärtige Version der Klasse auf maximal 1 Fileupload beschränkt.

Bsp. 3.8-2: Klasse HttpMultipartFormData

```
public class HttpMultipartFormData {
// public Eigenschaften
        public HttpFormField[] fields;

// public Methoden
        public HttpMultipartFormData(String content, String boundary) {}
        public int getNumberOfField() {}
        public String getFieldContent(String key) {}
        public String getFieldContentType(String key) {}
```

```
        public String getFieldContentDisposition(String key) {}
        public String getFieldType(String key) {}
}
```

HttpFormField (Bsp. 3.8-3) enthält die extrahierten Informationen für ein einzelnes HTML-Formularfeld. Für den Zugriff auf die Eigenschaften des Objektes stehen Methoden bereit:

Bsp. 3.8-3: Klasse HttpFormField

```
public class HttpFormField {
//public Methoden
        public String getContentDisposition(){}
        public String getName(){}
        public String getFileName(){}
        public String getType(){}
        public String getContentType(){}
        public String getFieldContent(){}
}
```

package forester.description

In dem Paket sind die beiden Klassen ForesterDescription und ForesterTree zusammengefasst.

Die Klasse ForesterDescription (Bsp. 3.8-4) stellt die XL/Java interne Entsprechung einer Forester-Bestandesbeschreibung dar. Bei der Instanziierung eines Objektes wird diese Bestandesbeschreibung analysiert und die relevanten Forester-Elemente werden für die spätere Nutzung im Modell extrahiert. Aufrufparameter sind die Forester-Bestandesbeschreibung als String und der Dateiname als String. In der gegenwärtigen Version werden folgende Forester-Elemente (vergl. 3.2.2) extrahiert:

a. Für jedes in der Bestandesbeschreibung enthaltenen TREE-Element (vergl. Kapitel 3.2.4.1) wird ein Objekt der Klasse ForesterTree erzeugt. Die Referenzen auf diese Objekte sind in einer ArrayList foresterTrees gespeichert.
b. Der übrige Teil der VRML-Beschreibung wird im Objekt zwischengespeichert.

Auf die extrahierten Elemente kann aus dem jeweiligen Modell heraus jederzeit direkt über das ForesterDescription-Objekt zugegriffen werden. Die Liste der extrahierten Elemente kann in zukünftigen Versionen einfach erweitert werden. Methoden für den Zugriff auf die Eigenschaften des Objektes werden bereitgestellt.

Ebenso existieren Methoden zur Rückkonversion eines ggf. durch das Modell aktualisierten Objektes in das Format einer Forester-Bestandesbeschreibung und zum Speichern als Datei.

Bsp. 3.8-4: Klasse ForesterDescription

```
public class ForesterDescription {
/** Array der Baumobjekte im Description Objekt*/
        public ArrayList foresterTrees;
// public - Methoden
        public ForesterDescription(String inbuf, String filename) {}
        public int getNumberOfTrees(){}
        public String getFileName(){}
        public String getForesterDescription(){}
        public void speichereForesterDescription( String filename){}
}
```

Die Klasse ForesterTree ist die XL/Java-Entsprechung eines Tree-Elementes aus der Forester-Bestandesbeschreibung (vergl. Kapitel 3.2.4.1). Jede Instanz der Klasse enthält die in der Bestandesbeschreibung definierten Eigenschaften eines Baumes. Bsp. 3.8-5 zeigt die Eigenschaften der Objekte mit den Defaultwerten. Auf die Eigenschaften kann – mit Ausnahme der data[] und labels[] Felder – aus dem Modell heraus direkt zugegriffen werden.

Wie im Kapitel 3.2.4.1: Zusätzliche Informationen beschrieben, enthalten die labels- / data-Felder eine für jeden Baum individuelle Anzahl von Merkmalen. Für die beiden Arrays gilt, dass, wenn labels[i] den Schlüsselbegriff eines Merkmals enhält, data[i] die dazugehörigen Merkmalsausprägung beinhaltet. Mit den Methoden getData(String key) setData(String key, String value) kann das korrespondierende Data-Feld zu einem Schlüsselbegriff ausgelesen und verändert werden.

Bsp. 3.8-5: Klasse ForesterTree

```
public class ForesterTree {

// public Eigenschaften
   public int id = 0;
   public int status = 1;
   public float[] position = {0,0,0};
   public float[] stemSize = {0,1,0};
   public float[] crownPosition = {0,1,0};
   public float[] crownSize = {1,1,1};
   public float[] crownValues = {1,1,1,1,1,1};

   public String species;
   public int numberOfData = 0;
   private String data[];
   public int numberOfLabels = 0;
   private String labels[];
   public String stemTexture;
   public String cownTexture;
   public int taperSystem;

// public Methoden
   public ForesterTree(String inbuf) {}
   public String getTreeString(){}     // Forester Tree-Element String
   public String getDataValue(String key){}
   public int setDataValue(String key, String value){}
   public void printLabels(){} //Servicefunktion. Ausgabe auf Konsole
   public void printData(){}   //Servicefunktion. Ausgabe auf Konsole
}
```

package forester.groimp

Das Paket enthält die Klasse ForesterUtils. Die Klasse beinhaltet statische Methoden, die keinen Einfluss auf die Simulation haben, im Modell aber zur Erzeugung von User-dialogen, dem Auslesen von GroIMP-Voreinstellungen etc. (siehe Anhang 10.6.1.1) verwendet werden. Diese Methodensammlung kann nach Bedarf erweitert werden.

3.8.3 Integration in das Modell

Aufgrund der unbestimmten Zeit, die – je nach Art und Komplexitätsstufe des Modells – eine Wachstumssimulation dauert, wurde das in Kapitel 3.4.5 definierte Austauschverfahren zwischen Elan-Sim-Server und dem Wachstumssimulator zweiphasig angelegt.

Phase 1 empfängt die Bestandesbeschreibung und liest sie aus, antwortet mit einer URL, mit der das Simulationsergebnis angerufen werden kann, und startet die Simulation. Abschluss der Phase 1 ist das Speichern der aus der Simulation resultierenden Bestandesbeschreibung unter der angegeben URL.

Die Umsetzung der Phase 1 erfolgt durch eine Anpassung der Methode run() in wald.rgg. Folgendes Beispiel zeigt die wesentlichen Teile der Methode:

Bsp. 3.8-6: run-Methode im Modell

```
protected void run (Object info)
{
…
// Auslesen der Formulardaten
   HttpMultipartFormData n1 = httpstartup.verarbeiteHttpInput(info);
// Auslesen der Foresterdescription
   foresterDescriptionIn = httpstartup.readForesterDescriptionFormfield(n1);
// Erzeugen und Senden der Antwort
   String filename = httpstartup.createAnswer(info,n1,foresterDescriptionIn);
   init();
   if(foresterDescriptionIn != null){
//Modellspezifischer Teil Anfang
         ersterSchritt();
         for (apply (simulationsPeriode)) kompletterJahresDurchlauf();
         abspeichern(foresterDescriptionIn , ( modelOutputDir + filename) );
//Modellspezifischer Teil Ende
   }
   closeWorkbench(); // schließt den Arbeitsbereich
}
```

Die Bearbeitung der HTTP-Seitenanfrage und der entsprechenden Antwort wurde zur besseren Übersichtlichkeit und Wiederverwendbarkeit in verschiedenen Methoden in der RGG-Datei httpstartup.rgg (siehe Anhang 10.2.5) gebündelt.

Der im Beispiel Bsp. 3.8-6: run-Methode im ModellBsp. 3.8-6 als modellspezifischer Teil markierte Block umfasst die Steuerung der Simulation. Bei Verwendung eines anderen Modells auf Basis einer Forester-Bestandesbeschreibung muss nur dieser Teil der run()-Methode angepasst werden

Phase 2 wird durch eine Seitenanforderung an die in Phase 1 zurückgeantworteten URL ausgelöst und umfasst die Auslieferung der fertigen Bestandesbeschreibung mit den Simulationsergebnissen bzw. der entsprechenden Statusmeldung, falls die Simulation noch nicht abgeschlossen ist.

Für diesen Zweck wurde das GroIMP-Projekt deliverscene.gsz (Anhang 10.3) erstellt. Die enthaltene RGG-Datei dient nur dem Zweck, eine – der Definition in Kapitel 3.4.5 für Phase 2 konforme – Statusmeldung über den Simulationsverlauf zu geben und ggf. die Bestandesbeschreibung mit dem Simulationsergebnis auszuliefern. Eine Anpassung für andere in Phase 1 verwendete Modelle ist nicht notwendig.

3.8.4 Kommunikation mit anderen Wachstumssimulatoren

Eine vollständige Integration eines anderen Wachstumssimulators in das vorliegende Projekt ist bislang noch nicht erfolgt. Grundsätzlich ist es nur notwendig, in die jeweiligen Programme eine Schnittstelle zur Kommunikation mit dem Elan-Sim-Server einzubauen. Für SIBYLA (siehe Kapitel 2.2.3) und BWIN (siehe Kapitel 2.2.4) bieten sich Alternativen.

3.8.4.1 Alternative Anbindung an SIBYLA

Für SIBYLA (siehe Kapitel 2.2.3) wurde eine Schnittstelle über einen Zugriff auf die Datenbank des Programms realisiert. Hierzu wurden in Cold Fusion Markup Language (CFML) Skripte entwickelt, die den Import/Export von Baumdaten aus der SIBYLA-Datenbank winnefeld.db erlauben. Die Skripte werden vom Cold Fusion-Applikations-Server Version 7-Standard unter Ansprache durch einen Apache 1.3.26 Web-Server ausgeführt. Die Datenbank wurde als Microsoft Access-Datenquelle im Cold Fusion-Server registriert.

Die Datenbank winnefeld.db beinhaltet durch SIBYLA generierte einzelbaumorientierte Strukturdaten der Bestände des Forstamtes Winnefeld im Solling. Durch den direkten Zugriff auf die Datenbank-Tabelle Trees können die Baumdaten eines ausgewählten Bestandes ausgelesen und manipuliert werden. Durch Zugriff auf die Tabelle Terrain können Höhendaten für den Untergrund ausgelesen werden.

Ein Skript für die Konvertierung zwischen der SIBYLA-Datentabelle und dem Forester-Bestandesbeschreibungs-Format wurde realisiert. Die Ansprache per HTTP ist durch die Anbindung über den Apache/Cold Fusion-Server realisiert.

Bislang nicht realisiert ist das Starten und Steuern der SIBYLA-Prognoserechnung über HTTP von außen. Eine mögliche Lösung kann hier die Einbindung einer Batchversion von SIBYLA auf dem Server sein.

3.8.4.2 Alternative Anbindung an BWIN

Bereits für die BWINPro Version 6.x hat Jürgen Nagel – der Autor des Programms – eine Exportschnittstelle für Forester-Bestandesbeschreibungen realisiert. Auf Basis der neuen Entwicklungen der BWIN Programmfamilie (vergl. Kapitel 2.2.4) ist aber eine andere, wesentlich besser integrierte Lösung möglich. Basis der neuen Programmversion BWINPro 7.1 [vergl. Nagel et al. 2006] ist die Open Source Java-Bibliothek TreeGross (Tree Growth Open Source Software) [siehe Nagel 2002 & 2003]. Die Bibliothek umfasst alle wesentlichen Wachstumsfunktionen zur direkten Einbindung in andere Programme. Damit eröffnet sich die Möglichkeit, die Wachstumsfunktionen von Nagel et al. in XL/Java-Modelle in GroIMP zu integrieren und auf diesem Weg effizient die Skalenebenen zu integrieren.

3.9 Exemplarische Anwendung des Systems

Zur Verdeutlichung des Zusammenwirkens der oben genannten Komponenten soll im Folgenden der Ablauf einer exemplarischen Anwendung des Systems kurz skizziert werden. Die verwendeten verkleinerten Bildschirmfotos illustrieren dabei – in der Art von Piktogrammen in einem Flußdiagramm – die einzelnen wesentlichen Arbeitsschritte. Eine größere Version der Bilder befindet sich im Anhang 10.7.

1. Vorbereitung: Generierung der Beschreibung

Erster Schritt ist das Erzeugen einer Forester-Bestandesbeschreibung anhand von Einzelbaumdaten aus der zentralen Datenbank von SIBYLA. Die Einzelbaumdaten wurden durch den Strukturgenerator von SIBYLA auf Basis der Forsteinrichtungsdaten der Niedersächsischen Landesforstverwaltung erzeugt [Zindler 2005]. Die Generierung der Bestandesbeschreibung (s. Kapitel 3.2) erfolgt über eine, für das vorliegende System entwickelte, Schnittstelle zu SIBYLA (s. Kapitel 3.8.4.1).

Verwendet wird der Fichtenreinbestand in Abteilung 284, Unterabteilung A des Forstamtes Winnefeld, mit der ID 0284010000 und einem Alter von 69 Jahren. Die Anzahl der Bäume beträgt 169 auf 50x50 Meter Fläche. Die BHDs variieren von 0,08 Meter bis 0,44 Meter, die Höhen von 10,8 Meter bis 33,8 Meter Kronen- und Stammschäden – je 10% der Bäume – werden per Zufallsgenerator auf die Bäume verteilt und mit den Schlüsselbegriffen für den Quality Identifier angegeben (s. Kapitel 3.5). Im Folgenden wird Baum 65 detailiert betrachtet und hierfür mit Status 2 (rot markiert) in der Bestandsbeschreibung gesetzt.

In der Virtual Forester Client (VRC) (s. Kapitel 3.3) Ansicht erscheint der Bestand mit den angegeben Standardtexturen wie folgt:

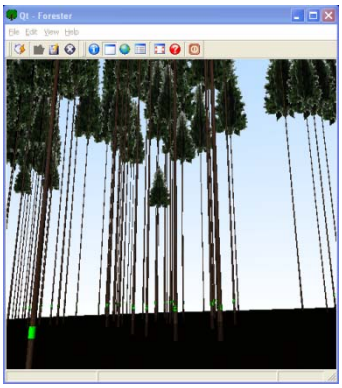


Abbildung 27: Standardansicht Virtual Forester Client.

2. Vorbereitung: Quality Identifier

Durch Anwenden des Quality Identifiers (s. Kapitel 3.5) wird die Bestandesbeschreibung umgeschrieben und für die Darstellung der Qualitätskriterien angepasst. Das folgende Bild (Abbildung 28) zeigt einen Baum mit gebrochener Krone. Der Vergleich mit Abbildung 27 zeigt den Unterschied in der Darstellung.

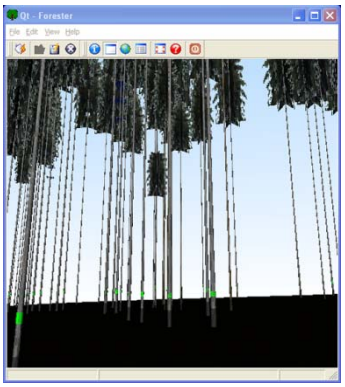
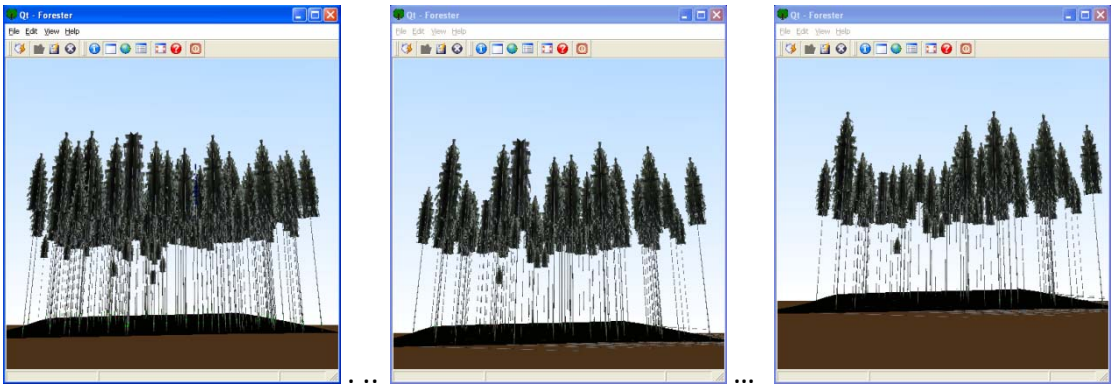


Abbildung 28: Baum (Bildmitte, unterständig) mit gebrochener Krone im VRC.

3. Durchforstung im Virtual Forester Client

Im vorliegenden virtuellen Bestand werden nun Durchforstungen im Virtual Forester Client vorgenommen. 3 Varianten werden verglichen (Abbildung 29): Variante A stellt die Null-Variante ohne Durchforstung mit 169 Bäumen dar. In der Variante B wird die Zahl der Bäume auf 93 gesenkt. Die Entnahme erfolgt über die gesamte Fläche verteilt. In Variante C wird die Zahl der Bäume auf 82 gesenkt. Ausgehend von Variante B werden in C gezielt die größten Bedränger von Baum 65 eliminiert. Baum 65 (Abbildung 30) hat eine Höhe von 27,02 Meter und liegt somit bei 71,3% der maximalen Baumhöhe des Bestandes. Die Entwicklung dieses Baumes in den 3 Varianten wird hier genutzt, um die Unterschiede in der Simulation zu illustrieren.



A **B** **C**

Abbildung 29: 3 Durchforstungsvarianten des Fichtenbestandes (Abteilung 284A) im Virtual Forester Client. (A linke Spalte, B Mittelspalte, C rechte Spalte).

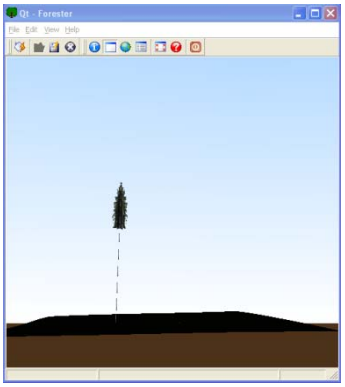
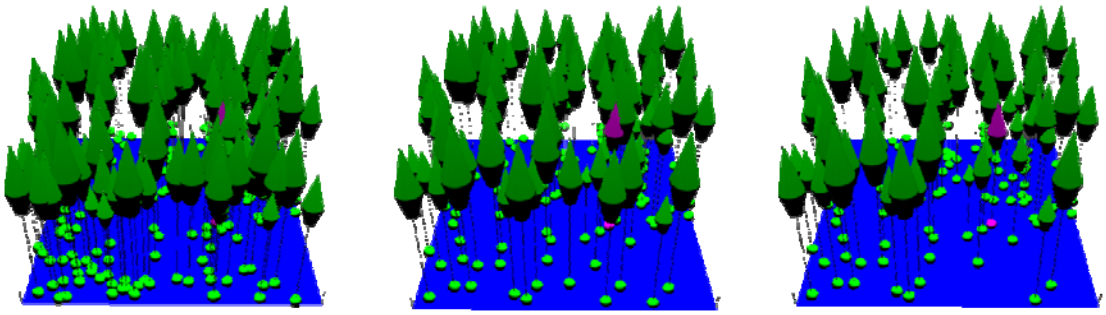


Abbildung 30: Baum 65 vor der Simulation in der Einzeldarstellung. (alle Varianten).

4. Versenden der Bestandesbeschreibungen an GroIMP

Im vierten Schritt werden die 3 Bestandesbeschreibungen per HTTP-POST an den Wachstumssimulator GroIMP (s. Kapitel 3.6) zur Prognoseberechnung geschickt (s. Kapitel 3.8). Das Verschicken der Daten und die Simulation der Varianten erfolgen in voneinander unabhängigen Prozessen. Zur besseren Übersicht werden die 3 Varianten im Folgenden nebeneinander dargestellt.

Abbildung 31 zeigt die räumliche Ausgangssituation vor der Simulation. Zur Erstellung der Bildschirmfotos wurde die Simulation manuell unterbrochen. Die Darstellung der Kronenform folgt dem Ansatz von Pretzsch [Pretzsch 2001 S. 205]. Der gesondert zu betrachtende Baum 65 wurde über die XL-Konsole manuell eingefärbt.

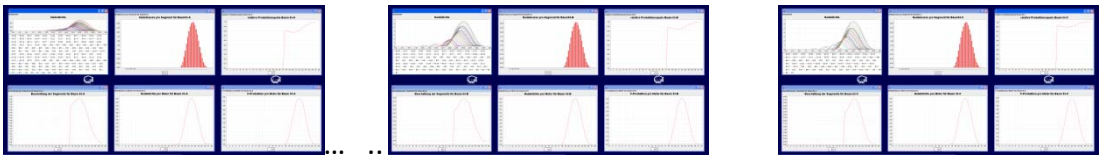


A B C

Abbildung 31: Räumliche Ausgangssituation der 3 Durchforstungsvarianten vor der Prognoseberechnung in GroIMP. Zur besseren Übersicht ist Baum 65 lila eingefärbt. Die Bilder wurden erstellt mit der Snapshotfunktion von GroIMP im png-Format [W3C:png 2003].

4.a. Zwischenschritt Analyse

Mit Hilfe der XL-Konsole und einiger im Modell enthaltener Service-Methoden (vergl. ausgabe.rgg im Anhang 10.2.6) können – bei manueller Unterbrechung des Simulationslaufes – Charts (Abbildung 32) einiger relevanter Parameter (z. B. Nadelmassen, Beschattung, Assimilatproduktion etc) der Segmente des Baumes 65 erzeugt werden. Diese dienen zur Verdeutlichung und Analyse des aktuellen Zustands.



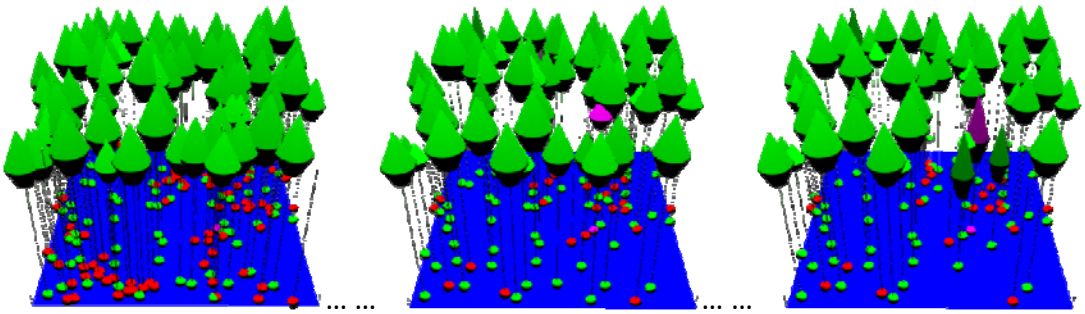
A B C

Abbildung 32: Ausgabecharts der aktuellen Situation vor der Simulation für Baum 65.

5. Simulation über 20 Jahre

Der Simulationsauf erfolgt für alle 3 Varianten mit dem beschriebenen Lichtmodell (s. Kapitel 3.7) über 20 Jahresschritte. Zu Abschluss der Simulation wird die jeweilige neue Bestandesbeschreibung generiert. Zur Verdeutlichung der Situation wird der automatische Ablauf der Simulation zum Ende abermals manuell unterbrochen. Abbildung 33 zeigt die Darstellung der geänderten räumlichen Konkurrenzsituation in GroIMP. Rote Baum-Basispunkte stehen für abgestorbene Bäume, d.h. Bäume deren Nadelmasse im

Laufe der Simulation auf Null gesunken ist. Grüne Punkte zeigen lebende Bäume mit Nadelmasse. Der lila eingefärbte Baum ist der betrachtete Baum 65. Deutlich zu erkennen ist die unterschiedliche Situation des Baumes 65. In Variante A ist er abgestorben. Variante B zeigt die starke Bedrängung durch einen Nachbarbaum und eine kurze Kronenform. In Variante C zeigt Baum 65 eine lange Krone. Abgestorbene Bäume werden bei der Generierung der Bestandesszene mit Status 3 gesetzt.

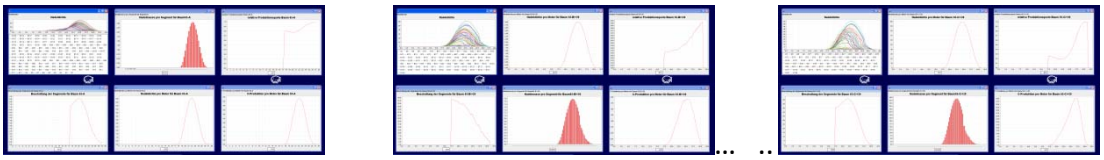


A B C

Abbildung 33: Räumliche Situation nach einem 20-jährigen Simulationslauf in GroIMP. Rote Baumstandpunkte zeigen abgestorbene Bäume.

5.a. Zwischenschritt Analyse

Analog zu Schritt 4.a. werden wiederum relevante Parameter der Bäume ausgegeben. Abbildung 34 zeigt die veränderten Werte für Baum 65.

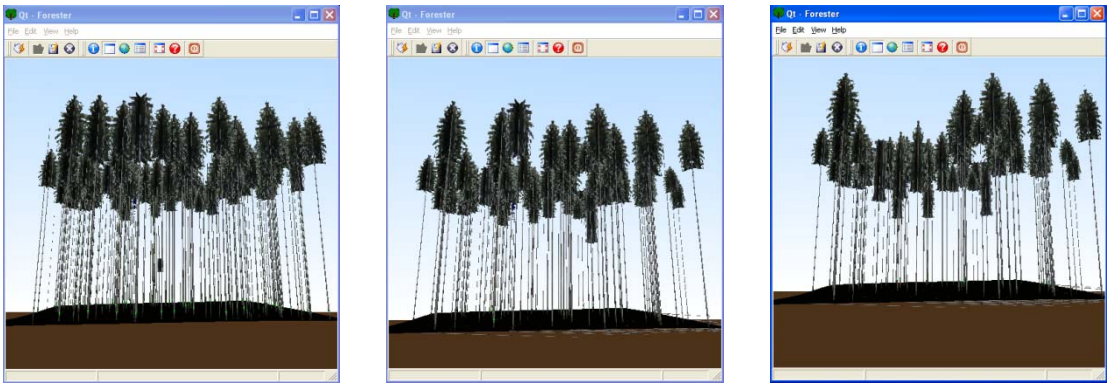


A B C

Abbildung 34: Ausgabecharts der Situation nach der Simulation.

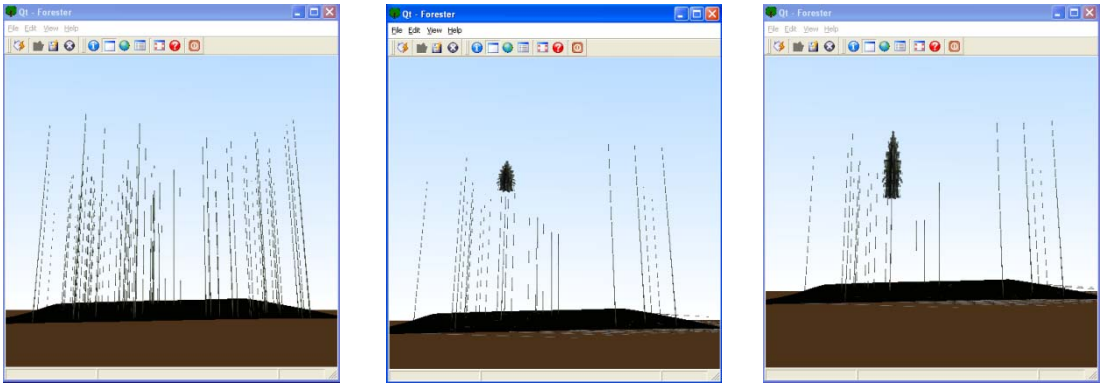
6. Visualisieren der Prognose

Im sechsten Schritt werden die Bestandesbeschreibungen mit den Simulationsergebnissen der 3 Durchforstungsvarianten per HTTP-POST ausgeliefert und im Virtual Forester Client angezeigt. Abbildung 35 zeigt für die Varianten jeweils den gesamten Bestand im Vergleich. In Abbildung 36 werden jeweils nur der Baum 65 und die während der Simulation abgestorbenen Bäume angezeigt. Hierzu wird im VRC die Anzeige der nicht markierten Bäume (Status 0 in der Bestandesbeschreibung) abgeschaltet.



A B C

Abbildung 35: Modelloutput für die drei Varianten im Virtual Forester Client. Angezeigt werden alle Bäume des jeweiligen Bestandes.



A B C

Abbildung 36: Modelloutput für die drei Varianten im Virtual Forester Client. um 65 und alle währen der Simulation abgestorbenen Bäume des jeweiligen Bestandes.

7. Wiederholungen und Variationen

Je nach Zielstellung und Simulationslauf können sich weitere Zyklen mit anderen Durchforstungsszenarien, anderen Beständen oder anderen Modellen anschließen.

4 Lernszenarien

Die Verwendung des beschriebenen Systems ist in verschiedenen Lernszenarien denkbar. Insbesondere durch den Austausch des Wachstumssimulators bzw. der Modellkomponente wird die mögliche Aussagekraft der Wachstumssimulationen stark beeinflusst und damit auch der mögliche Erkenntnisgewinn. Im Folgenden werden 3 Lernszenarien umrissen, für die das System Anwendung finden kann. Mit jedem Szenario wird der mögliche Erkenntnisraum vergrößert, den die Lernenden sich erschließen können. Zugleich werden aber auch die Ansprüche an die Lernenden immer größer. Das in dieser Arbeit enthaltenen Modell ist primär für das Lernszenario 3 gedacht zeigt aber auch – mit Einschränkungen in der Parametrisierung – die Möglichkeiten für Lernszenario 2. Für Lernszenario 1 ist Verwendung eines viel einfacheren Modells sinnvoll. Die Einbindung kann aber analog der beschriebenen Lösung erfolgen, oder durch einen Einbindung eines anderen Wachstumssimulators realisiert werden.

4.1 Lernszenario Stufe 1: Durchforstungstechnik

Beschreibung

Im Virtual Forester-Client werden Durchforstungen an Musterbeständen trainiert. Die Lernenden durchforsten selbstständig nach eigenem Ermessen – einzeln oder in Gruppen – die Bestände und versuchen hierbei vorgegebene Kenngrößen zu treffen (Durchforstungsstärke, Holzvolumen, Bestockungsgrad Baumartenverteilung etc.) Zum Abschluss der Durchforstungsmaßnahme werden die Bestände analysiert und gemäß den Kenngrößen ausgewertet.

Zielgruppe

Studierende und Praktiker in der Forstwirtschaft.

Ziel

Das Erlernen waldbautechnischer Fertigkeiten, wie das Abschätzen von Massen, Kronenräumen und Baumverteilung bei Durchforstungsmaßnahmen.

Ablauf

Die Nutzung des Lernsystems erfolgt ausschließlich im Rahmen von konsolidierenden Übungen als Begleitung zur herkömmlichen Ausbildung. Die Aufgaben werden extern durch den Lehrenden gestellt und können selbstständig durch die Lernenden bearbeitet werden. Die Auswertung erfolgt durch ein entsprechendes Modell automatisch.

Begleitendes Lernmaterial

Grundlagen der Waldbautechnik werden durch diesen Ansatz nur geübt und nicht vermittelt. Begleitendes Material muss für die Vermittlung der Grundlagen bereit gestellt werden.

Rolle der Dozentin

Die Dozentin übernimmt in starkem Maße die Rolle der fachlichen Autorität, die Aufgaben stellt, Wissen vermittelt und bei Problemen eingreift.

Erkenntnisraum für die Lernenden

Aufgrund des Drill-and-Practise-Ansatzes ist der Erkenntnisgewinn durch die Lernenden nur gering. Vielmehr setzt dieser Ansatz auf die Konsolidierung von bereits Gelerntem.

Anforderungen an die Lernenden

Geringe Anforderungen im Bereich des selbständigen und eigenmotivierten Lernens. Geringe Anforderungen in den Bereichen analytische Kompetenz, abstraktes Denken und Gestaltungskraft.

4.2 Lernszenario Stufe 2: Wachstumsprognosen

Beschreibung

Im Virtual Forester Client werden Durchforstungsmaßnahmen an ausgewählten Musterbeständen durchgeführt. Die Lernenden arbeiten selbstständig nach eigenem Ermessen – einzeln oder in Gruppen – in den Beständen und versuchen dabei ihre Kenntnisse über Ursache (Durchforstungen) und Wirkung (Wachstum) in ihre Entscheidung mit einfließen zu lassen. Zu selbstgewählten Zeitpunkten lassen die Lernenden durch den Wachstumssimulator eine Prognose der Bestandesentwicklung berechnen und vergleichen diese mit ihren Erwartungen.

Durch Analyse des zugrundeliegenden Modells, der Resultate von Simulationsläufen bei experimenteller Veränderung der Parametrisierung und ihrer eigenen Erwartung, versuchen die Lernenden die unterschiedlichen Ursache-Wirkung-Beziehungen zu vergleichen und zu bewerten.

Zielgruppe

Studierende der Themen Waldwachstum und Waldwachstumsmodelle

Ziel

Die Lernenden sollen ihre eigenen Annahmen zum Waldwachstum und zum Wirkungsmechanismus von Durchforstungen kritisch hinterfragen und in Beziehung zu weiteren Modellannahmen setzen. Hieraus soll sich ein tieferes Verständnis für die Anwendbarkeit von Modellannahmen zum Pflanzenwachstum im Allgemeinen und zum lichtabhängigen räumlichen Modell im Speziellen entwickeln. Durch die Reparametrisierung sollen die Lernenden ein Gefühl für die Sensibilität von Modellen und von den Wechselbeziehungen der Modellkomponenten entwickeln.

Weiterhin sollen die Lernenden den Einfluss von einzelbaumbasierten Durchforstungsentscheidungen auf die Bestandeskenngrößen abschätzen lernen.

Ablauf

Die Steuerung des Lernprozesses erfolgt durch die Ausgabe eines Themas für eine Hausarbeit und die Bereitstellung von Modellbeständen und begleitendem Material. Regelmäßige Seminare zu einzelnen relevanten Teilthemen können den Diskussionsprozess unter den Lernenden unterstützen.

Begleitendes Lernmaterial

- Erläuterungen und Anleitung zum aktuellen Modell im Modelltutorium in GroIMP.
- Literaturliste mit relevanten Artikeln zum Thema.
- Einführung in die grundlegenden Modellierungstechniken, soweit sie für die Reparametrisierung des Modells notwendig sind, im Tutorium von GroIMP.

Rolle der Dozentin

Die Dozentin übernimmt die Rolle als Tutorin/Coache und überwacht die Fortschritte der einzelnen Lerngruppen. Im Rahmen der Seminare übernimmt er die Moderation des Diskussionsprozesses und die Strukturierung der Themen.

Erkenntnisraum für die Lernenden

Der mögliche Erkenntnisgewinn liegt in den Themenfeldern des Waldwachstums und der praktischen Relevanz von Wachstumsmodellen. Durch den vorhandenen, explorativen Charakter des Lernprozesses ist selbstständiges Erschließen des Erkenntnisraumes in der Kategorie "Verstehen" möglich.

Anforderungen an die Lernenden

Hohe Anforderungen im Bereich des selbständigen Erarbeitens von Wissen. Hohe Anforderungen im Bereich der Eigenmotivation. Hohe Anforderungen im Bereich analytische Kompetenz. Mittlere Anforderungen im Bereich des abstrakten Denkens. Geringe Anforderungen im Bereich der Gestaltungskraft.

4.3 Lernszenario Stufe 3: Methoden der Wachstumsmodellierung

Beschreibung

Im Virtual Forester-Client werden Durchforstungsmaßnahmen an ausgewählten Musterbeständen durchgeführt. Die Lernenden arbeiten selbstständig nach eigenem Ermessen – einzeln oder in Gruppen – in den Beständen und versuchen dabei, ihre Kenntnisse über Ursache (Durchforstungen) und Wirkung (Wachstum) in ihre Entscheidung mit einfließen zu lassen. Zu selbstgewählten Zeitpunkten lassen die Lernenden durch den Wachstumssimulator eine Prognose der Bestandesentwicklung berechnen und vergleichen diese mit ihren Erwartungen.

Nach Analyse des zugrundeliegenden Modells versuchen die Lernenden, die von ihnen erwarteten Ursache-Wirkung-Beziehungen zu operationalisieren und selbst zu modellieren.

Zielgruppe

Studierende der Themen Waldwachstumsmodellierung und Waldwachstum

Ziel

Die Lernenden sollen – ausgehen von ihren eigenen Annahmen zum Waldwachstum und zum Wirkungsmechanismus von Durchforstungen – den Ansatz des vorliegenden Modells kritisch hinterfragen

Hieraus soll sich ein Verständnis für die Anwendbarkeit von Modellannahmen zum Pflanzenwachstum auf den verschiedenen Skalenebenen entwickeln. Im Folgenden sollen die Lernenden durch Entwicklung eines eigenen Modells lernen, Modellannahmen abstrakt zu formulieren und mit XL/GroIMP selbstständig umzusetzen. Hierdurch soll sich ein tieferes Verständnis von verschiedenen Modellierungsansätzen entwickeln.

Weiterhin sollen die Lernenden den Einfluss von einzelbaumbasierten Durchforstungsentscheidungen auf die Bestandeskenngrößen abschätzen lernen.

Ablauf

Die Steuerung des Lernprozesses erfolgt durch die Ausgabe eines Themas für eine umfangreiche Projektarbeit. Themenziel ist die Erstellung eines eigenständigen Wachstumsmodells. Regelmäßige Seminare zu einzelnen, relevanten Teilthemen können den Diskussionsprozess unter den Lernenden unterstützen und den Kontakt mit der Dozentin festigen. Regelmäßige Diskussionstermine der einzelnen Projektgruppen mit der Dozentin dienen der Erfolgskontrolle.

Begleitendes Lernmaterial

- Erläuterungen und Anleitung zu Modellierungstechniken am Beispiel des Modells im Tutorium von GroIMP.
- Nachschlagewerk zu XL/GroIMP mit erweiterten Beispielen im Tutorium von GroIMP.
- Erläuterungen und Anleitung zum aktuellen Modell im Modelltutorium in GroIMP.
- Literaturliste mit relevanten Artikeln zur Wachstumsmodellierung und zum Waldwachstum.
- Onlinedatenbank der SUB.

Rolle der Dozentin

Die Dozentin übernimmt die Rolle als Tutorin/Coache und überwacht die Fortschritte der einzelnen Lerngruppen. Im Rahmen der Seminare übernimmt er die Moderation des Diskussionsprozesses und die Strukturierung der Themen. Besonders zu beachten ist die individuelle Betreuung der einzelnen Lerngruppen bei der Entwicklung des Modells.

Erkenntnisraum für die Lernenden

Der mögliche Erkenntnisgewinn liegt in den Themenfeldern Theorie und Anwendung der Waldwachstumsmodellierung. Durch den stark explorativen Charakter des Lernprozesses ist selbstständiges Erschließen des Erkenntnisraumes in den Kategorien "Verstehen" und "Anwenden können" möglich.

Anforderungen an die Lernenden

Sehr hohe Anforderungen im Bereich des selbständigen Erarbeitens von Wissen. Sehr hohe Anforderungen im Bereich der Eigenmotivation. Hohe Anforderungen im Bereich analytische Kompetenz. Sehr hohe Anforderungen im Bereich des abstrakten Denkens. Hohe Anforderungen im Bereich der Gestaltungskraft.

5 Diskussion

5.1 Einleitung

Die Diskussion wird im Folgenden in drei Abschnitte gegliedert:

Abschnitt 1 beschäftigt sich mit den generellen Anforderungen an eine Lernsoftware und versucht, die Frage zu beantworten, ob E-Learning ein sinnvolles Konzept im Bereich der Wachstumsmodellierung ist.

Abschnitt 2 beschäftigt sich mit der Frage, ob das vorliegende Konzept der Kombination verschiedener Softwarekomponenten den geforderten Kriterien nach Flexibilität und Einfachheit genügt.

Abschnitt 3 diskutiert die Eignung einer Kombination der Sprache XL/Java mit der Entwicklungsumgebung GroIMP und dem beschriebenen ökophysiologischen Modell für einen Einsatz in der Lehre.

5.2 E-Learning

5.2.1 Der Lernprozess

Baumgartner weisst darauf hin, dass es verschiedenen Formen des Wissens gibt, und stellt die Theorie des Hintergrundwissens auf. Seine These "*nimmt zwei prinzipiell unterschiedliche Arten des Wissens an: ein propositional formulierbares Wissen und einen Hintergrund des Wissens. Beide Wissenstypen haben unterschiedliche Struktur und Funktionsweise. […] Während propositionales Wissen sich sprachlich formulieren läßt, widersetzt sich der Hintergrund des Wissens einer analytischen Vorgangsweise.*" [Baumgartner 1993, S. 11]

Auf Basis dieser Theorie benutzt er ein 5-stufiges, hierarchisches Modell der Brüder Dreyfuss [Dreyfuss/Dreyfuss 1987, zitiert in Baumgartner 1993] um den Zusammenhang zwischen *know what* (Faktenwissen), *know how* (prozeduralem Wissen) und Können deutlich zu machen. Zusammengefasst sagt Baumgartner [1993, S. 281ff]:

1. Der Neuling

Der Neuling ist mit der zu lernenden Sache nicht vertraut und hat noch keine diesbezüglichen Erfahrungen. In den ersten Lernschritten geht es um die Vermittlung von eindeutig definierten und überwiegend kontextfreien Fakten und Regeln, die anhand künstlicher Situationen dargestellt werden, und um das Erkennen dieser Fakten durch die Lernende. Vermittelte, kontextabhängige Regeln sind im Allgemeinen heuristische Regeln, die häufig, aber nicht immer zu einem richtigen Ergebnis führen. Sie werden vom Neuling – der mit ihren Feinheiten nicht vertraut ist – wie kontextfreie Regeln behandelt. Das Wissen von Neulingen ist abstrakt und meistens praxis- und realitätsfern.

Charakterisiert werden kann dieses Stadium mit „Wissen, dass etwas der Fall ist."

2. Die fortgeschrittene Anfängerin

Die Stufe der fortgeschrittenen Anfängerin wird erst durch das Sammeln von eigenen Erfahrungen mit der Anwendung der kontextfreien Regeln in wirklichen Situationen erreicht. Dies führt zu einem erweiterten Fundus an Regeln – viele davon auch komplexer Natur.

Gleichzeitig erlernt die fortgeschrittene Anfängerin, verschiedene Situationen tatsächlich als verschieden wahrzunehmen, und erkennt Unterschiede in der Anwendung und Auswirkung der für sie kontextfreien Faustregeln, handelt aber noch nicht danach. Charakterisiert werden kann diese Stufe in etwa mit: „aufgrund eigener Erfahrungen den Kontext spüren, aber noch keine Anpassung der Regeln vornehmen können."

3. Kompetenz

Die Stufe der Kompetenz zeichnet sich durch die bewusste Analyse des Kontextes und eine darauf basierende Auswahl aus, ob und welche der Regeln angewandt werden soll.

Auf der Stufe der Kompetenz arbeitet die Lernende das erste Mal aktiv an der Konstruktion der Situation mit. Maßgebend ist nun der Aspekt der Zielerreichung. Es geht nicht mehr um die Anwendung der einzelnen Regeln, sondern um die Auswirkung der Anwendung. Charakterisiert werden kann diese Stufe mit „wissen, wie man etwas macht und die Fähigkeit dieses "Wie" zu gestalten“.

4. Gewandtheit

In der Stufe der Gewandtheit wird die Situation nicht mehr in einzelne Elemente zerlegt und bewertet, sondern es kommt zu einen “*holistischen Erkennen von Ähnlichkeiten*“ [Dreyfuss & Dreyfuss 1987, S. 52 zitiert in Baumgartner 1993, S. 288]. Es stellt sich ein persönlicher Blickwinkel ein, der auf den Erfahrungen des kompetenten Handelns beruht. Das Wissen wird in der gewandten Person integriert und kann durch die „*intuitive Nutzung von Mustern, ohne die Situation in ihre Komponenten zerlegen zu müssen*“ [Dreyfuss & Dreyfuss 1985, S. 52 zitiert in Baumgartner 1993, S. 289] charakterisiert werden.

5. Expertinnentum

Das Expertinnentum beschreibt Baumgartner als das Verwachsen der in Stufe 4 erworbenen intuitiven Fertigkeiten mit dem Körper, so dass sie überhaupt nicht mehr bewusst erscheinen. Die Expertin wendet ihr Wissen an, ohne bewusst die Problemstellung zu erfassen oder zu analysieren. In vielen Fällen ist eine nachträgliche Analyse und distanzierte Betrachtung einer getroffenen Entscheidung für die Expertin kaum möglich.

Wenngleich diese schematische und hierarchische Betrachtung der 5 Lernstufen uns ein Verständnis der Entwicklungsschritte der Lernenden auf dem Weg zur Expertin vermittelt, warnt Baumgartner davor, den Lernprozess hierarchisch konzeptionieren zu wollen. Für einen ganzheitlichen Lernprozess trifft er vielmehr einige wesentliche didaktische Rückschlüsse:

Es gibt keine Automatik, um von einer Stufe zur anderen zu kommen. Das Lernen von mehr kontextfreien Regeln hilft nicht, die nächste Stufe zu erreichen. Die Lernenden sollen ermutigt werden, durch fortlaufende Praxis eine Evaluation der Situation vorzunehmen. Der Lernprozess sollte so organisiert werden, dass die Unzulänglichkeiten der unteren Stufen deutlich werden und zur nächsten Stufe führen. Die fortlaufende Praxis und Evaluation ist das Bindeglied zwischen den Lernstufen. Eine schrittweise Abfolge von der „unteren“ zur „höheren“ Ebene muss verworfen werden.

5.2.2 Bewertung der Verwendung von E-Learning für dieses Fach

Betrachten wir das Themengebiet der Waldwachstumslehre, so fällt auf, dass insbesondere die langen Zeiträume zwischen Ursache – der Durchforstungsmaßnahme – und Wirkung – sichtbare Veränderungen in der Bestandesstruktur – die Erfahrungskomponente eines Lernprozesses schwierig gestalten. Gerade diese Komponente ist aber essentiell für den Lernfortschritt (vergl. Kapitel 5.2.1). In der klassischen universitären Lehre werden neben den Vorlesungen Exkursionen und Übungen im Wald angeboten, um den Studierenden die notwendigen Fertigkeiten zu vermitteln. Leider kann auch in Exkursionen die Dynamik struktureller Veränderungen nur unzureichend dargestellt werden. Anschauungsbeispiele sind immer nur Momentaufnahmen. Weiterhin stellt sich das Problem, dass durch die komplexen Beziehungen im Ökosystem Wald nur die wenigsten Fakten und Regeln kontextfrei sind. Es besteht daher die große Gefahr, dass sich die Lernenden in einer der klassischen Gefahren der ersten Lernstufen verfangen und sich im Übermaß auf die gelernten Fakten und Regeln konzentrieren und diese als das Lernziel ansehen und nicht als einen ersten Schritt im Lernprozess. Faustregeln manifestieren sich allzu leicht als feststehende Wahrheiten und werden nicht in ihrem Kontext beurteilt.

Viele Autoren von Wachstumssimulatoren schlagen eine Nutzung Ihrer Programme in der Lehre vor [vergl. Fabrika 2003, Nagel 2003, Pretzsch et al. 2002, Pretzsch & Seifert 1999]. In der Verwendung von Computermodellen zum Waldwachstum bietet sich in der Lehre eine Möglichkeit zur verbesserten Darstellung der Dynamik und der Zusammenhänge. Die Möglichkeit der Bestandesvisualisierung erlaubt die Erschaffung virtueller Welten.

Es ergibt sich also die These, dass Bestandessimulatoren – als ein E-Learningwerkzeug – ein Mittel sind, in den Lernprozess in beschränktem Maße Erfahrungen einzubringen, die in der Realität nur über große Zeiträume hinweg ermittelt werden können.

An dieser Stelle stellt sich dann schnell die erste Frage, ob die virtuellen Welten die Exkursionen ersetzen können. Nähern wir uns dieser Frage anhand des beschriebenen Lernszenarios 2 zu Waldwachstumsprognosen (s. Kapitel 4.2) und des beschriebenen Lernsystems.

Jede Softwarelösung bildet die Realität nur dahingehend ab, wie die Erstellerin der Software die Realität begreift. Dies erstreckt sich im vorliegenden System auf der Ebene der virtuellen Welten von der Auswahl der dargestellten Eigenschaften über deren Darstellung bis hin zu den möglichen Interaktionen. Auf der Ebene des Modells sind die Einschränkungen noch deutlicher. Das Modell ist ein abstrahiertes Abbild einer Realität, welches die Expertinnen für sich und ihre Studierenden konstruieren. Dies umfasst die gewählten Bestände für die Simulation und geht über den verwendeten Modellansatz bis hin zur Parametrisierung der einzelnen Funktionen. Jede getroffene Entscheidung im Gedankenbild der Dozentin überträgt sich auf die Software, das Modell und so auf die Lernenden.

Es muss also wiederum darauf geachtet werden, dass die Lernenden nicht auf der Ebene des Faktischen stecken bleiben, d. h. den Weg finden, das Gedankenbild der Dozentin zu überwinden.

Im vorliegenden Lernszenario 2 bedeutet dies, dass es nicht darum gehen darf, für das vorliegende Modell eine richtige Parametrisierung zu finden oder eine weitere Baum-

art einzufügen. Vielmehr ist Ziel, dass die Lernenden das Modell als Basis benutzen, um sowohl ihr Regelwissen wie auch das der Dozentin zu hinterfragen und zu einem tieferen Verständnis des Kontextes zu gelangen.

Die Dozentin übernimmt hierbei die Rolle der Tutorin und versucht durch Anregung einer kritischen Diskussion einen Prozess anzuleiten, der es den Lernenden ermöglicht, ihr propositionales Wissen zu einem stabilen Netzwerk zu verknüpfen und aktiv eigene Ansätze zu entwickeln.

Insbesondere im Übergang von der Stufe der fortgeschrittenen Anfängerin zur Stufe der Kompetenz ist es aber die Auseinandersetzung mit der Realität, die es am sichersten ermöglicht, die immanenten Grenzen der Lernsoftware zu überwinden und aktiv eigene Problembeschreibungen zu formulieren [vergl. auch Baumgartner 1997]. Wir können die virtuellen Welten der Waldwachstumsprognosen also vielmehr als eine Vorbereitung auf die Exkursionen sehen.

Anders hingegen sieht die Beurteilung aus, wenn wir das Lernszenario 3 zu den Methoden der Waldwachstumsmodellierung betrachten. Wachstumsmodelle bleiben immer eine abstrahierte Form der wirklichen dynamischen Prozesse im Wald. Die Formulierung findet immer mit Hilfe eines fachspezifischen Symbolsystems statt – seien dies mathematische Gleichungssysteme, Programmiersprachen oder relationale Wachstumsgrammatiken. Beim Lernprozess zu diesem Thema liegt die Realität also im Abstrakten. Das „objektiv wahre“ Modell existiert nicht. Es gibt dementsprechend auch nicht eine Realität sondern vielmehr einen Raum vom Möglichkeiten, der von verschiedenen Modellierungsansätzen, Methoden und Erklärungsebenen definiert wird. Sich diesen Raum zu erschließen ist die Realität für die Lernenden im Lernszenario 3.

Die oben gemachten Aussagen zu den immanenten Grenzen der Lernsoftware gelten für dieses Lernszenario daher in besonderem Maße. Wie in Kapitel 1.2 ausgeführt, ist nach dem konstruktivistischen Ansatz das Lernen ein aktiver Prozess, bei dem die Lernenden eigenständige Aufgaben- und Problemstellungen entwickeln. Für den Lernerfolg ist es also notwendig, dass die Lernenden sich selbst eine innere Ordnung im Themenfeld der Waldwachstumsmodellierung konstruieren. Obwohl die Plausibilität der Modellannahmen und die Evaluierung der Simulationsergebnisse im Raum der Möglichkeiten der Wachstumsmodellierung ein wichtiger Bestandteil sind, sind sie nicht primäres Ziel des Lernprozesses. Es geht nicht darum, ein richtiges Modell zu erlernen, sondern vielmehr darum, durch eigene Erfahrungen die Grenzen der einzelnen Ansätze zu erkennen und die Fähigkeiten zur Gestaltung neuer Ansätze zu entwickeln.

Plausibilität und Evaluation sind Elemente, die die Lernenden selbstständig im Rahmen der Analyse ihres Erfahrungsschatzes einbauen.

5.2.3 Bedeutung für die Studierenden

Jede noch so gute Serviceleistung (z. B. die Verteilung von Skripten oder die Bereitstellung von Software) scheitert, wenn sie nicht von einem Akt der individuellen persönlichen Erkenntnis [vergl. Polanyi 1962, 1969] genützt wird. Auch wenn wir durch unsere Reden, Schriften oder andere Medien auf etwas hindeuten können, so bleiben die Erfassung der Gestalt und die Anwendung der Anstrengung den individuell Studierenden vorbehalten. *„Und dieser Akt des Verstehens ist gerade **nicht** umgekehrt proportional zu den ihm vorgelagerten didaktischen Anstrengungen“* [Baumgartner 1999 (*„**nicht**“* im Original durch kursiv hervorgehoben, Anmerkung des Autors].

Klares Ziel einer universitären Ausbildung ist es, diesen Vorgang der persönlichen Erkenntnis bei den Studierenden zu induzieren. Wesentlich hierfür wiederum ist es, wieweit es gelingt, die Studierenden für den Lernprozess zu motivieren. Deci und Ryan zeigen, dass die intrinsische Motivation des Menschen – wie z. B. die Neugier – von Ereignissen verstärkt wird, die als Unterstützung der eigenen Selbstständigkeit und als Förderung der eigenen Kompetenz empfunden werden. Sie nennen diese Ereignisse informatorisch. Ereignisse hingegen, die als Druck in Richtung eines speziellen Ergebnisses empfunden werden, verringern die intrinsische Motivation. Diese Ereignisse nennen sie kontrollierend [vergl. Deci und Ryan 1985].

Das in dieser Arbeit beschriebene System hat den Vorteil, ein zeitlich unabhängiges und selbstbestimmtes Lernen zu realisieren. Für die Zielsetzung in Lernszenario 1 kann ein direktes Feedback über das Modell erfolgen, so dass die Lernende nahezu autonom ist. Hintergrundwissen kann durch das begleitende Material bereitgestellt werden. In diesem Sinne ist das System geeignet, die intrinsische Motivation zu stärken.

Deci und Ryan [1985] halten aber auch fest, dass Ereignisse, die dazu führen, dass eine Person eine Aktivität nicht meistern kann, d. h. bei ihr das Empfinden von Inkompetenz auslösen, die intrinsische Motivation untergraben. Sie nennen diese Ereignisse demotivierend.

Lernszenario 2 und insbesondere Lernszenario 3 beinhalten in ihren Lernzielen und ihrer Aufgabenstellung eine hohe Komplexität, die bei völlig eigenständiger Arbeit das Risiko hoch erscheinen lässt, an unlösbaren Problemen – so werden sie zumindest von den Lernenden empfunden – zu scheitern. Da es in der Zielsetzung gerade darum geht, eigenständig Problemstellungen zu entwickeln, ist es systemimmanent, dass auch das begleitenden Material nicht alle aufkommenden Fragen klären kann. Es gilt also Wege zu finden, die Lernenden vor demotivierenden Ereignissen zu schützen.

Als eine Maßnahme wurde zu diesem Zweck in das Virtual Forester-Projekt (Client und Elan Sim Server) die Teamfähigkeit eingebaut. Der Lernprozess kann so für Lerngruppen gestaltet werden.

Wichtiger als die technischen Komponenten im System ist aber die Rolle der Dozentin als Tutorin/Coach im Lernprozess. Der von ihr organisierte kritische Diskurs mit und zwischen den Kurseilnehmerinnen hat existenzielle Bedeutung in den Anwendungsszenarien. Auch ist es ihre Aufgabe, die möglichen Fehlentwicklungen auf den einzelnen Lernstufen zu beobachten, zu erkennen und durch geeignete Maßnahmen zu korrigieren [vergl. Baumgartner 1993, S. 296ff].

Abschließend kann man sagen, dass das in dieser Arbeit beschriebene Softwaresystem einen möglichen Ansatz zum selbstbestimmten Lernen im Bereich der Waldwachstumsmodellierung bieten soll. Die Studierenden gewinnen in starkem Maße Kontrolle über den Lernprozess, werden aber stark in die Eigenverantwortung genommen. Der Lernprozess muss durch die Studierenden aktiv gestaltet werden. Die Lehrenden müssen ebenfalls ihre Rolle ändern. Weg vom klassischen Dozieren übernehmen sie stärker die Rolle einer Tutorin, die einen kritischen Diskurs mit den Studierenden pflegt, ohne den Lernprozess dominieren zu wollen. Im Folgenden gilt es also zu diskutieren, inwieweit das beschriebene System die Anforderungen des selbstbestimmten Lernens erfüllt.

5.3 Beurteilung des Systems für die Lehre

Virtuelle Welten werden bereits in vielen Bereichen verwendet. Die Leistungsfähigkeit moderner Computer erlaubt mittlerweile die 3-dimensionale realitätsnahe Darstellung von komplexen Objekten als begehbare Welt. Wo noch vor Jahren teure Spezialcomputer nötig waren, reicht heute ein normaler PC. Stark vorangetrieben von den Computerspielen, haben sich die Anwendungen von Virtueller Realität auf zahlreiche Anwendungsgebiete ausgebreitet. Auch in die Forstwirtschaft und Landschaftsplanung haben diese Möglichkeiten Einzug gehalten [s. Fabrika 2006, Nagel et al. 2006, Pretzsch & Seifert 1999].

Die Möglichkeiten zur Beschreibung solcher Welten haben sich ebenfalls erheblich vereinfacht. Neben Ansätzen, die wie Java3D [Sun: Java3D] in eine Computersprache integriert sind, steht mit der Virtual Reality Modelling Language (VRML) [Diehl 1997] auch eine mächtige und flexible Sprache zur Beschreibung von virtuellen Welten zur Verfügung, die von der Umsetzung im Visualisierungsprogramm getrennt ist. Das in dieser Arbeit beschriebene System nutzt VRML, um plattformunabhängige Beschreibungen von Waldbeständen für die Lehre nutzbar zu machen. Grundsätzlich können diese Beschreibungen in jedem VRML-Viewer [vergl. blaxxun 3D-Viewer] visualisiert werden. Diesen Ansatz verfolgt auch Fabrika [2006] mit der Visualisierungskomponente von SIBYLA (s. Kapitel 2.2.3).

Im Gegensatz zu Programmen wie BWINPro, SILVA und SIBYLA [Nagel et al. 2006, Pretzsch et al. 2002, Fabrika 2006], die als integrierte Software-Suiten neben der Funktionalität zur Berechnung von Wachstumsprognosen auch umfangreiche andere Programmteile wie Datenbanken, Analysefunktionen, Visualisierungsmodule und ggf. auch Strukturgeneratoren enthalten, besteht das vorliegende System aus verschiedenen Programmen und Komponenten, die über das Internet in ein gemeinsames System eingebunden werden (s. Kapitel 3). Der Vorteil einer solchen offenen Architektur liegt in der Flexibilität, mit der Komponenten im System verändert, ausgetauscht oder neu hinzugefügt werden können. Insbesondere mit dem Austausch des Wachstumssimulators und der Modellkomponente können so unterschiedliche Nutzungsszenarien bedient werden.

Ausgehend von den im Kapitel 5.1 diskutierten Aspekten des selbstbestimmten Lernens ist die Flexibilität, sich den Lernprozessen der Studierenden so weit wie möglich anzupassen, die wichtigste Forderung an ein Softwaresystem im Lehreinsatz. Das vorliegende System legt daher seinen ersten Schwerpunkt auf die **Flexibilität**. In anderen Einsatzgebieten hingegen, in denen eine klar zielgerichtete Nutzung angestrebt wird, haben integrierte Lösungen wie die oben genannten Programme deutliche Vorteile.

Herzstück des beschriebenen Systems ist die Bestandesbeschreibung, welche die Datenkomponente bildet. Wie in Kapitel 1.4.1 ausgeführt wird, liegt ein weiterer Schwerpunkt der Software auf einer **einfachen Nutzung**. In Kapitel 3.2.1 wurde gezeigt, welche Konsequenzen diese Forderung für die Bestandesbeschreibung hat. Es war erklärtes Ziel bei der Konzeption der Bestandesbeschreibung, das Format auch für VRML-Laien lesbar und verständlich zu halten. Dies war einer der zwingenden Gründe, die zur Entwicklung des Virtual Forester Client führten. Zur Vereinfachung der Bestandesbeschreibung mussten dynamische Elemente aus der VRML-Szene in den Virtual Forester Client als Visualisierungsprogramm überführt werden. Als zweiter Grund ist der

Wunsch zu nennen, die Lernprozesse in Gruppen organisieren zu können, d. h. mehrere Client-Programme in einer gemeinsamen Arbeitssession in einem virtuellen Bestand zu verbinden. Dies führte zur Entwicklung des Elan Sim Servers. Vor dem Hintergrund der aktuellen Entwicklungen in der VRML-Community ist zumindest der zweite Grund nicht mehr zwingend. Mittlerweile sind VRML-Server wie der von blaxxun [blaxxun Contact] verfügbar, um diese Funktionalität zu bieten. Auch wenn man diskutieren kann, wie kompliziert eine Bestandesbeschreibung für einen VRML-Laien sein darf, behält das Argument der maximalen Einfachheit seine Gültigkeit. Vorbild für den Grad der Einfachheit für die Bestandesbeschreibung war das Program SVS [McGaughy 1997]. Dennoch muss darauf hingewiesen werden, dass Eigenentwicklungen immer mit Risiken – insbesondere Wartungsaufwand und Programmierfehlern – verbunden sind.

Die Schwerpunkte Flexibilität und Einfachheit standen auch Pate bei der Konzeption der Schnittstelle zur Einbindung der Wachstumssimulatoren in das System. Mit http-POST [Fielding et al. 1999] als Protokoll für den Austausch von Bestandesbeschreibungen zwischen dem Elan Sim Server und den Wachstumssimulatoren wurde bewusst ein Standardprotokoll verwendet, welches es erlaubt, über das Internet ortsungebunden Wachstumssimulatoren anzuschließen. Wachstumssimulatoren mit entsprechendem HTTP-Interface werden auf diese Weise zum Web-Service. Als Perspektive für die Zukunft wäre es denkbar, über solche Web-Services ein Netzwerk von Prognosemodellen für eine gemeinsame Nutzung im Austausch zwischen Forschungs- und Lehreinrichtungen aufzubauen. Die Anbindung der einzelnen Wachstumssimulatoren bleibt dabei den jeweiligen Autoren der Software selbst überlassen. In Kapitel 3.4.6 wurde gezeigt, dass verschiedene Ansätze hierfür zur Verfügung stehen.

Die Nutzung solcher Web-Services bleibt aber nicht auf Kursgruppen beschränkt, die über den Elan Sim Server angebunden werden. Ein auf jedem PC verfügbarer Web-Browser reicht aus, eine Bestandesbeschreibung im Einzelplatzbetrieb manuell zu versenden.

Die Anbindung verschiedener Wachstumssimulatoren und unterschiedlicher Prognosemodelle setzt natürlich voraus, dass die VRML-Beschreibung alle notwendigen Informationen über den Bestand enthält, die für die Simulation benötigt werden.

Wachstumssimulator und Bestandesbeschreibung bilden immer eine Einheit. Neben den Standardelementen, die im Virtual Forester Client angezeigt werden, wird über die labels/data-Felder (s. Kapitel 3.2.4.1: Zusätzliche Informationen) und das worldInfo-Element (s. Kapitel 3.2.3) sichergestellt, dass alle modellspezifischen Daten in einer Bestandesbeschreibung bereitgestellt werden können. Dieser Mechanismus erlaubt jedem Wachstumsgenerator, die nötigen Informationen in der Szene zu speichern, ohne die Übersichtlichkeit zu beeinträchtigen und die Darstellung zu gefährden. Kontrollelemente wie das GGInfo-Element verhindern bei Verwendung des Elan Sim Servers Fehlbedienungen.

Zusammenfassend hofft der Autor gezeigt zu haben, dass das Zusammenspiel der Komponenten den Kriterien der Flexibilität und Einfachheit so weit wie möglich entspricht und den in Kapitel 1.3 angestrebten Zielen dienlich ist.

5.4 Modelle auf Basis von relationalen Wachstumsgrammatiken und GroIMP

Neben der **Flexibilität** und der **einfachen Nutzung** sind der dritte und vierte Schwerpunkt für das vorliegende System und dessen Einsatz in der Lehre die **Verständlichkeit** und **Transparenz** der Modellkomponente.

Die Anforderungen an die Transparenz der Modellkomponenten sind in der Regel einfach zu erfüllen. Das Modell muss in allen Teilen der Berechnung dokumentiert sein, und im Idealfall liegen die Berechnungen offen. Zieht man aber die Verständlichkeit und die einfache Nutzung mit in die Bewertung ein, so stellt sich die Bewertungssituation komplexer dar. Selbst offengelegter, prozeduraler Programmcode erfüllt nur in wenigen Fällen die Anforderung der Verständlichkeit. Auch ist die Nutzung nur für erfahrene Programmiererinnen einfach.

Nach Ansicht des Autors bietet die Sprache XL/Java in Kombination mit der Entwicklungsumgebung einen Ausweg aus diesem Zwiespalt. Der regelbasierte Sprachteil von XL beruht auf dem Konzept der Relationalen Wachstumsgrammatiken [vergl. Kniemeyer 2007] und ist äußerst effizient bei der Modellierung von Pflanzen- oder Bestandesstrukturen. Mit anfänglich nur geringen Kenntnissen der Programmiersprache Java lassen sich regelbasierte Modelle erstellen. Die umfangreichen Klassenbibliotheken für räumliche graphische Objekte in GroIMP erlauben dabei eine einfache Visualisierung der Struktur. Durch die hohe Spezifizierung bleibt der Programmcode dabei dennoch verständlich und einfach anzuwenden. Relationale Wachstumsgrammatiken sind aufgrund ihrer Affinität zu Pflanzenstrukturen ein sehr eingängiger Ansatz für raumbezogene Pflanzenmodelle, XL/Java eine effiziente Implementation dieses Ansatzes (s. Kapitel 3.6)

Der imperative Teil von XL/Java hingegen – zu dem der volle Umfang der objektorientierten Sprache Java gehört – eröffnet mit zunehmenden Programmierkenntnissen den vollen methodischen Umfang der funktionsorientierten Modellierung. Zudem eröffnet die Integration von Java den Zugriff auf evtl. bereits existierende Klassenbibliotheken im Bereich der Wachstumsmodellierung. So kann auf Basis von GroIMP die Modellkomponente verändert werden, ohne die Anbindung eines weiteren Wachstumssimulators vornehmen zu müssen. Wie in Kapitel 3.8.4.2 beschrieben, erscheint hierfür besonders die Einbindung der TreeGrOSS Bibliotheken [Nagel 2002, 2003] eine ideale Möglichkeit zur Erweiterung des Anwendungsspektrums in der Lehre zu sein.

Will man den Studierenden, wie in Kapitel 5.2 diskutiert, die Möglichkeit zum selbstbestimmten Lernen geben, so sind diese Eigenschaften der Kombination von XL/Java und GroIMP eine gute Grundlage für die Gestaltung des Lernprozesses. Durch Verzahnung von Modelltutorien mit den Modellteilen wird ein gutes Angebot von begleitendem Material direkt integriert. Zudem kann ein umfangreiches optionales Angebot von Grundlagen- und Vertiefungswissen als Hyper-Media realisiert werden, auf das die Lernenden zugreifen können, ohne einem obligatorischen Weg folgen zu müssen.

Natürlich werden die Verständlichkeit, die Transparenz und die einfache Nutzung wesentlich von der konkreten Modellkomponente, d. h. von ihren theoretischen Ansätzen und von ihrer Umsetzung vorgegeben.

Das im Kapitel 3.7 beschriebene Modell stellt die Umsetzung eines einfachen ökophysiologischen, raumbezogenen Ansatzes dar. In der Beschreibung des Modells wurde bereits auf die vorhandenen Grenzen und Einschränkungen des Ansatzes und der Parametrisierung eingegangen. Unabhängig davon hofft der Autor, mit dem Modell gezeigt zu haben, wie funktions-/strukturorientierte Modelle mit XL/Java umgesetzt und für die universitäre Lehre verfügbar gemacht werden können. Er folgt damit dem Gedanken, dass die Studierenden im explorativen Lernen einen Zugang zur Wachstumsmodellierung finden, durch eine kritische Betrachtung des Modells Grenzen und Möglichkeiten erfahren und mit zunehmender Beschäftigung mit der Materie eigenständige Ansätze entwickeln.

5.5 Ausblick

Das vorliegende System stellt einen ersten Schritt auf dem Weg zu einem vielseitigen Lernsystem für Waldwachstumsmodellierung dar. Es bleiben aber zahlreiche Aspekte, die in Zukunft verbessert werden können. Diese betreffen sowohl die einzelnen Programme als auch die Modellkomponente und die mögliche Integration in einen Lehrprozess.

Zu nennen ist hier z.B. der Ausbau des in GroIMP integrierten Tutoriums. Anzustreben wäre ein vollständiges Hyper-Media Tutorium zu den Grundlagen der Relationalen Wachstumsgrammatiken, der Sprache XL/Java und dem Programm GroIMP, sowie modellspezifische und anwendungsspezifische Ergänzungen.

Ein weiterer Aspekt für die Weiterentwicklung des Systems betrifft die Modellkomponente. Hier sind in erster Linie die Ergänzung weiterer Baumarten und die Validierung des bestehenden Ansatzes mit Messdaten zu nennen. Aber auch die Umsetzung alternativer Modellansätze, z.B. durch Verwendung der TreeGrOSS-Funktionen ist sehr wünschenswert.

Für den Virtual Forester Client stehen Funktionen zur Verbesserung der Übersicht und Orientierung, z.B. Kronenraumkarten und direkte Sprungoptionen zu Bäumen, auf der Liste der nächsten Entwicklungsschritte. Auch sollte die Textur-Datenbank mit den Qualitätskriterien für alle forstlich relevanten Baumarten ausgebaut werden.

Und für das Programm GroIMP stehen verbesserte interaktive Eingriffsmöglichkeiten in die Simulation auf der Aufgabenliste der Entwickler.

Weiterer Bedarf für eine Weiterentwicklung des Systems und seiner Komponenten wird aber mit Gewissheit im Rahmen der Anwendung von den Studierenden selbst formuliert werden. Wie ein Lernprozess ist auch Lehrmaterial nie abschließend fertig.

6 Zusammenfassung

Die vorliegende Arbeit behandelt den Aufbau eines E-Learning-Systems, welches den Ansatz der Funktions-/Strukturmodellierung für die Verwendung in der universitären Lehre zur Waldwachstumsmodellierung benutzt. Das E-Learning-System verwendet „Virtuelle Realität" – in Form von begehbaren, interaktiven virtuellen Waldbeständen – kombiniert mit Waldwachstumssimulatoren – zur Prognoseberechnung möglicher Waldentwicklungen – als explorative Lernumgebung zur Erforschung der Wachstumsmodellierung durch die Studierenden der Forstwissenschaften und Waldökologie.

Das Waldwachstum ist durch lange Zeiträume geprägt. Zur Abschätzung der zukünftigen Waldentwicklung – z.B. nach einem waldbaulichen Eingriff – werden mathematische Modelle verwendet. Für den Bereich der Modellierung der Walddynamik bietet sich daher das Computerexperiment als ideale Ergänzung zur Verbreiterung des Erfahrungsspektrums in der universitären Lehre an.

Primäre Zielsetzung des E-Learning-Systems ist der Aufbau einer vorbereiteten Arbeitsumgebung für das explorative Lernen der Studierenden. Der Ansatz des explorativen Lernens – welcher der didaktischen Theorie des Konstruktivismus nahe steht – stellt die persönliche Erfahrungen der Studierenden in den Vordergrund: Erfahrungen, die durch eigenständiges Arbeiten mit Faktenwissen, Theorien und Hypothesen die Studierenden zu einem tieferen Verständnis des Wissenskontextes und der Zusammenhänge führen sollen.

Dementsprechend werden für ein Lernsystem die grundlegenden Anforderungen der **Flexibilität**, der **Verständlichkeit**, der **Transparenz** und der **einfachen Nutzung** angestrebt.

Für den deutschen Raum gibt es mit BWINPro [NAGEL], SILVA [PRETZSCH] und SIBYLA [FABRIKA] mehrere einzelbaumorientierte Simulatoren für Walddynamik, mit teilweise unterschiedlichen Ansätzen – z.B. zur Behandlung der räumlichen Konkurrenz – die in der universitären Lehre angewandt werden und über zahlreiche Funktionen verfügen. Jedoch ist bei diesen Programmen, obwohl sie großteils die geforderten Eigenschaften besitzen, der Modellteil fest in den kompilierten Programmcode integriert und somit für die Studierenden nur mit umfangreichen Programmierkenntnissen verfügbar.

Um die, für das explorative Lernen im Themenfeld der Walddynamik notwendige Flexibilität in der Formulierung der Modellannahmen zu erreichen, verfolgt das in der Arbeit beschriebene Lernsystem auf zwei Ebenen einen neuen Ansatz.

Auf der ersten Ebene steht die Systemarchitektur: Das System integriert verschiedene Programme in eine offene, internetbasierte Architektur. Zentrales Verbindungsglied und Datenkomponente des Systems ist ein speziell entwickeltes Beschreibungsformat für Waldbestände auf Basis der "Virtual Reality Modelling Language" (VRML). Diese Bestandesbeschreibung enthält ein einfaches – für den Informatik-Laien verständliches – Format zur Definition von virtuellen Waldbeständen und ist gleichzeitig flexibel genug, die zusätzlich notwendigen Daten verschiedener Wachstumsmodelle aufzunehmen.

Die Visualisierung der Bestandesszenen übernimmt der Virtual Forester Client – ein eigens für die forstliche Lehre entworfener interaktiver VRML-Viewer. Er bildet die

graphische Schnittstelle für die Benutzerin und ermöglicht das interaktive Behandeln des Bestandes (Bäume fällen, Bäume auszeichnen, Baumdaten abfragen etc.).

Die Verbindung verschiedener Client-Programme über das Internet realisiert der Elan Sim Server. Er erlaubt es mehreren Studierenden, simultan in einer Bestandesszene zu arbeiten, miteinander zu kommunizieren und gemeinsam Problemlösungen zu erarbeiten.

Als Wachstumssimulator wurde die Growth "Grammar-related Interactive Modelling Platform" (GroIMP) in das E-Learning-System mit eingebunden. GroIMP [KNIEMEYER] ist eine neue offene Modellierplattform für die Sprache XL/JAVA mit zahlreichen Klassenbibliotheken für räumliche Objekte, Turtle-Geometrien und I/O Operationen. Über einen integrierten http-Server ist GroIMP ebenfalls über das Internet in das System eingebunden und kann als Web-Service für Wachstumsmodellierung dienen.

Die zweite Ebene betrifft das Modell. Die Modellkomponente ist auf Basis der "Relationalen Wachstumsgrammatiken" [KURTH] in der Sprache XL/JAVA umgesetzt. Relationale Wachstumsgrammatiken (RGG) sind die Erweiterung des regelbasierten Ansatzes der Lindenmayer-Systeme – die in der Strukturmodellierung bereits lange erfolgreich eingesetzt werden – von Strings auf Graphen und ermöglichen somit die einfache Überwindung der Skalengrenzen in einem Modell.

Die Sprache XL/JAVA ist eine Implementierung des RGG-Ansatzes und vereint die Möglichkeiten der regelbasierten Programmierung mit den Methoden der imperativen Programmierung einer objektorientierten Sprache, wie sie für prozessorientierte Funktionsmodelle benötigt werden. XL/JAVA stellt – in Verbindung mit der Modellierungsplattform GroIMP – somit die Basis für eine effiziente interaktive Entwicklung von räumlichen Modellen auf allen Skalenebenen des Waldbestandes.

Das realisierte Modell verwendet einen einfachen, räumlichen, ökophysiologischen Ansatz auf Basis der Biomasse [SLOBODA & PFREUNDT]. Die Berechnung der Photosynthese, Respiration, Allokation und des Wachstums erfolgt in Jahresschritten. Dem objektorientierten Ansatz der Funktion-/Strukturmodellierung folgend wird die Nadelbiomasse Segmenten entlang der Stammachse über eine Beta-Verteilung zugewiesen. Die Ermittlung der Photosyntheseleistung erfolgt in Abhängigkeit der jeweiligen beschattenden Nadelmasse in einem Lichtkegel von 60 Grad über dem Segment und in Relation zu unbeschatteten Verhältnissen an der Baumspitze. Respiration und Allokation werden für die 5 Baumkompartimente Nadeln, Äste, Fein- und Grobwurzeln und Stamm getrennt errechnet. Das Stammwachstum wird über eine Alter-Höhenfunktion mit stochastischer und rangerhaltender Korrekturkomponente bstimmt. Das Modell ist initial mit Literaturwerten für Fichte und Kiefer parametrisiert, kann aber für andere Nadelbaumarten erweitert werden.

Ausgehend von der Zielsetzung des E-Learning-Systems bieten die vollständige Offenlegung der Modellkomponente und der Einsatz der regelbasierte Sprache XL den Studierenden ein effizientes Umfeld, um von der Analyse des vorliegenden Modells über dessen schrittweise Veränderung bis hin zur selbstständigen Konstruktion eigener Modellansätze zu gelangen. Eingebunden in ein Ablaufkonzept mit begleitenden Seminaren zur kritischen Diskussion und entsprechendem Begleitmaterial zum Modell, den Grundlagen der Modellierung und der Modellierungstechnik, kann das Lernsystem ein sinnvolles Instrument der computergestützten Lehre zur Modellierung der Waldynamik an den Universitäten Einsatz finden.

Die offene, internetbasierte Architektur des Systems und der einfache, flexible Aufbau der Bestandesbeschreibung erlaubt zudem die Einbindung weiterer Programme – z. B. alternative Wachstumssimulatoren – und somit die Erweiterung der möglichen Anwendungsszenarien.

7 Literaturverzeichnis

Abelson, H., diSessa, A. A., 1982: Turtle Geometry. MIT Press, Cambridge.

Ames, L. A., Nadeau, R. D., Moreland, L. J., 1997: VRML Sourcebook Second Edition. John Wiley & Sons Inc.

Amthor, J. S., 1989: Respiration and Crop Productivity. Springer-Verlag, New York, NY.

Anzola Jürgenson, G., 2002: Linking structural and process-oriented models of plant growth. Dissertation, Georg August Universität Göttingen; abgerufen am 27.03.2007; http://webdoc.sub.gwdg.de/diss/2003/anzola/index.html

Assmann, E., Franz, F.,1963: Vorläufige Fichtenertragstafel für Bayern. Institut für Ertragskunde der Forstlichen Forschungsanstalt München.

blaxxun 3D Viewer: VRML Applet von blaxxuntechnologies; abgerufen am 05.01.2007; http://www.blaxxun.com/home/index.php?option=com_content&task=view&id=40&Itemid=89

blaxxun Contact: VRML Plugin von blaxxuntechnologies; abgerufen am 05.01.2007; http://www.blaxxun.com/home/index.php?option=com_content&task=view&id=42&Itemid=85

Baumgartner, P., 1993: Der Hintergrund des Wissens. Vorarbeiten zu einer Kritik der programmierbaren Vernunft. Kärntner Druck- und Verlagsanstalt, Klagenfurt.

Baumgartner, P., 1997: Didaktische Anforderungen an (multimediale) Lernsoftware. In: Informationen und Lernen mit Multimedia. 2,. überarbeitete Auflage, herausgegeben von Issing, L. J. & Klimsa, P. Psychologie Verlags Union, Weinheim, S. 241-252.

Baumgartner, P., 1999: 10 Todsünden in der Evaluation interaktiver Lehr– und Lernmedien. In: Studieren 2000 - Alte Inhalte in neuen Medien? (mit CDROM), herausgegeben von K. Lehmann. Waxmann, Münster, S. 199-220.

Baumgartner, P., Payr, S., 1994: Lernen mit Software. Reihe Digitales Lernen. Österreichischer Studien Verlag, Innsbruck.

Baumgartner, P., Payr, S., 1997: Erfinden lernen. In: Konstruktivismus und Kognitionswissenschaft. Kulturelle Wurzeln und Ergebnisse. Zu Ehren Heinz von Foersters. K.H. Müller und F. Stadler. Wien-New York, Springer, 8: S. 89-106.

Berners-Lee, T., Fielding, R., Masinter, L., 2005: Uniform Resource Identifier (URI): Generic Syntax, IETF RFC 3986; abgerufen am 05.01.2007; http://www.ietf.org/rfc/rfc3986.txt, http://www.gbiv.com/protocols/uri/rfc/rfc3986.html#URLvsURN

Bosc, A., de Grandcourt, A., Loustau, D., 2003: Variability of stem and branch maintenance respiration in a Pinus pinaster tree. Tree Physiology 23: p. 227–236.

Bossel, H., 1994: TREEDYN3 Forest simulation model. Mathematical model, program documentation and simulation results, Berichte des Forschungszentrum Waldökosysteme, Reihe B 35.

Bossel, H., 1996: TREEDYN3 forest simulation model. Ecological Modeling 90: p. 187–227.

Cannell, M. G. R., Thornley, J. H. M., 2000: Modelling the Components of Plant Respiration: Some Guiding Principles. Annals of Botany 85: p. 45-54.

Deci, E. L., Ryan, R. M., 1985: The general causality orientations scale: Self-determination in personality. Journal of Research in Personality 19: p. 109-134.

Diehl, S., 1997: VRML - Virtual Reality Modeling Language, Informatiklexikon der Gesellschaft für Informatik e.V.; abgerufen am 05.01.2007; http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/90/

Dreyfus, H. L., Dreyfus, S. E. 1987: Künstliche Intelligenz. Von den Grenzen der Denkmaschine und dem Wert der Intuition. Rowohlt, Reinbek b. Hamburg.

DWD, 2006a: Deutscher Wetterdienst, Klimadaten der Station 10315 Münster / Osnabrück, in Tabellenform; abgerufen am 12.09,2006; http://www.dwd.de/de/FundE/Klima/KLIS/daten/online/nat/ausgabe_tageswerte.htm

DWD, 2006b: Deutscher Wetterdienst, Beschreibung der Klimadaten in Tabellenform; abgerufen am 12.09.2006; http://www.dwd.de/de/FundE/Klima/KLIS/daten/online/nat/elemente_tageswerte.htm

Dzierzon, H., 2003: Development of methods for characterizing plant and stand architectures and for model comparisons. Dissertation, Georg August Universität Göttingen; abgerufen am 27.03.2007; http://webdoc.sub.gwdg.de/diss/2003/dzierzon/index.html

Ek, A. R., Monserud, R. A., 1974: Trials with program FOREST: Growth and reproduction simulation for mixed species even- or uneven-aged forest stands. In: Fries, J. (Hrg.): Growth models for tree and stand simulation. Royal College of Forestry, Stockholm, Schweden, Research Notes 30, p. 56-73.

ELAN, 2005: E-Learning Academic Network Niedersachen, Bericht zur Förderphase ELAN I (1.10.2002 – 31.12.2004). ELAN-Management Board (Herausgeber), ISBN 3-8142-0945-1

ELAN, 2006: E-Learning Academic Network Niedersachsen: Wer sind wir? abgerufen am 03.04.2006; http://www.elan-niedersachsen.de/index.php?id=173

Ellenberg, H., 1963: Vegetation Mitteleuropas mit den Alpen: in kausaler, dynamischer und historischer Sicht. Verlag Eugen Ulmer, Stuttgart.

Fabrika, M., 2003: Virtual forest stand as a component of sophisticated systems. Journal of Forest Science 49 (9): p. 419-428.

Fabrika, M., 2005: Simulátor biodynamiky lesa SIBYLA. Koncepcia, konštrukcia a programové riešenie (*Simulator of Forest Biodynamics SIBYLA. Conception, construction and software solution*). Habilitačná práca (*Habilitation work*), Technická univerzita vo Zvolene.

Fabrika, M., Ďurský, J., 2005: Algorithms and software solution of thinning models for growth simulator SIBYLA. Bericht der Jahrestagung der Sektion Ertragskunde im Deutschen Verband Forstlicher Forschungsanstalten, Freising 9.-11. Mai 2005: p. 77 - 91.

Fabrika, M., Ďurský, J., Pretzsch, H., Sloboda, B., 2005: Regionalisation of climatic values for ecological site classification using growth simulator SIBYLA with GIS. In: Remote Sensing and Geographical Information Systems for Environmental Studies. Application in Forestry, Edited by Klein, Ch., Nieschulze, J., Sloboda, B., Band 138, Universität Göttingen, J.D. Sauerländers Verlag, Frankfurt am Main: p. 245-255.

Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P., Bernes-Lee, T., 1999: Hypertext Transfer Protocol -- HTTP/1.1, W3C Specification 2616 1999; abgerufen am 05.02.2007; http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf

FreeWrl: VRML Standalone Browser des Communications Research Centre Canada, sourceforge.net Projektwebsite mit Sourcecode und Dokumentation; abgerufen am 05.01.2007; http://freewrl.sourceforge.net/index.html

Früh, T., 1995: Entwicklung eines Simulationsmodells zur Untersuchung des Wasserflusses in der verzweigten Baumarchitektur. Berichte des Forschungszentrums Waldökosysteme, Reihe B 131.

Früh, T., Kurth, W., 1999: The hydraulic system of trees: Theoretical framework and numerical simulation. Journal of Theoretical Biology 201: p. 251–270.

FSPM, 2004: 4TH International Workshop on Functional-Structural Plant Models: Aim of the series zitiert Risto Sievänen, Annikki Mäkelä, Eero Nikinmaa, organisers of the first FSPM workshop (preface of Silva Fennica, 31, issue 3, 1997); abgerufen am 14.03.2007; http://amap.cirad.fr/workshop/FSPM04/aim.html

Gadow, v., K., 1987: Untersuchungen zur Konstruktion von Wuchsmodellen für schnellwüchsige Plantagenbaumarten. Forstliche Forschungsberichte, München, 77.

Gausche, S., 2003: Unterlagen des Seminars "Konzeption von Medialen Lernveranstaltungen" am Forschungszentrum L3S in Hannover, unveröffentlicht.

Gielen, B., Scarascia-Mugnozza, G., Ceulemans, R., 2003: Stem respiration of Populus species in the third year of free-air CO2 enrichment. Physiologia Plantarum 117: p. 500–507.

GNU:GPL, 1991: General Public License, Version 2, Stand Juni 1991, Free Software Foundation, Inc.; abgerufen am 22.01.2007; http://www.gnu.org/copyleft/gpl.html

Hanisch, B., 1990: Waldschäden erkennen. Verlag Ulmer, London

Hasenauer, H., 1994: Ein Einzelbaumwachstumssimulator für ungleichaltrige Kiefern- und Buchen-Fichtenmischbestände. Forstliche Schriftenreihe, Universität für Bodenkunde, Wien.

Hartmann, J., Wernecke, J., 1996: The VRML 2.0 Handbook – Building Moving Worlds on the Web. Addison-Wesley.

Hauhs, M., Kastner-Maresch, A., Rost-Siebert, K., 1995: A model relating forest growth to ecosystem-scale budgets of energy and nutrients. Ecological Modelling 83: p. 229–243.

Hauhs, M., Knauft, F.-J., Lange, H., 2003: Algorithmic and interactive approaches to stand growth modelling. In: A. Amaro, D. Reed, P. Soares (eds.), Modelling Forest Systems. CABI Publishing, Wallingford, UK.

Heck, C. R., 1931: Handbuch der freien Durchforstung. Schweizerbart, Stuttgart.

Hoffmann, F., 1995: FAGUS, a model for growth and development of beech. Ecological Modelling 83 (3): S. 327-348.

Issing, L. J., 1997: Instruktionsdesign für Multimedia. In: Informationen und Lernen mit Multimedia, 2., überarbeitete Auflage; herausgegeben von Issing, L. J. & Klimsa, P.; Psychologie Verlags Union, Weinheim.

ITU-T:JPEG, 1992: Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines, ISO/IEC IS 10918-1; ITU-T Recommendation T.81, September 1992; abgerufen am 05.01.2007; http://www.itu.int/rec/T-REC-T.81-199209-I/en

Java.net:jogl: Java Bindings for OpenGL auf java.net; abgerufen am 05.02.2007; https://jogl.dev.java.net/

Kahn, M., 1994: Modellierung der Höhenentwicklung ausgewählter Baumarten in Abhängigkeit vom Standort. Forstliche Forschungsberichte München 141.

Kahn, M., Pretzsch, H., 1997: Das Wuchsmodell Silva 2.1-Parametrisierung für Rein- und Mischbestände aus Fichte und Buche. Allgemeine Forst- und Jagdzeitung, 168 (6/7): S. 115-123.

Kellomäki, S., Hari, P., 1980: Eco-Physiological studies on young scots pine stands: I. Tree class as indicator of needle biomass, illumination, and photosynthetic capacity of crown system. Silva Fennica 14 (3): S. 227-242.

Kellomäki, S., Hari, P., Kanninen, M., Ilonen, P., 1980: Eco-Physiological studies on young scots pine stands: II. Distribution of needle biomass and its application in approximating light conditions inside the canopy. Silva Fennica 14 (3): S. 243-257.

Kennel, R., 1969: Formzahl und Volumentafeln für Buche und Fichte. Selbstverlag des Institutes für Ertragskunde der Forstlichen Forschungsanstalt München, München

Kniemeyer, O., 2004: Rule-based modelling with the XL/GroIMP software. In: Harald Schaub, Frank Detje, Ulrike Brüggemann (eds.), The Logic of Artificial Life. Proceedings of 6th GWAL, Bamberg 14.-16.4.2004. AKA Akademische Verlagsgesellschaft, Berlin: S. 56-65: abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/kniemeyer_gwal04.pdf.

Kniemeyer, O., 2007: Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. Dissertation, Brandenburgische Technische Universität Cottbus; in Vorbereitung.

Kniemeyer, O., Buck-Sorlin, G., Kurth, W., 2006: GroIMP as a platform for functional-structural modelling of plants. In: J. Vos, L. F. M. Marcelis, P. H. B. deVisser, P. C. Struik, J. B. Evers (eds.), Functional-Structural Plant Modelling in Crop Production. Proceedings of a workshop held in Wageningen (NL), 5.-8.3.2006. Kluwer, Dordrecht (accepted).

Kniemeyer, O., et al., 2007: GroIMP API Dokumentation: Version 0.94 API; abgerufen am 27.03.2007; http://www-gs.informatik.tu-cottbus.de/grogra.de/api/index.html

Kobayashi, S., 1981: Simulation model of the stand growth of Japanese larch. IUFRO poster session, Kyoto, Japan 11. September 1981.

Kramer, H., 1984: Grundlagen zur forstlichen Ertragskunde. Veröffentlicht durch den Autor, 3. Auflage.

Künstle, E., Mitscherlich, G., 1975: Photosynthese, Transpiration und Atmung in einem Mischbestand im Schwarzwald. I. Teil: Photosynthese. Allgemeine Forst- und Jagdzeitung 146: S. 45-63.

Künstle, E., Mitscherlich, G., 1976: Photosynthese, Transpiration und Atmung in einem Mischbestand im Schwarzwald. III. Teil: Photosynthese. Allgemeine Forst- und Jagdzeitung 147: S. 169-177.

Kurth, W., 1994a: Morphological models of plant growth: Possibilities and eccological relevance. ISEM's 8th International Conference on the State-of-the-Art in Ecological Modelling, 28.9.–2.10.1992, Kiel. Ecological Modelling 75/76: S. 299-308.

Kurth, W., 1994b: Growth Grammar Interpreter GROGRA 2.4: A software tool for the 3-dimensional interpretation of stochastic, sensitive growth grammars in the context of plant modelling. Introduction and Reference Manual. Berichte des Forschungszentrums Waldökosysteme Göttingen, Serie B 38; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/gro.ps.gz.

Kurth, W., 1998: Some new formalisms for modelling the interactions between plant architecture, competition and carbon allocation. Bayreuther Forum Ökologie, 52: S. 53-98. abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/form.ps.gz.

Kurth, W., 1999: Die Simulation der Baumarchitektur mit Wachstumsgrammatiken. Stochastische, sensitive L-Systeme als formale Basis für dynamische, morphologische Modelle der Verzweigungsstruktur von Gehölzen. Habilitationsschrift, Universität Göttingen; Wissenschaftlicher Verlag Berlin.

Kurth, W., 2002: Spatial structure, sensitivity and communication in rule-based models. In: Franz Hölker (ed.), Scales, Hierarchies and Emergent Properties in Ecological Models. Reihe "Theorie in der Ökologie", Bd. 6. Peter Lang, Frankfurt a. M.: S. 29-46; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/helenen.doc.gz.

Kurth, W., Anzola Jürgenson, G.,1997: Triebwachstum und Verzweigung junger Fichten in Abhängigkeit von den beiden Einflussgrößen "Beschattung" und "Wuchsdichte": Datenaufbereitung und -analyse mit GROGRA. In: D. Pelz (ed.), Deutscher Verband Forstlicher Forschungsanstalten, Sektion Forstl. Biometrie u. Informatik, 10. Tagung Freiburg i. Br. 1997, Ljubljana, Biotechn. Fakultät: S. 89-108; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/freibg.doc.gz.

Kurth, W., Bredemeyer, M.,1996: Modellkonzeption und -vernetzung am Forschungszentrum Waldökosysteme (Göttingen). In: K. Mathes, B. Breckling, K. Ekschmitt (eds.), Systemtheorie in der Ökologie, ecomed, Landsberg: S. 67-72; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/grill.doc.gz.

Kurth, W., Buck-Sorlin, G., Kniemeyer, O., 2006: Relationale Wachstumsgrammatiken: Ein Formalismus zur Spezifikation multiskalierter Struktur-Funktions-Modelle von Pflanzen. In: Modellierung pflanzlicher Systeme aus historischer und aktueller Sicht. Symposium zu Ehren von Prof. Dr. Dr. h.c. Eilhard Alfred Mitscherlich, Schriftenreihe des Landesamtes für Verbraucherschutz, Landwirtschaft und Flurneuordnung Brandenburg, Reihe Landwirtschaft 7 (1): S. 36-45; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/mitsch.pdf.

Kurth, W., Kniemeyer, O., Buck-Sorlin, G., 2005: Relational Growth Grammars - a graph rewriting approach to dynamical systems with a dynamical structure. In: J.-P. Banâtre, P. Fradet, J.-L. Giavitto, O. Michel (eds.), Unconventional Programming Paradigms. Lecture Notes in Computer Science 3566, Springer, Berlin: p. 56-72.

Kurth, W., Sloboda, B.,1999a: Tree and stand architecture and growth described by formal grammars. I. Non-sensitive trees. Journal of Forest Science, 45: p. 16-30; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/les1f.gz.

Kurth, W., Sloboda, B.,1999b: Tree and stand architecture and growth described by formal grammars. II. Sensitive trees and competition. Journal of Forest Science, 45: p. 53-63; abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/les2.gz.

Kurth, W., Sloboda, B., 2001: Sensitive growth grammars specifying models of forest structure, competition and plant-herbivore interaction. Proceedings of the IUFRO 4.11 Congress "Forest Biometry, Modelling and Information Science", Greenwich, UK (25.-29. 6.2001) (*in press*); abgerufen am 05.02.2007; http://www-gs.informatik.tu-cottbus.de/~wwwgs/gree_tx.pdf

Lavigne M. B., 1996: Comparing stem respiration and growth of jack pine provenances from northern and southern locations. Tree Physiology 16: p. 847–852.

Lavigne M. B., Franklin, S.E., Hunt, E. R. J., 1996: Estimating stem maintenance respiration rates of dissimilar balsam fir stands. Tree Physiology 16: p. 687–695.

Lanwert, D., Dauzat, J. Früh, T., 1998: Water use of forest trees: a possibility of combining structures and functional models. Bayreuther Forum Ökologie, 52: p. 117–128.

Lanwert, D., Dzierzon, H., Sloboda, B., 2003: ELearning in den Forstwissenschaften am Beispiel einer lern-CD zur Erstellung von Ökologischen Modellen mit Lindenmayersystemen und der Modellschale GROGRA. Tagungsband der 15. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten, Freiburg 9.-10. Oktober 2003; Grüne Reihe, ISSN 1860-4064: S. 60-70.

Lanwert, D., Eilhard, M., Rusin, A., Sloboda, B.,2004: ElanSim Forester Package: Net based VRML software package for viewing and manipulating forest stands in the manner of an interactive computer game. Proceedings of the 4th International Workshop on Functional-Structural Plant Models, Montpellier, France: p. 413 & Poster.

Lembcke, Knapp, Dittmar, 2000: Ertragstafel für die Kiefer (Pinus sylvestris L.) im nordostdeutschen Tiefland. Herausgeber: K.-W. Lockow, Landesforstanstalt Eberswalde, 2000; abgerufen am 27.03.2007; http://www.lfe.brandenburg.de/cms/media.php/lbm1.a.4595.de/kieferet.pdf

LGS: XL Language Specification, 2006: The XL Language Specification (INCOMPLETE DRAFT), Lehrstuhl für Grafische Systeme, Brandenburgische Technische Universität Cottbus; abgerufen am 06.02.2007; http://www.grogra.de/xlspec

Lindenmayer, A., 1968: Mathematical models for cellular interaction in development, parts I and II. Journal of Theoretical Biology, 18: p. 280–315.

Lo, E., Wang, Z. M., Lechowicz, M., Messier, C., Nikimaa, E., Perttunen, J., Sievänen, R., 2001: Adaption of Lignum model for growth and light response in Jack pine. Forest Ecology and Management 150: p. 279-291.

Mäkelä, A., 1979: Simulointimalli valon vaikutuksesta männikön dynaamiseen kehitykseen (A simulation model for the impact of light on pine stand dynamics). Master of Science Thesis, Helsinki University of Technology, Department of Technical Physics, Espoo.

Mäkelä, A., 1982: Dynamics of biomass and light conditions in Scots pine stands. Helsinki University of Technology, System Theory Laboratory, Report B67.

McDowell, S. C. L., McDowell, N. G., Marshall, J. D., Hultine, K, 2000: Carbon and nitrogen allocation to male and female reproduction in rocky mountain douglas fir (Pseudotzuga menziesii Var. Glauca, Pinaceae). American Journal of Botany 87(4): p. 539–546.

McGaughy, R. J., 1997: Visualizing forest stand dynamics using the stand visualization system; Proceedings of the 1997 ACSM/ASPRS Annual Convention and Exposition; April 7-10, 1997, Seattle, WA. Bethesda, D: American Society for Photogrammetry and Remote Sensing, 4: p. 248-257.

Masinter, L. 1988: Returning Values from Forms: multipart/form-data, W3C RFC 2388; abgerufen am 05.01.2007; http://www.ietf.org/rfc/rfc2388.txt

Meining, S., v. Wilpert, K., Schröter, H., (2005): Waldzustandsbericht 2005 der Forstlichen Versuchs- und Forschungsanstalt Baden-Württemberg. Forstliche Versuchs- und Forschungsanstalt Baden-Württemberg (Hg.).

Monsi, M., Saeki, T., 1953: Über den Lichtfaktor in den Pflanzengesellschaften und seine Bedeutung für die Stoffproduktion. Japanese Journal of Botany, 14: S. 205-234

Muukkonen, P., Lehtonen, A., Mäkipää, R., Peltoniemi, M., 2004: Litter production in carbon budget modelling. Finnish Forest Research Institute (Metla). Research seminar on Carbon budget of Finnish forests 1920-2000; 23 March, 2004, Tieteidentalo, Helsinki; unveröffentlichte Präsentation.

Nagel, J., 1999: Konzeptionelle Überlegungen zum schrittweisen Aufbau eines wald-wachstumkundlichen Simulationssystems für Nordwestdeutschland. J. D. Sauerländers's Verlag, Frankfurt am Main.

Nagel, J., 2002: Das Open Source Entwicklungsmodell – eine Chance für Waldwachstumssimulatoren. Tagungsband der Jahrestagung der Sektion

Ertragskunde des Deutschen Verbandes Forstlicher Forschungsanstalten, Schwarzburg 13.–15. Mai 2002: S. 1-6.

Nagel, J., 2003: TreeGrOSS eine Java basierte Softwarekomponente zur Waldwachstumsmodellierung für Forschung, Lehre und Praxis. Tagungsband der 15. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten, Freiburg 9.-10. Oktober 2003; Grüne Reihe, ISSN 1860-4064: S. 33-37.

Nagel, J., 2006: CD-ForestTools2 -Forstliche Software Sammlung. Herausgeber: Prof. Dr. Jürgen Nagel, Selbstverlag J. Nagel, Göttingen; Beschreibung abgerufen am 04.04.2007; http://wwwuser.gwdg.de/~jnagel/cdft.html

Nagel, J., 2007: SimWald - Das Spiel zur Nachhaltigkeit; abgerufen am 14.04.2007; http://www.nw-fva.de/index.php?id=217

Nagel, J., Albert, M., Schmidt, M., 2002: Das waldbauliche Prognose- und Entscheidungsmodell BWINPro 6.1. Forst und Holz 57, (15/16): S. 486-493.

Nagel, J., Duda, H., Hansen, J., 2006: Forest Simulator BWINPro7. Forst und Holz 61 (10): S.427-429.

Nebel, E., Masinter, L., 1995: Form-based File Upload in HTML, W3C RFC 1867; abgerufen am 05.01.2007; http://www.ietf.org/rfc/rfc1867.txt

O'Neill, R. V., de Angelis, D. L., Weide, J. B., Allen, T. F. H., 1986: A hierachical concept of ecosystems. Princeton University Press**.**

OpenVrml: sourceforge.net Projektwebsite zu OpenVrml mit Sourcecode und Dokumentation; abgerufen am 05.01.2007; http://openvrml.sourceforge.net/

Ortlepp, P., 2006: Darstellung von Qualitätskriterien in vrml-basierten Waldszenen: Konzeption und Software. Masterarbeit zur Erlangung des MSc für Forstwissenschaften und Waldökologie, Fakultät für Forstwissenschaften und Waldökologie der Georg-August Universität Göttingen.

Penning de Vries, F. W. T., 1975: The cost of maintenance processes in plant cells. Annals of Botany 39: p. 77–92.

Penning de Vries, F. W. T., Brunsting, A. H. M., van Laar, H. H., 1974: Products, requirements and efficiency of biosynthesis: a quantitative approach. Journal of Theoretical Biololgy 45: p. 339–377.

Persistence of Vision Raytracer: Pov-Ray® registered Trademark by Persistence of Vision Raytracer Pty. Ltd.; abgerufen am 08.02.2007; http://www.povray.org/

Perttunen, J., Sievänen, R., Nikinmaa, E., Salminen, H., Saarenmaa, H., Väkevä, J., 1996: LIGNUM: A Tree Model Based on Simple Structural Units. Annals of Botany 77: p. 87-98.

Perttunen, J., Sievänen, R., Nikinmaa, E., 1998: LIGNUM: a model combining the structure and the functioning of trees. Ecological Modelling 108: p. 189-198

Pestov, S., et al.: jEdit Programmer's Text Editor; abgerufen am 08.02.2007; http://www.jedit.org/

Pfreundt, J., 1988: Modellierung der räumlichen Verteilung von Strahlung, Photosynthesekapazität und Produktion in einem Fichtenbestand und ihre Beziehung zur Bestandesstruktur. Dissertation, Georg August Universität Göttingen.

Pfreundt, J., Sloboda, B., 1996: The relation of local stand structure to photosynthetic capacity in a spruce stand: a model calculation. Lesnictví-Forestry 42: p. 149–160.

Piegl, L., Tiller, W., 1997: The NURBS book. 2nd edn. Springer, Berlin, Monographs in Visual Communication.

Polanyi, M., 1962: Personal Knowledge. Towards a Post-Critical Philosophy. Chicago Press, Chicago, London.

Polanyi, M., 1969: Knowing and Being. Essays edited by Marjorie Grene. Chicago Press, Chicago, London.

Pretzsch, H., 1990: Analyse der Bestandesstruktur als Grundlage für die Entwicklung eines Wachstumssimulators für Mischbestände. In: Tagungsberichte der Arbeitsgruppe Biometrie in der Ökologie 1: S. 51–71.

Pretzsch, H., 1992: Konzeption und Konstruktion von Wuchsmodellen für Rein- und Mischbestände. Forstliche Forschungsberichte München 115.

Pretzsch, H., 1993: Analyse und Reproduktion räumlicher Bestandesstrukturen. Versuche mit dem Strukturgenerator STRUGEN. Schriften aus der Forstlichen Fakultät der Universität Göttingen und der Niedersächsischen Forstlichen Versuchsanstalt 114, J. D. Sauerländers Verlag, Frankfurt.

Pretzsch, H., 2001: Modellierung des Waldwachstums. Parey Buchverlag, Berlin.

Pretzsch, H., 2002: Grundlagen der Waldwachstumsforschung. Parey Buchverlag, Berlin.

Pretzsch, H., Biber, P., Durský, J., 2002: The single tree-based stand simulator SILVA: construction, application and evaluation. Forest. Ecol. Management 162: p. 3-21.

Pretzsch, H., Kahn, M., 1998: Konzeption und Konstruktion des Wuchsmodells 2.2, methodische Grundlagen. Abschlussbericht Projekt W28, Lehrstuhl für Waldwachstumskunde der Technischen Universität München, Freising/Weihstephan, Teil 2.

Pretzsch, H., Seifert, S., 1999: Wissenschaftliche Visualisierung des Waldwachstums. AFZ/Der Wald 18: S. 960-962.

Prodan, M., 1965: Holzmesslehre. J.D. Sauerländers Verlag, Frankfurt.

Prusinkiewicz, P., Lindenmayer, A., 1990: The algorithmic beauty of plants. Springer, New York.

Rusin, A. O., 2004: Elan Sim Server Version 0.91 Sourcecode, Institut für Forstliche Biometrie und Informatik, Universität Göttingen; eingesehen am 29.01.2007.

Schober, R., 1987: Ertragstafeln wichtiger Baumarten bei verschiedener Durchforstung. 3. erweiterte. Auflage., J. D. Sauerländers Verlag, Frankfurt (a. Main).

Schulze, E. D., Fuchs, M. I., Fuchs, M., 1977: Spatial distribution of photosynthetic capacity and performance in a mountain spruce forest of northern Germany. I. Biomass distribution and daily carbon dioxide uptake in different crown layers. Oecologia, Berlin 29: S. 43-61.

Schwappach, A., 1889: Wachstum und Ertrag normaler Kiefernbestände in der norddeutschen Tiefebene. Verlag Julius Springer, Berlin.

Seifert, S., 2002: Visualisierung und Modellierung von Waldlandschaften. Tagungsband der 14. Tagung der Sektion Forstliche Biometrie und Informatik des Deutschen Verbandes Forstlicher Forschungsanstalten, Tharandt 3.-5. April 2002; Grüne Reihe, ISBN 961-6020-32-3: S. 131-139.

SGI, 2007: OpenGL®, Registered Trademark by Silicon Graphics; abgerufen am 08.02.2007; **http://www.opengl.org/**

Sievänen, R., 1993: A process-based model for the dimensional growth of even-aged stands. Scandinavian Journal of Forest Research 8: S. 28-48.

Sievänen, R., Nikinmaa, E., Nygren, P., Ozier-Lafontaine, H., Perttunen, J., Hakula, H., 2000: Components of functional-structural tree models. Annals of Forest Science 57: p. 399-412.

Sloboda, B., 1988: Representation und projection of diameter distribution in regarding change of rank in indicative experimental plots. In: Biometrische Modelle und Simulationstechniken bei Prozessen in forstlicher Forschung und Praxis. Schriften der Forstlichen Fakultät der Universität Göttingen und der Niedersächsischen Forstlichen Versuchsanstalt 90: S. 6-21.

Sloboda, B., Gaffrey, D., Matsumura, N., 1998: Representation of Tree Stem Taper Curves and their Dynamic, using a Linear Model and the Centroaffine Transformation. Journal of Forest Research 3: S. 67-74.

Sloboda, B., Pfreundt, J., 1989: Baum- und Bestandeswachstumsprozess. Ein systemanalytischer Ansatz mit Versuchsplanungskonsequenzen für die Durchforstung und Einzelbaumentwicklung. Bericht der Jahrestagung der Sektion Ertragskunde im Deutschen Verband Forstlicher Forschungsanstalten, Attendorn 1999: S. 17/1-17/25.

Sterba, H., Moder, HM., Monserud, R., 1995: PROGNAUS – Ein Waldwachstumssimulator für Rein- und Mischbestände. Österreichische Forstzeitung 5: S. 19-20.

Stickan, W., Gansert, D., Neemann, G., Rees, U., 1994: Modelling the influence of climatic variability on carbon and water budgets of beech saplings (Fagus sylvatica L.) based on field data. In: Ecological Modelling 75/76: S. 331-343.

Stockfors, J., 1998: Respiration Losses in Norway spruce: The effect of growh and nutrion. Doctoral thesis, Swedisch University of Agricultural Sciences, Upsala, veröffentlicht in Acta Universitatis Agriculturae Sueciae, Silvestria 20.

Stockfors, J., Linder, S. 1998: Effect of nitrogen on the seasonal course of growth and maintenance respiration in stems of Norway spruce trees. Tree Physiology, 18: p. 155-166.

Sun:Java: Java von Sun Microsystems, Inc.; abgerufen am 08.02.2007; http://www.sun.com/software/opensource/java/index.jsp

Sun: Java3D: API Spezifikationen für Java 3D von Sun Microsystems, Inc.; abgerufen am 18.03.2007; http://java.sun.com/products/java-media/3D/reference/api/index.html

Sun: Javadoc: Javadoc von Sun Microsystems, Inc.; abgerufen am 08.02.2007; http://java.sun.com/j2se/javadoc/

System in Motion: Coin3D: About Coin3D; abgerufen am 05.01.2007; http://www.coin3d.org/lib/about_coin3d/

System in Motion: Qt: Qt; abgerufen am 05.01.2007; http://www.coin3d.org/coin3d/lib/soqt/

System in Motion: Coin Free Edition: Coin3D Free Edition Licensing; abgerufen am 22.01.2007; http://www.coin3d.org/licensing/coin3d/gpl/

System in Motion: SoQt: SoQt; abgerufen am 05.01.2007; http://doc.coin3d.org/SoQt/

Tauer, M., 2006: Implementation eines Raytracers innerhalb der 3D-Plattform GroIMP. Studienarbeit an der Brandenburgischen Technischen Universität Cottbus, Lehrstuhl Graphische Systeme.

Trolltech: Qt: Qt Open Source Editions von Trolltech ASA, Norway; abgerufen am 05.01.2007; http://www.trolltech.com/developer/downloads/qt/

W3C: PNG, 2003: Portable Network Graphics (PNG) Specification (Second Edition), Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E), W3C Recommendation 10 November 2003; abgerufen am 05.01.2007; http://www.w3.org/TR/2003/REC-PNG-20031110/.

W3C: GIF, 1989: Graphics Interchange Format, W3C Specification; abgerufen am 05.01.2007; http://www.w3.org/Graphics/GIF/spec-gif89a.txt

W3C: HTML 4.01, 1999: HTML 4.01 Specification, W3C Recommendation 24 December 1999; abgerufen am 05.01.2007; http://www.w3.org/TR/1999/REC-html401-19991224/

Web3D: VRML, 1997: The Virtual Reality Modeling Language Specification, Version 2.0, ISO/IEC WD 14772; abgerufen am 05.01.2007; http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

Wenk, G., Antanaitis, V., Šmelko, S.., 1990: Waldertragslehre. Deutscher Landwirtschaftsverlag, Berlin.

Wenk, G., Römisch, K., Gerold, D., 1982: Die Grundbeziehung der neuen Fichtenertragstafel für das Mittelgebirge der DDR. Wissenschaftliche Zeitschrift der TU Dresden 31: S. 267-271.

Wiedemann, E., 1936: Die Fichte 1936. Verlag M. & H. Schaper

Wieser, G., Bahn, M., 2004: Seasonal and spatial variation of woody tissue respiration in a Pinus cembra tree at the alpine timberline in the central Austrian Alps. Trees, Band 18: S. 576–580.

Woodmann, J. N., 1971: Variation of net photosynthesis within the crown of a large forest-grown conifer. Photosynthetica, 5: p. 50-54.

Yergeau, F., 2003: UTF-8, a transformation format of ISO 10646, IETF RFC 3629, STD 63, Nov. 2003; abgerufen am 05.01.2007; http://www.ietf.org/rfc/rfc3629.txt

Zah, T., Kellomäki, S., Wang, K.-Y., Ryyppo, A., Ninisto, S., 1994: Seasonal and Annual Stem Respiration of Scots Pine Trees under Boreal Conditions. Annals of Botany 94: p. 889–896.

Zindler, A., 2005: Adaption von niedersächsischen forstlichen Inventur- und Standortdaten zur Prognoserechnung mit dem einzelbaumbasierten Simulator für Biodynamik SIBYLA, Bachelorarbeit im Studiengang Forstwissenschaften und Waldökologie an der Georg August Universität Göttingen.

8 Danksagung

Nach Abschluss einer solchen Aufgabe, wie sie diese Dissertation darstellt, erscheint es mir wünschenswert all denen zu danken, die mich zu diesem Punkt begleitet haben.

Zuerst und besonders möchte ich Professor Branislav Sloboda und Professor Winfried Kurth danken. Ihr Einfluss – der weit über diese Arbeit hinaus geht – hat meinem bisherigen Werdegang immer wieder entscheidende Impulse gegeben und mir neue Möglichkeiten eröffnet. Ich bedanke mich im besonderen Maße für ihr Vertrauen und ihre Unterstützung.

Ein weiterer besonderer Dank gilt Ole Kniemeyer für seine stets freundliche Hilfe bei meinen Fragen zu GroIMP und XL. Auch den anderen Mitgliedern der Arbeitsgruppe Pflanzenmodellierung, Gerhard-Buck-Sorlin und Reinhard Hemmerling, möchte ich herzlich danken.

Der von mir verehrten Margit Göbel danke ich sehr für ihre Hingabe an die neue deutsche Rechtschreibung, von der ich profitieren durfte.

Professor Jürgen Nagel bin ich sehr verbunden für die Bereitschaft als Gutachter dieser Arbeit zu dienen. Professor Marek Fabrika danke ich für seine Inspiration zu virtuellen Waldbeständen.

Marc Eilhard und Arkadius Rusin möchte ich dankend erwähnen, die mit großem Einsatz am Forester gearbeitet haben.

Meine aktuellen und früheren Kollegen am Institut für Forstliche Biometrie und Informatik möchte ich erwähnen, weil ich es immer geschätzt habe, Teil dieses Teams zu sein. Liebe/r Helge Dzierzon, Ulrike Hinüber, Reinhold Meyer, Jens Nieschulze, Felix Mader, Susanne Sprauer, Robert Nuske, Dominik Cullmann, Ole Kniemeyer, Michael Henke, Ilona Watteler-Spang, Kati Ahlert, Max Daenner, Joachim Saborowski, Peter Surovy, Rainer Schulz, Sarah Baumert, Lena Becker, Nikolas v. Lübke, Alexander Zindler, Christian Kleinschmidt und Peter Ortlep – behaltet die Freude am kritischen Dialog. Erwähnen möchte ich auch Thomas Früh, J.F. Barczi und Jean Dauzat.

Abschließend möchte ich mich noch bei drei weiteren Menschen bedanken, die einen wichtigen Einfluss auf meinen Lebensweg ausgeübt haben. Dies sind meine Eltern Gisela und Josef Lanwert und Forstamtsrat a.D. Ernst Rutsch. Ihnen widme ich diese Arbeit.

9 Index

9.1 Abbildungen

9.2 Formeln

(1) $$KKL_z = \sum_{\substack{i=1 \\ i \neq z}}^{n} (60 - wa_{i_z}) \cdot \frac{KQF_i}{KQF_z} \cdot TM_i$$ 12

(2) $$r_l = a \cdot E^b$$ 12

(3) $$r_s = c + E \cdot d$$ 12

(4) $$C66_z = \frac{\sum_{i=1}^{n} ks_i \, 66\% Kronenlänge_z}{A}$$; mit A als Bestandsfläche 15

(5) $$C66c = C66_{vor\_Durchforstung} - c66_{nach\_Durchforstung}$$ 15

(6) $$v_{stamm} = \pi \cdot \frac{d_{1,3}^{\,2}}{4} \cdot h \cdot f_{1,3}$$ und 64

(7) $$M_{stamm} = v_{stamm} \cdot cQuote \cdot \rho_{Holz}$$ 64

(8) $$f_{1,3} = k_1 + k_2 / d + k_3 / h + k_4 / d^2 + k_5 / (d \cdot h) + k_6 / (d \cdot h)^2 + k_7 \cdot \ln^2(d)$$ 65

(9) $$o_{stamm} = \int_0^h 2 \cdot \pi \cdot f(x) \cdot \sqrt{1 + f'(x)^2} \cdot dx$$ 65

(10) $$O_{stamm} = \pi \cdot d_{1,3} \cdot h \cdot \sqrt{f_{1,3}}$$ 65

(11) $$\text{Baumhöhe}_j = \text{Baumhöhe}_1 - ((j-1) * (\text{Baumhöhe}_1 / \text{Baumalter}))$$ 66

(12) $$\text{length(Seg)} = \text{Baumhöhe} / (\text{Baumalter} * i); \text{bis length(Seg)} \leq \text{maxlength}; i++$$ 67

(13) $$\rho_j(z) = M_j \cdot \frac{1}{B(p,q)} \cdot (z - a_j)^p \cdot (b_j - z)^q$$ 68

(14) $$B(p,q) = \int_{a_j}^{b_j} (z - a_j)^p (b_j - z)^q \, dz$$ 68

(15) $$m_{total}(Seg) = \sum_{j=1}^{\max} \rho_j(basis(Seg)) + \rho_j(top(Seg)) / 2 * length(Seg)$$ 68

(16) $$P(\vec{x}) = \int_{t_o}^{t_1} p(I(\vec{x}, t)) dt \;\; ; \vec{x} \in \Re^3$$ 68

(17) $P_0 = \int_{t_o}^{t_1} p(I_0(t))dt$ 69

(18) $q(\vec{x},t_0,t_1) := \frac{\int_{t_o}^{t_1} p(I(\vec{x},t))dt}{\int_{t_o}^{t_1} p(I_0(t))dt}; \vec{x} \in \Re^3$ oder (19) $q(\vec{x},t_0,t_1) := \frac{P(\vec{x})}{P_0}; \vec{x} \in \Re^3$ ...69

(20) $q(\vec{x},t_o,t_1) = F(M(\vec{x}))$, ...69

(21) $B(z) = \int_z^\infty \int_0^{2\pi} \int_0^{y\tan\alpha} \frac{p(y)}{l^2} r \,\mathrm{dr}\,\mathrm{d}\varphi\,\mathrm{dy}$ oder $B(z) = \int_z^\infty \int_0^{2\pi} \int_0^{y\tan\alpha} \frac{p(y)}{y^2+r^2} r \,\mathrm{dr}\,\mathrm{d}\varphi\,\mathrm{dy}$ 70

(22) $Y_{\min,i} = MAX(Kronenansatz\_relativ_i, z + d_i \cdot \tan\frac{\pi}{2} - \alpha)$, ...71

(23) $B_i(\vec{x}) = \int_{y_{\min,i}}^{h_i} \frac{p_i(y)}{\overline{l^2}} \mathrm{dy}$, ...71

(24) $l^2 = d_i^{\,2} + (y-z)^2$ 71

(25) $\overline{l^2} = l^2 = d^2$; wenn (y=z) und Nadelmasse in $\vec{y}$ konzentriert. 72

(26) $\overline{l^2} = \int_0^{2\pi}\int_0^R l^2 \frac{1}{\pi R^2} \mathrm{r}\,\mathrm{dr}\,\mathrm{d}\varphi = \frac{1}{\pi R^2}\int_0^{2\pi}\int_0^R (d^2 + r^2 - 2rd\cos\varphi)\,\mathrm{r}\,\mathrm{dr}\,\mathrm{d}\varphi$ 72

(27) $\overline{l^2} = \int_0^{2\pi} l^2 \frac{1}{2\pi} \mathrm{dr}\,\mathrm{d}\varphi = d^2 + R^2$ 72

(28) $\frac{1}{(\mathrm{y}-\mathrm{z})^2 + d^{\,2} + \frac{R^2}{cf}}$ oder $\frac{1}{\mathrm{l}^2 + \frac{R^2}{cf}}$ $1 \le cf \le 2$, ...72

(29) $B_0(\vec{x}) = \int_z^{h_0} \frac{p_0(y)}{(y-z)^2 + \frac{R_o^{\,2}}{cf}} \mathrm{dy} + \sum_{i=1}^{n} \int_{y_{\min,i}}^{h_i} \frac{p_i(y)}{(y-z)^2 + d_i^{\,2} + \frac{R_i^{\,2}}{cf}} \mathrm{dy}$ 72

(30) $q(\vec{x},t_o,t_1) = F(B(\vec{x}))$ bzw. $q(\vec{x}) = F(B(\vec{x}))$ 73

(31) $\mathrm{p}_0 = \mathrm{p}_0(\mathrm{t}_o, \mathrm{t}_1) = \int_{t_o}^{t_1} p(I_o(t))dt$ 73

(32) $$\mathrm{p}(\vec{x}) = \int_{t_o}^{t_1} p(I(\vec{x},t))dt = p_0 \cdot q(\vec{x}) = p_0 \cdot F(B(\vec{x}))$$ 73

(33) $$B_0(\vec{x}) = \int_{z_.}^{h_0} \frac{p_0(y) \cdot TM_o}{(y-z)^2 + \frac{R_o^{\,2}}{cf}}\,\mathrm{dy} + \sum_{i=1}^{n} \int_{y_{\min,i}}^{h_i} \frac{p_i(y) \cdot TM_n}{(y-z)^2 + d_i^{\,2} + \frac{R_i^{\,2}}{cf}}\,\mathrm{dy}$$ 73

(34) $$B(seg_0) = \sum_{i=1}^{n} \frac{M(seg_i) \cdot TM_{Baum(i)}}{Dist(\overrightarrow{segbasis_0}, \overrightarrow{segbasis_i}) + c_i} \text{ mit } c_i = \frac{R^2{}_{Baum(i)}}{cf_{Baum(i)}},$$ 74

(35) $$\mathrm{p}(Baum) = p_0 \cdot \sum_{Seg_I}^{n} F(B(Seg_i)) \sum_{jhg=0}^{n} M_{jhg} \cdot \mu_{jhg}$$ 76

(36) $$F(M(\vec{x})) = \frac{c}{1 + e^{-2 \cdot \frac{\log M(\vec{x}) - a}{b}}} + 1 - c$$; mit a = 2.3, b = -1.2 und c = 0.8. 76

(37) $$F(B(\vec{x})) = MAX(1 - a \cdot B(\vec{x})^b; 0)$$ 77

(38) $$F(B(\vec{x})) = a \cdot e^{-k \cdot B(\vec{x})}$$ 77

(39) $$R = R_{15} Q_{10}^{(T-15)/10}$$ 79

(40) $$T_{stamm} = 0{,}56 + 0{,}92 \cdot T_{luft}$$ mit $r^2 = 0.98$ 79

(41) $$R = \sum_{d=1}^{365} R_{15} Q_{10}^{((0{,}56+0{,}92 \cdot \overline{T_d})-15)/10)} \cdot 0{,}0010368$$ in (kgC m^{-2} j^{-1}). 79

(42) $$Nettoassimilate_j(i+1) = Nettoassimilate_j(i) - Verbrauch_j(i)$$ 80

(43) $$FWMasse_{j+1} = FWMasse_j \cdot FWÜberlebensrate + FWAllokationsrate \cdot Bruttophoto_j$$ 81

(44) $$NadelMasse_{j+1} = Nettoassimilate_j(3) \cdot MIN(maximalRate; \frac{minimalRate}{relativeProduktionseffizienz_j})$$ 81

(45) $$relativeProduktionseffizienz_j = \frac{Bruttophotosyntheseleistung_j}{Nadelmasse_j \cdot Produktionsrate_{unbeschattet}}$$ 81

(46) $$AstMasse_{j+1} = AstMasse_j \cdot Astüberlebensrate + NadelMasse_{j+1} \cdot Astwachstumsrate$$ 81

(47) $$GWneu = (Nettoassim._j(5) - GWMasse_j \cdot (1 - GWÜLrate)) \cdot GWAnteilsrate + (1 - GWÜLrate)$$ 82

(48) $GWMasse_{j+1} = GWMasse_j \cdot (1 - GWÜLsrate) + GWneu_j$ 82

(49) $\Delta h(t) = \Delta \bar{h}(t) + \Delta h_2(t) + \varepsilon(t)$ 82

(50) $\bar{h}(t) = 0{,}68 + 0{,}12 \cdot t + 0{,}0087 \cdot t^2; t < 25$
$\bar{h}(t) = 0 - 5{,}12 + 0{,}636 \cdot t + 0{,}00265 \cdot t^2; t \geq 25$ 82

(51) $\Delta \bar{h}(t) = \bar{h}(t+1) - \bar{h}(t)$ 82

(52) $\Delta h_2(t) = r_{H(t),\Delta h(t)} \cdot \frac{s\Delta h(t)}{\sigma h(t)} \cdot (h_i(t) - \bar{h}(t))$ 83

(53) $\varepsilon(t) = random(-s\Delta h(t), s\Delta h(t))$ 83

(54) $d13_i(t+1) = \sqrt{\frac{M_{stamm}(t+1)}{\frac{\pi}{4} \cdot h(t+1) \cdot f_{1,3} \cdot cQuote \cdot \rho_{Holz}}}$ 84

(55) $Kronendurchmesser(t+1) = \frac{Kronendurchmesser(t)}{t} \cdot (t+1)$ 84

9.3 Tabellen

10 Anhang

10.1 VRML-Szene

```
#VRML V2.0 utf8

# ------------------------------------------------------------------------
#
# ATTENTION: DONT CHANGE THESE VALUES !!!
#               IMPORTANT INFORMATION FOR ELANSIM FORESTER !
#
# ------------------------------------------------------------------------

Script {
    field SFString Name ""
    field SFInt32 Intervall 0
}

PROTO ShaftSystem [
    field SFInt32 Id 0
    field MFFloat aVec []
    field MFFloat bVec []
    field SFFloat BHDmin 0.50
]{
       DEF     ShaftSystem Script {
          field SFInt32 Id IS Id
          field MFFloat aVec IS aVec
          field MFFloat bVec IS bVec
          field SFFloat BHDmin IS BHDmin
    }
}

ShaftSystem    {
    Id 100,
    aVec  [0 0 0 0 0 0] #[ 18.354 -16.507 -23.066 -131.917 351.720 -197.613 ]
     bVec  [0 0 0 0 0 0]#[ 1.180 0.474 -3.6 12.381 -19.008 9.572 ]
}

ShaftSystem    {
    Id 101
    aVec [ 38.078 -444.489 2230.545 -4933.512 4876.312 -1764.531 ]
    bVec [ 0.384 15.462 -79.549 170.589 -166.447 60.493 ]
}

ShaftSystem    {
    Id 102
    aVec [ 48.495 -151.543 -45.411 659.228 -799.353 290.271 ]
    bVec [ 0.424 8.506 -29.883 48.218 -40.680 14.347 ]
}

ShaftSystem    {
    Id 103
    aVec [ 30.251 -277.133 1116.357 -2182.552 2070 -755.509 ]
    bVec [ 0.739 9.872 -45.724 89.447 -84.461 31.079 ]
}

ShaftSystem    {
    Id 104
```

```
    aVec [ 35.520 -230.925 1152.046 -2863.003 3181.882 -1274.832 ]
    bVec [ 0.406 7.379 -39.080 95.781 -108.051 44.566 ]
}



PROTO Species [
    field SFInt32 Id 0
    field SFString description ""
    field SFInt32 Shaftform -1
    field SFString CrownTexture ""
    field SFString StemTexture ""
]{
     DEF Species Script       {
          field SFInt32 Id IS Id
          field SFString description IS description
          field SFInt32 Shaftform IS Shaftform
          field SFString CrownTexture IS CrownTexture
          field SFString StemTexture IS StemTexture
     }
}

  # Douglasie Pseudotsuga menziesii
Species   {
    Id 0
    description"DG"
    Shaftform 103
    CrownTexture       "tree_3_right.gif"
    StemTexture        "S_S_Bark1.jpg"
}

# Kiefer pinus spec. (exkl. VJ)
 Species  {
    Id 1
    description"BO"
    Shaftform 104
    CrownTexture       "tree_7_right.gif"
    StemTexture        "wood_20.jpg"
}
  # Fichte picea spec. (excl. sp)
Species   {
    Id 2
    description"SM"
    Shaftform 104
    CrownTexture       "tree_7_right.gif"
    StemTexture        "wood_20.jpg"
}

# Strobe Pinus strobus
Species   {
    Id 3
    description"VJ"
    Shaftform 103
    CrownTexture       "tree_3_right.gif"
    StemTexture        "S_S_Bark1.jpg"
}
  # Laerche Larix spec.
 Species  {
    Id 4
```

```
    description"SC"
    Shaftform 104
    CrownTexture       "tree_7_right.gif"
    StemTexture        "wood_20.jpg"
}
  #Tannen abies       , tsuga, sequiadendron, thuja, chamecyparis, cryptomeria
Species   {
    Id 5
    description"JD"
    Shaftform 104
    CrownTexture       "tree_7_right.gif"
    StemTexture        "wood_20.jpg"
}
 # Eibe taxus baccata
Species   {
    Id 6
    description"TS"
    Shaftform 103
    CrownTexture       "tree_3_right.gif"
    StemTexture        "S_S_Bark1.jpg"
}

 # Stechfichte Picea pungens
Species   {
    Id 7
    description"SP"
    Shaftform 103
    CrownTexture       "tree_3_right.gif"
    StemTexture        "S_S_Bark1.jpg"
}


PROTO GGInfo [
    field SFString Name ""
    field SFString Scriptpath ""
    field SFString IP ""
    field SFInt32  minAge 50
    field SFInt32  maxAge 70
] {
    DEF   GGInfo Script {
          field SFString Name IS Name
          field SFString Scriptpath IS Scriptpath
          field SFString IP IS IP
          field SFInt32 minAge IS     minAge
          field SFInt32 maxAge IS     maxAge
    }
}


PROTO TREE [
    field SFInt32 ID 0
    field SFString Species ""
    field SFInt32 status 0
    field MFString labels [""]
    field MFString data [""]
    field SFVec3f position 0 0 0
    field MFString stem_texture "wood_20.jpg"
    field SFVec3f stem_size 0.7 1 1
    field MFString crown_texture "tree_7_right.gif"
    field MFFloat crown_values [1 1 1 1 1 1]
```

```
    field SFVec3f crown_size 1 1 1
    field SFVec3f crown_position 0 0 0
    field SFInt32 Shaftform -9999
]
{
    DEF Tree Transform {
    translationIS position
    children [
          DEF   Info Script {
                field SFInt32 ID IS   ID
                field SFString Species IS Species
                field SFInt32 Shaftform IS Shaftform
                field SFInt32 Status IS status
                field MFString Labels IS labels
                field MFString Data IS data
                field MFFloat CrownValues IS crown_values
          }

          DEF   Stem Transform {
                translation    0 0 0
                scale IS stem_size
                children [
                       DEF StemShape Shape {
                              geometry DEF StemGeometry Extrusion {
                                     spine [
                                            0 0 0, 0 .02 0, 0 .04 0, 0 .05 0,
                                            0 .0735648 0, 0 .110625 0,0 .159907 0,
                                            0 .19 0, 0 .33 0, 0 .77 0,.84208 0,
                                            0 .89824 0,0 .95893 0, 0 1 0
                                     ]

                                     scale [
                                            1 1, 0.901 0.901, 0.87 0.87,
                                             0.8545 0.8545, 0.8179 0.8179,
                                            0.768 0.768, 0.7215 0.7215,
                                            0.7 0.7, 0.6517 0.6517, 0.5 0.5,
                                            0.4464 0.4464, 0.3872 0.3872,
                                            0.2932 0.2932, .2 .2
                                     ]
                                     crossSection [
                                            .5 0, .46194 -.191342, .353553 -.353553,
                                            .191342 -.46194,0 -.5, -.191342 -.46194,
                                            -.353553 -.353553, -.46194 -.191342,
                                            -.5 0, -.46194 .191342, -.353553 .353553,
                                            -.191342 .46194, 0 .5, .191342 .46194,
                                            .353553 .353553, .46194 .191342,.5 0
                                     ]
                              }

                              appearance DEF StemAppearance Appearance {
                                     material Material {
                                            transparency 0
                                            diffuseColor 0.8 0.8 0.8
                                     }
                                     texture ImageTexture {
                                            url IS stem_texture
                                     }
                                     textureTransform TextureTransform {
                                             scale 2 7
```

```
                        }
                    }
                }
        ]
}

DEF Marker Transform {      #Die Markierung
        translation    0 0 0
        scale 1 1 1
        children [
                DEF     MarkerShape Shape {
                        geometry DEF MarkerGeometry Cylinder {
                                height 0.2
                                radius 1
                        }
                        appearance DEF MarkerAppearance Appearance {
                                material Material {
                                        diffuseColor 0 1 0
                                        transparency 1
                                }
                                texture ImageTexture { url "" }
                        }
                }
        ]
}

DEF   Crown Transform {             #Die Baumkrone
        scale IS crown_size
        translation    IS crown_position
        center 0 0 0
        children [

                Transform {
                        children [
                                DEF CrownShape Shape {
                                        appearance DEF CrownAppearance Appearance {
                                                textureTransform TextureTransform {
                                                        scale 1 1
                                                }
                                                texture  ImageTexture {
                                                        url IS crown_texture
                                                        repeatS FALSE
                                                        repeatT FALSE
                                                }
                                                material Material {
                                                        transparency 0
                                                        diffuseColor 0.8 0.8 0.8
                                                }
                                        }
                                        geometry DEF CrownGeometry IndexedFaceSet {
                                                coord Coordinate {
                                                        point   [
                                                        0 0 0,.5 0 0,  .5 .5 0,  0 .5 0
                                                        ]
                                                }
                                                coordIndex [
                                                        0 1 2 3 -1
                                                ]
```

```
                                texCoord TextureCoordinate {
                                point [0 0, 1 0, 1 1, 0 1]
                                }
                                solid FALSE
                        }
                }
        ]
}
Transform {
        rotation 0 1 0 1.046
        children [
                DEF CrownShape Shape {
                        appearance DEF CrownAppearance Appearance {
                                textureTransform TextureTransform {
                                        scale 1 1
                                }
                                texture ImageTexture  {
                                        url IS crown_texture
                                        repeatS FALSE
                                        repeatT FALSE
                                }
                                material Material {
                                        transparency 0
                                        diffuseColor 0.8 0.8 0.8
                                }
                        }
                        geometry DEF CrownGeometry IndexedFaceSet {
                                coord Coordinate {
                                        point [
                                        0 0 0, .5 0 0,.5 .5 0,0 .5 0
                                        ]
                                }
                                coordIndex [
                                        0 1 2 3 -1
                                ]
                                texCoord TextureCoordinate {
                                        point [0 0, 1 0, 1 1, 0 1]
                                }
                                solid FALSE
                        }
                }
        ]
}
Transform {
        rotation 0 1 0 2.092
        children [
                DEF CrownShape Shape {
                        appearance DEF CrownAppearance Appearance {
                                textureTransform TextureTransform {
                                        scale 1 1
                                }
                                texture ImageTexture  {
                                        url IS crown_texture
                                        repeatS FALSE
                                        repeatT FALSE
                                }
                                material Material {
                                        transparency 0
```

```
                                        diffuseColor 0.8 0.8 0.8
                                }
                        }
                        geometry DEF CrownGeometry IndexedFaceSet {
                                coord Coordinate {
                                        point [
                                        0 0 0, .5 0 0,.5 .5 0,0 .5 0
                                        ]
                                }
                                coordIndex [
                                        0 1 2 3 -1
                                ]
                                texCoord TextureCoordinate {
                                        point [0 0, 1 0, 1 1, 0 1]
                                }
                                solid FALSE
                        }
                }
        ]
}

Transform {
        rotation  0 1 0 3.14
        children [
                DEF CrownShape Shape {
                        appearance DEF CrownAppearance Appearance {
                                textureTransform TextureTransform {
                                        scale 1 1
                                }
                                texture ImageTexture  {
                                        url IS crown_texture
                                        repeatS FALSE
                                        repeatT FALSE
                                }
                                material Material {
                                        transparency 0
                                        diffuseColor 0.8 0.8 0.8
                                }
                        }
                        geometry DEF CrownGeometry IndexedFaceSet {
                                coord Coordinate {
                                        point [
                                        0 0 0, .5 0 0,.5 .5 0,0 .5 0
                                        ]
                                }
                                coordIndex [
                                        0 1 2 3 -1
                                ]
                                texCoord TextureCoordinate {
                                        point [0 0, 1 0, 1 1, 0 1]
                                }
                                solid FALSE
                        }
                }
        ]
}
Transform {
        rotation 0 1 0 4.152
```

```
                children [
                        DEF CrownShape Shape {
                                appearance DEF CrownAppearance Appearance {
                                        textureTransform  TextureTransform {
                                                scale 1 1
                                        }
                                        texture  ImageTexture {
                                                url IS crown_texture
                                                repeatS FALSE
                                                repeatT FALSE
                                        }
                                        material Material {
                                                transparency 0
                                                diffuseColor 0.8 0.8 0.8
                                        }
                                }
                                geometry DEF CrownGeometry IndexedFaceSet {
                                        coord Coordinate {
                                                point [
                                                0 0 0, .5 0 0,.5 .5 0,0 .5 0
                                                ]
                                        }
                                        coordIndex [
                                                0 1 2 3 -1
                                        ]
                                        texCoord TextureCoordinate {
                                                point [0 0, 1 0, 1 1, 0 1]
                                        }
                                        solid FALSE
                                }
                        }
                ]
        }

        Transform {
                rotation 0 1 0 5.166
                children [
                        DEF CrownShape Shape {
                                appearance DEF CrownAppearance Appearance {
                                        textureTransform TextureTransform {
                                                scale 1 1
                                        }
                                        texture ImageTexture  {
                                                url IS crown_texture
                                                repeatS FALSE
                                                repeatT FALSE
                                        }
                                        material Material {
                                                transparency 0
                                                diffuseColor 0.8 0.8 0.8
                                        }
                                }
                                geometry DEF CrownGeometry IndexedFaceSet {
                                        coord Coordinate {
                                                point [
                                                0 0 0, .5 0 0,.5 .5 0,0 .5 0
                                                ]
                                        }
```

```
                                                coordIndex [
                                                       0 1 2 3 -1
                                                ]
                                                texCoord TextureCoordinate {
                                                       point [0 0, 1 0, 1 1, 0 1]
                                                }
                                                solid FALSE
                                        }
                                }
                        ]
                }
        ]
    }
  ] }
}

# ---------------------------------------------------------------------------
# Information about the stand Growth Generator entry
#----------------------------------------------------------------------------

GGInfo { Name "GROIMP Model 2" Scriptpath "http://134.76.194.32:58080/open?model2.gs" IP
"http://134.76.194.32" minAge 0 maxAge 20 }
GGInfo { Name "Wachstumsgenerator2" Scriptpath "http://134.76.193.210/elan-
sim/empfangen4.cfm" IP "134.76.193.210" minAge 1 maxAge 30 }
GGInfo { Name "Wachstumsgenerator3" Scriptpath "http://134.76.193.215/elan-
sim/empfangen4.cfm" IP "134.76.193.215" minAge 2 maxAge 45 }

# ---------------------------------------------------------------------------
# Information about the stand :
# ---------------------------------------------------------------------------
WorldInfo {
    title "1. Waldbestand"
    info ["Dieser Waldbestand ist Testbestand"
          "des Instituts für Forstliche Biometrie und"
            "Informatik."
            "Ökophysiologisches Model 2"
            "Groimp Model2 via HTTP Server"
         ]
}
# ---------------------------------------------------------------------------
# Background Definition :
# ---------------------------------------------------------------------------
Background {
    skyColor [0.0 0.0 0.0 ,
              0.0 0.5 1.0 ,
                1.0 1.0 1.0 ]
    skyAngle [ 0.785, 1.571 ]
    groundColor[      0.0 0.0 0.0,
                      0.2 0.1 0.0,
                      0.3 0.2 0.1 ]
     groundAngle [ 0.785, 1.57 ]
}
# ---------------------------------------------------------------------------
#          Defaut Shaft System :
# ---------------------------------------------------------------------------
DEF DefaultSystem ShaftSystem {
    Id -1,
    aVec  [ 18.354 -16.507 -23.066 -131.917 351.720 -197.613 ]
     bVec [ 1.180 0.474 -3.6 12.381 -19.008 9.572 ]
```

```
}


 DEF BuildInfo Script {
    field SFBool useShaftSystem TRUE
    field SFBool adjustYPosition TRUE
}

# ------------------------------------------------------------------------
# Ground Definition :
# ------------------------------------------------------------------------
Shape {
      appearance Appearance {
        material Material{}
          texture      ImageTexture {
                repeatT TRUE
                repeatS TRUE
                url     "bewuchs1.jpg"

          }
          textureTransform TextureTransform {
                scale 30 30
          }
    }
geometry ElevationGrid {
    creaseAngle 1
     xDimension 12
     zDimension 12
     xSpacing 5.0
     zSpacing 5.0
     solid TRUE
     height [

0.406983175482,0.424073008815,0.414134442149,0.388052108815,0.331991308815,0.25252354214
9,0.168335708815,0.088074475482,0.027832775482,-0.0399670911847,-0.126311524518,0.0
0.910899608815,0.923269375482,0.907339208815,0.872610208815,0.804962908815,0.71618727548
2,0.627126808815,0.534455108815,0.445709975482,0.347382342149,0.241232042149,0.0
1.40913974215,1.40013707548,1.35649690882,1.30411857548,1.24263584215,1.17167230882,1.08
707757548,0.972585775482,0.845215575482,0.708639242149,0.577688342149,0.0
1.87229484215,1.81312127548,1.73405024215,1.67849807548,1.64188714215,1.59577507548,1.51
335727548,1.38279290882,1.22932000882,1.06060864215,0.910045108815,0.0
2.30472894215,2.19576084215,2.09470697548,2.04443434215,2.02203444215,1.98215814215,1.89
965894215,1.77087477548,1.60445224215,1.41513137548,1.26004104215,0.0
2.74039787548,2.60861280882,2.49553500882,2.43260777548,2.39457267548,2.34666007548,2.26
513747548,2.13837760882,1.96607537548,1.77676467548,1.62752354215,0.0
3.17223177548,3.03426180882,2.90770540882,2.81810580882,2.75365267548,2.69887364215,2.61
868360882,2.48687817548,2.31571527548,2.13777744215,1.98368400882,0.0
3.57767814215,3.43684970882,3.29732334215,3.18556797548,3.10376050882,3.04066034215,2.95
489577548,2.82066927548,2.65739007548,2.48371454215,2.30440340882,0.0
3.95429557548,3.81331454215,3.66861037548,3.54206414215,3.44233270882,3.36020990882,3.26
405917548,3.13426790882,2.98084587548,2.80471877548,2.59817580882,0.0
4.30642774215,4.16029940882,4.00980707548,3.87280347548,3.75711134215,3.65645420882,3.55
159580882,3.42669750882,3.27932814215,3.09759597548,2.86937534215,0.0
4.63516310882,4.47447787548,4.31197177548,4.17005487548,4.05287754215,3.94479444215,3.82
664057548,3.69091870882,3.53637770882,3.35000684215,3.11487907548,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
          ]
      }
}
```

```
# ------------------------------------------------------------------------
#   TREESTAND DEFINTION
# ------------------------------------------------------------------------
TREE { ID 1 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "1" "69" "10.8" "0.08" "SM"
"7.67" "1.9" "0" "SM" "Krone_gebrochen" "0" "sehr_astig" ] status 1 position 25.002
2.983 34.156 stem_size 0.04 10.8 0.04 crown_position 0 7.67 0 crown_size 1.9 6.26 1.9
Species "SM" }
TREE { ID 2 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "2" "69" "12.5" "0.09" "SM" "9"
"1.95" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 34.202 2.115
24.401 stem_size 0.045 12.5 0.045 crown_position 0 9 0 crown_size 1.95 7 1.95 Species
"SM" }
TREE { ID 3 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "3" "69" "14.2" "0.1" "SM"
"10.33" "2.01" "0" "SM" "normal" "0" "sehr_astig" ] status 1 position 22.93 2.911 32.74
stem_size 0.05 14.2 0.05 crown_position 0 10.33 0 crown_size 2.01 7.74 2.01 Species "SM"
}
TREE { ID 4 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "4" "69" "15.6" "0.11" "SM"
"11.35" "2.08" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 31.43
1.443 14.607 stem_size 0.055 15.6 0.055 crown_position 0 11.35 0 crown_size 2.08 8.5
2.08 Species "SM" }
TREE { ID 5 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "5" "69" "15.6" "0.11" "SM"
"11.35" "2.08" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 32.077
1.174 11.604 stem_size 0.055 15.6 0.055 crown_position 0 11.35 0 crown_size 2.08 8.5
2.08 Species "SM" }
TREE { ID 6 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "6" "69" "17" "0.12" "SM"
"12.36" "2.16" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 28.731
1.511 14.728 stem_size 0.06 17 0.06 crown_position 0 12.36 0 crown_size 2.16 9.28 2.16
Species "SM" }
TREE { ID 7 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "7" "69" "17" "0.12" "SM"
"12.36" "2.16" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 24.481
3.122 36.166 stem_size 0.06 17 0.06 crown_position 0 12.36 0 crown_size 2.16 9.28 2.16
Species "SM" }
TREE { ID 8 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "8" "69" "18.3" "0.13" "SM"
"13.27" "2.24" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 23.968
2.898 32.731 stem_size 0.065 18.3 0.065 crown_position 0 13.27 0 crown_size 2.24 10.06
2.24 Species "SM" }
TREE { ID 9 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "9" "69" "18.3" "0.13" "SM"
"13.27" "2.24" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 25.407
3.043 35.143 stem_size 0.065 18.3 0.065 crown_position 0 13.27 0 crown_size 2.24 10.06
2.24 Species "SM" }
TREE { ID 10 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "10" "69" "19.4" "0.14" "SM"
"13.97" "2.33" "0" "SM" "tot" "0" "sehr_astig" ] status 1 position 41.333 3.242 45.209
stem_size 0.07 19.4 0.07 crown_position 0 13.97 0 crown_size 2.33 10.86 2.33 Species
"SM" }
TREE { ID 65 labels [ "Baum" "Alter" "Hoehe" "D13" "Baumart" "Kronenansatz"
"Kronendurchmesser" "BiomasseC" "Qc_Tree_species" "Qc_Crown_structure"
"Qc_Crown_damage_class" "Qc_Stem_damage_class" ] data [ "65" "69" "27.2" "0.24" "SM"
"17.72" "3.28" "0" "SM" "normal" "0" "schoener_Nutzstamm" ] status 1 position 33.33
1.663 18.051 stem_size 0.12 27.2 0.12 crown_position 0 17.72 0 crown_size 3.28 18.96
3.28 Species "SM" }
```

… gekürzt um weitere 158 Bäume

10.2 Modell

10.2.1 Code definitionen.rgg

```
//---------------------------------------------------------------------
// Klasse Definitionen für Baumarten spezifischen Eigenschaften
//---------------------------------------------------------------------

class Definitionen {
   static const int test = 33;
// Numerischer Werte für Baumarten (Verwendung als Arrayindex)
// Angezeigt im Kommentar durch [ba]
   const int MAXBAUMARTEN = 3;
   const int Douglasie = 0;
   const int Kiefer = 1;
   const int FICHTE = 2;

// Baumartenkürzel der Bestandesszene für die einzelnen Baumarten [ba]
   const String [] BAUMARTENKUERZEL = {"DG", "BO", "SM"} ;

// Folgende Baumart wird verwendet, wenn kein passendes Kürzel gefunden wird
   const int DEFAULTBAUMART = 2;

// Numerische Werte der Kompartimente (Verwendung als Arrayindex)
// Angezeigt im Kommenta durch [ka]

   const int NADEL = 0;
   const int AST = 1;
   const int STAMM = 2;
   const int FEINWURZEL= 3;
   const int GROBWURZEL = 4;

// Werte für die Ausdehnung und Grundfläche des Bestandes
   const float BESTANDES_X_DIMENSION = 50.0f;
   const float BESTANDES_Y_DIMENSION = 50.0f;
   const float BESTANDES_FLAECHE = 2500f;

//---------------------------------------------------------------------------
// folgende Werte werden bei der Konstruktion der Baumobjekte übernommen
// und könnefür das Objekt individuell geändert werden
//---------------------------------------------------------------------------

// Definition der maximalen Standard-Segmentlänge [ba]
   const float[] MAX_SEGMENTLAENGE = {0.5f, 0.5f, 0.5f};

// Anzahl der Nadeljahrgänge (max. erlaubte Zahl ist 7: sonst Programmfehler) [ba]
   const int[] MAX_NADELJAHRGANG = {6,4,6};

//Startwerte für die Kompartimentverteilung [ba][ka]
   const float[][] KOMPARTIMENT_ANTEILE  = {
// Blatt, Ast, Stamm, Feinwurzel, Grobwurzel
      {0.11f, 0.16f, 0.52f, 0.03f, 0.18f},
      {0.06f, 0.13f, 0.62f, 0.02f, 0.17f},
      {0.11f, 0.16f, 0.52f, 0.03f, 0.18f}
   };
//Überlebensrate % des Nadeljahrganges i für das jahr i+1 [ba][jahrgang]
// beginnt bei Jahrgang 0 (neue Nadeln)
   const float[][] NADELJAHRGANG_UEBERLEBENSRATE  = {
      {1.0f, 1.0f, 1.0f, 0.9f, 0.85f, 0.75f, 0.0f},
      {0.9f, 0.78f, 0.71f, 0.6f, 0.0f, 0.0f, 0.0f},
         {1.0f, 1.0f, 1.0f, 0.9f, 0.85f, 0.75f, 0.0f}
   };
// geht von {neuer jahrgang 0, 1.jhg ... 6.jhg} für der maximale Nadeljahrgang
// der möglich ist muss die :>> Überlebensrate  0 sein)

// Faktor  zur Korrektur der Verteilung der Nadelmasse (1 für Konzentration auf der
// Kronenoberfläche) wird verwendet zur Korrektur der Gewichtung der beschattenden
// Nadelmasse mit Ihre Distanz vom Segment (r^2/x : ) [ba]
   const float[] NADELMASSE_VERTEILUNG_KORREKTURFAKTOR  =  {1.0f, 1.0f, 1.0f};

// Defaultwert wenn Kronendurchmesser nicht angegeben wird [ba]
   const float[] KRONENDURCHMESSERDEFAULTWERT = { 3.16f, 3.16f , 3.16f};

// relative Photosynthesekapazität der einzelnen Nadelahrgänge. Faktor für Jahrgang 1
// enthält auch die neuen Nadel des aktuellen Jahres ::> daher Faktor > 1 [ba][jahrgang]
```

```
   const float[][] NADELJAHRGANG_REL_PHOTO_KAPAZITAET  = {
      {2.0f, 0.65f, 0.6f, 0.55f, 0.5f, 0.45f, 0.0f},
      {2.0f, 0.65f, 0.6f, 0.55f, 0.5f, 0.45f, 0.0f},
      {2.0f, 0.65f, 0.6f, 0.55f, 0.5f, 0.45f, 0.0f}
   }; // Jahrgänge 1--- 7

// Koeffizienten zur Berechnung der unechten Formzahl k1-k7 Pretzsch 2001 [ba][k1-k7]
   const float [][] FORMZAHL_KOEFFITZIENTEN = {
      {0.3398f, 3.9392f, 0.0f, -16.5646f, -17.8652f, 117.1410f, 0.0f, },
      {0.5858f, -0.2693f, -0.4593f, 0.0f, 5.5060f, -5.1390f, -0.0116f},
      {0.5073f, 1.5685f, -0.4237f, -9.1576f, 3.5585f, 17.3395f, -0.0068f}
   };

// Parameter der Altershöhenkurve (Kombination zweier quadtatischer Gleichungen
// a + b*t + c*t^2. Zelle 0 enthält den Schwellenwert: Zellen 2-7 enthalten 2
// Parametersätze a, b, c. Bei Alter >= Schwellenwert: wird Parametersatz 2 (Zellen 5-7)
// genommen sonst Parametersatz 1 (Zelle 2-4) [ba][parameter 0, 1-6]
   const float[][] MITTELSTAMM_AlTERS_HOEHENKURVE  = {
         {25.0f, 0.68f, 0.12f, 0.0087f, -5.12f, 0.636f, -0.00265f},
         {25.0f, 0.68f, 0.12f, 0.0087f, -5.12f, 0.636f, -0.00265f},
         {25.0f, 0.68f, 0.12f, 0.0087f, -5.12f, 0.636f, -0.00265f}
      };

// Lichttransmissionskoeffizient nach Pretzsch 2001 zum Ausgleich der
// Beschattungswirkung von unterschiedlichen Baumarten. bei Faktor 1 bleibt die
// Beschattungswirkung gewichtet [ba]
   const float[] LICHT_TRANSMISSIONS_KOEFFIZIENT = { 1.0f, 0.2f, 0.8 };

// Numerische Werte für Wahl der Photosynthesefunktion
   const int F_FUNKTION_NACH_BEER_LAMBERT = 0;
   const int F_FUNKTION_NACH_PFREUNDT = 1;
// const int F_FUNKTION_NACH_MAEKELA = 2;

// Setzt per Default die verwendete F-Funktion für die Baumarten. [ba]
   const int [] F_FUNKTION_BAUMARTEN_DEFAULT = {F_FUNKTION_NACH_BEER_LAMBERT,
      F_FUNKTION_NACH_BEER_LAMBERT, F_FUNKTION_NACH_BEER_LAMBERT };

// Parameter der BEER-LAMBERT Funktion zur Reduktion der potenziellen Photosynthetischen
// Kapazität anhand der beschattenden Nadelmasse [ba]
   const float[] BEER_LAMBERT_COEFF_A = {1.0f, 1.0f, 1.0f};
   const float[] BEER_LAMBERT_COEFF_B = {0.03f, 0.03f, 0.03f};

// Parameter der PFREUNDT Funktion zur Reduktion der potenziellen Photosynthetischen
// Kapazität anhand der beschattenden Nadelmasse [ba]
   const float[] PFREUNDT_COEFF_A = {0.1f, 0.1f, 0.1f};
   const float[] PFREUNDT_COEFF_B = {1.1f, 1.1f, 1.1f};

// Mäkela noch nicht implemntiert  [ba]
   const float[] MAEKELA_COEFF_A = {2.3f, 2.3f, 2.3f};
   const float[] MAEKELA_COEFF_B = {-1.2f, -1.2f, -1.2f};
   const float[] MAEKELA_COEFF_C = {0.8f, 0.8f, 0.8f};

// Photosyntheseleistung unbeschattet [ba]
   const float[] PRODUKTIPONSRATE_UNBESCHSATTET = {2.8f, 4.6f, 2.8f};

// Respirationsparameter [ba]
   const float[] NADEL_RESPIRATION_RATE = {0.5f, 0.2f, 0.5f};

// Respirationsraten auf Basis der BHD ;Führt zu stark überhöhten Werten;
// für Holzteile (Wurzel, Stamm, Ast) [ba]
   const float[] RESPIRATION_COEFF_A = {1.6f, 1.6f, 1.6f};

// Respirationsraten für alternative Berechnungsmethode[ba]
// kgC / m^2 Stammoberflaeche:
   const float[] STAMM_RESPIRATIONRATE_METHODE_2 = {0.24211f, 0.24211f, 0.24211} ;
   const float[] AST_RESPIRATIONRATE_METHODE_2 = {0.048f, 0.048f, 0.048f} ;
   const float[] WURZEL_RESPIRATIONRATE_METHODE_2 = {0.24f, 0.24f, 0.24f} ;

// % Anteil des Verbrauchs von Assimilaten zur Energiegewinnung des Wachstums
   const float[] WACHSTUMSRESPIRATION_RATE = {0.3f, 0.3f, 0.3f };

// Relation zur Nettophotosynthese für Feinwurzelwachstum [ba]
   const float[] FEIN_WURZEL_ALLOCATIONS_RATE = {0.03f, 0.03f, 0.03f };
   const float[] FEIN_WURZEL_UEBERLEBENS_RATE = {0.33f, 0.33f, 0.33f };

   const float[] NADEL_WACHSTUM_EFFIZINZANPASSUNG_COEFF_A={0.8f, 0.8f, 0.8f};
   const float[] NADEL_WACHSTUM_EFFIZINZANPASSUNG_COEFF_B={-0.1f,-0.1f,-0.1f};
```

```
   const float[] NADEL_WACHSTUM_ASSIMILATE_MINIMALRATE = {0.3f, 0.3f, 0.2f};
   const float[] NADEL_WACHSTUM_ASSIMILATE_MAXIMALRATE = {0.7f, 0.7f, 0.7f};
   const float[] NADEL_WACHSTUM_ASSIMILATE_EINFLUSS_FAKTOR_EXP = {1.0f,1.0f,2.0f};

//Relation des Astwachstums proportional zur neuen Nadelmasse[ba]
   const float[] AST_WACHSTUM_RATE = {0.4f, 0.4f, 0.4f };

//Überlebensrate der vorjährigen Nadelmasse [ba]
   const float[] AST_UEBERLEBENS_RATE = {0.975f, 0.975f, 0.975f };

//Relation des Grobwurzelwachstums proportional zum Assimilatespeicher [ba]
   const float[] GROB_WURZEL_WACHSTUM_RATE = {0.25f, 0.25f, 0.25f };

//Überlebensrate der vorjährigen Grobwurzelmasse [ba]
   const float[] GROB_WURZEL_UEBERLEBENSRATE_RATE = {0.975f, 0.975f, 0.975f };

//Korrelationskoeffizienz für den Zusammenhang von BaumHöhe und Höhenzuwachs [ba]
   const float[] HOEHENZUWACHS_CORR_COEFF = {0.3f, 0.3f, 0.3f };

// Variationskoeffizienz zu Schätzung von Sigma Delta Hoehe zum Zeitpunkt t [ba]
   const float[] HOEHENZUWACHS_VARIATIONS_COEFF = {0.2f, 0.2f, 0.2f };

// Dichte der Trocken-Biomasse in kg/m3 [ba]
   const float[] STAMMDICHTE = {450f, 450f, 450f };

// Anteil C an Trockenmasse
   const float C_QUOTE = 0.5f;

// Kronenform nach Pretsch 2001; Parameter { i0:Länge Sonnenkrone = Länge Krone * x;
// maximaler Kronenasatz /( Länge Sonenkrone^x), b, Radius Kronenansatz =
//  maximaler rRadius * x }
   const float[][] KRONENFORM_PARAMETER  = {
      {0.66f, 1.0f, 1.0f, 0.5f},
      {0.64f, 0.5f, 0.5f, 0.5f},
      {0.66f, 1.0f, 1.0f, 0.5f}
   };

//---------------------------------------------------------------------
//            Methoden
//---------------------------------------------------------------------

public static float altersHoehenkurveMittestamm(float alter, int art)
{
// berechnet die Höhe des Bestandesmitelstammes in Abhängigkeit des Alters
   float a = 0.0f, b= 0.0f, c= 0.0f;
   if (alter < MITTELSTAMM_AlTERS_HOEHENKURVE[art][0]){
      a = MITTELSTAMM_AlTERS_HOEHENKURVE[art][1];
      b= MITTELSTAMM_AlTERS_HOEHENKURVE[art][2];
      c= MITTELSTAMM_AlTERS_HOEHENKURVE[art][3];}
   else {
      a = MITTELSTAMM_AlTERS_HOEHENKURVE[art][4];
      b= MITTELSTAMM_AlTERS_HOEHENKURVE[art][5];
      c= MITTELSTAMM_AlTERS_HOEHENKURVE[art][6];
   }
   return a + b * alter + c * alter * alter;
}

static int getBaumartenIndex(String buf)
// ermttelt Modellindex der Baumart anhand des Schlüssels der Forester-Description
{
   int retValue = DEFAULTBAUMART;
   for (int i = 0; i < MAXBAUMARTEN; i++)
   {
      if(BAUMARTENKUERZEL[i].equalsIgnoreCase(buf))
         retValue = i;
      }
      return retValue;
   }
}
// Ende definitionen.rgg
```

10.2.2 Code wald.rgg

```
import de.grogra.ext.cpfg.*;
import static java.lang.Math.*;
import java.io.*;
import de.grogra.imp.net.*;
import de.grogra.util.*;
import de.grogra.pf.registry.*;
import java.util.regex.*;
import java.util.*;
import forester.http.formdata.*;
import forester.description.*;
import forester.groimp.*;
import de.grogra.pf.ui.*;
import de.grogra.math.*;

//------------------------------------------------------------------------------
//
//        Modulbeschreibungen der wald.rgg Datei
//
//------------------------------------------------------------------------------
// Hilfsmodul als Startobkect des Bestandes

module Nullpunkt extends Null;

// Hauptelement der Simulation
module Bestand(Point3d referenzpunkt, float xseite, float yseite, boolean aktiv) extends
   Null(){setTransform(referenzpunkt);}

//Begrenzungen für die Bestandesfläche
module Pfosten() extends Null ==> F(0.5f, -1.0f, 12) F(0.5f, -1.0f, 15) F(0.5f, -1.0f,
   11) F(0.5f, -1.0f, 15) F(0.5f, -1.0f, 12) F(0.5f, -1.0f, 15);

// Räumlicher Nullpunkt eines Bestandes
module Referenz;

// Modul Baum definiert die Baum spezifischen Werte und wird als Sphere dargestellt
module Baum(int bnr, super.baumAlter , super.baumHoehe, super.d13,
          super.baumArt, super.kronenAnsatz)
         extends Baumklasse(baumAlter, baumHoehe, d13, baumArt, kronenAnsatz);

// Bestandteile des oberirdischen Teil des Baumes.
module Segment(int bnr, int snr, super.length, float base) extends
   Segmentklasse(length);

// Hilfsobjekt zur Visualisierung der Kronen
module Krone(int bnr, float ansatz, float hoehe);


const int BAUMGRUNDFARBE = 0xCCCCCC; const int F_BAUMGRUNDFARBE = 7;

// Farbwerte
const int UNBENADELT = 0xCCCCCC;    const int F_UNBENADELT = 7;
const int BENADELT = 0x009900;   const int F_BENADELT = 10;
const int BESCHATTET = 0x00FF00;    const int F_BESCHATTET = 2;

// Farbwert für nicht aktive Bäume des Randgürtels
const int NICHT_AKTIV = 0xFF0000;

const DatasetRef nadeldichte = new DatasetRef ("nadeldichte");
const MaterialRef cronemat = material ("Lambert");

// Voreinstellung der Länge der Simulationsperiode in Jahren
int simulationsPeriode = 5;
int beobachteterNadeljahrgang = 0;

// Parameter für die per HTTP gestartete Simulation
transient ForesterDescription foresterDescriptionIn = null;
```

```
//-------------------------------------------------------------------------
//                    Startup durch eine ForesterSzene per HTTP Server
//-------------------------------------------------------------------------

// Startmethode beim HTTP-Server Betrieb.

protected void startup ()
{
   super.startup ();
   HttpResponse resp = HttpResponse.get (workbench());
   System.out.println (resp);
   if (resp != null)
   {
      runLater (resp);
   }
}



protected void run (Object info)
// Methode run wird von runLater aus startup aufgerufen.
// Findet nur Verwendung wenn die Simulation über den HTTP-Server gestartet wird.
{
   String modelOutputDir = ForesterUtils.leseHttpServerBasisDirectoryAusPreferences()
      + "\\" + "modeloutput\\";

   // Auslesen der Formulardaten
   HttpMultipartFormData n1 = httpstartup.verarbeiteHttpInput(info);
   // Auslesen der Foresterdescription

   foresterDescriptionIn = httpstartup.readForesterDescriptionFormfield(n1);
   // die voreingestellte Länge der Simulationsperiode wird ggf.durch Formulardaten
   überschrieben
   String filename = httpstartup.createAnswer(info, n1,foresterDescriptionIn);

   simulationsPeriode = httpstartup.readSimulationPeriode(n1, "forecastperiode",
       simulationsPeriode);
   init ();
   if(foresterDescriptionIn != null){
      // Initalisieren des Modells
      ersterSchritt();
      //Simulationsläufe
      for (apply (simulationsPeriode)) kompletterJahresDurchlauf();
      // Schreiben der Struktur
      abspeichern(foresterDescriptionIn , ( modelOutputDir + filename) );
   }
   // schießt das Programm automatisch
   closeWorkbench();
}

//-------------------------------------------------------------------------
//                       Startup manuell mit einer ForesterSzene
//-------------------------------------------------------------------------

// Methode runbyHand() erlaubt die manuelle Verwendung der Simulation. Es erfolgt ein
   kompletter Simulationsdurchlauf. Die Methode kann durch das Menü aufgerufen werden

public void runByHand()
// Methode runbyHand() erlaubt die manuelle Verwendung der Simulation. Es erfolgt ein
   kompletter Simulationsdurchlauf. Die Methode kann durch das Menü aufgerufen werden
{
   init ();
   manualStart();
   if(foresterDescriptionIn != null){
      ersterSchritt();
      simulationsPeriode = ForesterUtils.getParameterFromDialog("Länge der Simulation",
         "Bittte geben Sie Länge der Simulation in Jahren an:",
         Integer.toString(simulationsPeriode), simulationsPeriode);
      for (apply (simulationsPeriode)) kompletterJahresDurchlauf();
      manualEnd();
   }
}

public void runByHandMitSpeichern(){
// Methode runbyHand() erlaubt die manuelle Verwendung der Simulation. Es erfolgt ein
   kompletter Simulationsdurchlauf. Methode kann durch das Menü aufgerufen werden
```

```
   String modelOutputDir = ForesterUtils.leseHttpServerBasisDirectoryAusPreferences() +
      "\\" + "modeloutput\\";
   init();
   manualStart();
   if(foresterDescriptionIn != null){
      String filename = ForesterUtils.getParameterFromDialog("Ausgabename", "Bitte den
         namen für das Ausagebfile angeben:", "ausgabe_", "ausgabe_");
      ersterSchritt();
      simulationsPeriode = ForesterUtils.getParameterFromDialog("Länge der Simulation",
         "Bittte geben Sie Länge der Simulation in Jahren an:",
         Integer.toString(simulationsPeriode), simulationsPeriode);
      int i = 1;
      for (apply (simulationsPeriode)){
         kompletterJahresDurchlauf();
         abspeichern(foresterDescriptionIn , ( modelOutputDir + i + "_" + filename) );
         i++;
      }
   }
}


// Der Ablauf kann auch manuell schrittweise gesteuert werden durch:
//    1. manualStart()
//    2. ersterSchritt()
//    3. kompletterDurchlauf() // kann beliebig wiederholt werden
//    4. manualEnd()

public void manualStart()
// Die Methode erlaubt den manuelle Verwendung einer Forester Bestandesszene
{
   Workbench wrk = workbench();
   FileChooserResult fcr = wrk.chooseFile("description" ,null, Window.OPEN_FILE, true );
   String fileContent = ForesterUtils.readFileInString(fcr.file.toString());
   String fileName = ForesterUtils.extractFileName(fcr.file.toString());
   foresterDescriptionIn =  new ForesterDescription(fileContent, fileName);
}

public void manualEnd()
// Die Methode schließt die manuelle Ausführung einer Simulation ab und erlaubt das
   Abspeichern der Description Datei
{
   Workbench wrk = workbench();
   FileChooserResult fcr = wrk.chooseFile("description" , null, Window.SAVE_FILE, false
   );
   abspeichern(foresterDescriptionIn, fcr.file.toString());
}

//-------------------------------------------------------------------------------
//                   Steuerung der Simulation auf Bestandesebene
//-------------------------------------------------------------------------------

protected void init()
// Startmethode
{
   setSeed(0);
   Point3d referenzpunkt;
   [
      {referenzpunkt = new Point3d( Definitionen.BESTANDES_X_DIMENSION / -2 ,
         Definitionen.BESTANDES_Y_DIMENSION / -2,0);}
      Axiom ==> Nullpunkt [ Bestand(referenzpunkt, Definitionen.BESTANDES_X_DIMENSION,
         Definitionen.BESTANDES_Y_DIMENSION, true) ];
   ]
}

public void ersterSchritt()
//Startfunktion führt alle Schritte zum Zeichnen der Bäume aus
{
   // zur Sicherheit werden bestehende Bäume gelöscht
   for (apply(1)) deleteBestandesReferenzpunkt();
   // Baumobjekte werden erzeugt und initialisiert
   for (apply(1)) bauemePflanzen(foresterDescriptionIn);
   for (apply(1)) setzeBaumGrundfarbe(BAUMGRUNDFARBE);
   for (apply(1)) erzeugeBaumSegmente();
   for (apply(1)) initialisiereSegmenteFuerJahresDurchlauf(); ;
}
```

```
public  void kompletterJahresDurchlauf()
// Methode zur Durchführung eines kompletten Jahresdurchlaufs der Simulation
{
   long t1 = System.currentTimeMillis();
   long t2 = 0; long t3 = 0;

   println("Kompletter Jahreszyklus beginnt:");
   for (apply(1)) kalkuliereBeschattung();
      t2 = System.currentTimeMillis() -t1 -t3 ; t3 += t2;
       println("                    Zeit: " +  ( t2  ));
   for (apply(1)) kalkuliereNettoPhotosynthese() ;
      t2 = System.currentTimeMillis() -t1 -t3; t3 += t2;
      println("                    Zeit: " +  ( t2 ));
   for (apply(1)) verteileAssimilate();
      t2 += System.currentTimeMillis() -t1 -t3; t3 += t2;
       println("                    Zeit: " +  ( t2 ));
   for (apply(1)) wachse();
      t2 += System.currentTimeMillis() -t1 -t3; t3 += t2;
      println("                    Zeit: " +  ( t2  ));
   println(" Kompletter Jahreszyklus endet. Rechenzeit: " + t3);
}


public  void reinitialisiereNachWerteAenderungBeiHand()
// Diese Funktion muss ausgeführt werden, wenn die Baumnadelmassen manuell geändert
   wurden, Die Segnmenteigenschaften werden aktualisiert.
{
   for ((* b:Baum *)) {
      b[nadelMasseG] = b.addiereNadelJahrgangMassen();
   }
    for(apply(1)) initialisiereSegmenteFuerJahresDurchlauf();
}

public void kalkuliereBeschattung()
// Steuerung der Beschattungsberechnung
{
   for(apply(1))erzeugeRandbestaendeAlsKopieDesHauptbestandes();
   for(apply(1)) kalkuliereExterneBeschattung(60);
   for(apply(1))deleteRandBestaende();
   for(apply(1)) kalkuliereSelbstBeschattung();
   for((* b:Baum *)) {
      addiereGesamtBaumBeschattung(b.bnr);
   }
   println("  -1 Beschattung berechnet: ");
}

public  void kalkuliereNettoPhotosynthese()
//Steuerung der Nettophotosyntheseberechnung
{
    for(apply(1)) kalkuliereBaumPhotosynthese();
    for(apply(1)) kalkuliereRespiration();
    for(apply(1)) addiereNettoPhotosynthese();
    println("   -2 Photosynthese berechnet:   ");
}

public  void verteileAssimilate()
//Steuerung der Assimilatverteilung
{
   for(apply(1)) verteileAssimilateAufKompartimente();
   println("   -3 Assimilate verteilt:   ");
}

public  void wachse()
//Steuerung des Wachstums
{
   for(apply(1)) berechneHoehenwachstumsBaumdaten();
   for(apply(1)) anfuegeSegmente();
   for(apply(1)) ermittleNeuenKronenAnsatz();
   println("   -4 Wachstum abgeschlossen:   ");
   for(apply(1))  uebertrageBaumdatenInsNaechsteJahr();
   for(apply(1))  initialisiereSegmenteFuerJahresDurchlauf();
   println("   -5 Neuinitialisierung abgeschlossen:   ");
}
```

```
//------------------------------------------------------------------------
//                        Anlegen des Bestandes
//------------------------------------------------------------------------


private void deleteBestandesReferenzpunkt()
// Sicherheitsfunktion zum Löschen bereits bestehender Bäume
[
   Referenz ==>> ;

];


private void bauemePflanzen(ForesterDescription f1)
// Hauptmethode zum Anlegen des Bestandesonjektes mit allen Bäumen
[
   {

   }
   best:Bestand(referenzpunkt, xseite, yseite, aktiv) ==> best [ Referenz
   // Pfosten zur bgrenzungsvisualisierung warden angelegt
      [ p1:Pfosten]
      { p1.setTransform(0, 0, 0);}
      [ p2:Pfosten]
      { p2.setTransform(0 , yseite, 0);}
      [ p3:Pfosten]
      { p3.setTransform(xseite, yseite, 0);}
      [ p4:Pfosten]
      { p4.setTransform(xseite, 0 , 0);}

      for(int i = 0; i < f1.getNumberOfTrees(); i++)(
         {
         float x =   ((ForesterTree)(f1.foresterTrees.get(i))).position[0];
         float y =   ((ForesterTree)(f1.foresterTrees.get(i))).position[2];
         float z =   ((ForesterTree)(f1.foresterTrees.get(i))).position[1];
         float bhd = ((ForesterTree)(f1.foresterTrees.get(i))).stemSize[0];
         float ca =  ((ForesterTree)(f1.foresterTrees.get(i))).crownPosition[1];

         // Achtung Kronenradius und Kronenlänge werden im Forester Client Version 0.90
   2 fach überhöht angegeben
         // Die Werte müssen korrigiert werden (*0.5f)
         float ch = (((ForesterTree)(f1.foresterTrees.get(i))).crownSize[1]) *0.5f;
         float cd = (((ForesterTree)(f1.foresterTrees.get(i))).crownSize[0])* 2.0f*0.5f;
         float h =   ca + ch;
         int alter = 1;
         int id = ((ForesterTree)(f1.foresterTrees.get(i))).id;
         int art ;
         float biom = 0;
         int status = ((ForesterTree)(f1.foresterTrees.get(i))).status;

         // Dursuchen des Baum-Data Bereichs nach relevanten Baumdaten
         // einige der hier angegebenen Werte können auch aus den Baum-Lageparametern
   (position, ste,_size, crown_position, corwn_size ermittelt werden
         // /( Value Overwrite) gibt an wenn die bereits ermittelten Werte überschrieben
   werden
         String buf;
   // Alter des Baums
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Alter");
            if( buf != null ){ alter = Integer.valueOf(buf);}
   // Nummer des Baums
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Baum");
         if( buf != null){ id = Integer.valueOf(buf);}
// Baumarten gemäß der Definitionenklasse
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Baumart");
         art = Definitionen.getBaumartenIndex(buf);
// Hoehe des Baums ( Value Overwrite)
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Hoehe");
         if( buf != null){ h = Float.valueOf(buf);}
   // BHD /( Value Overwrite)
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("D13");
         if( buf != null){ bhd = Float.valueOf(buf);}
   // Kronenansatz - Höhe am Stamm ( Value Overwrite)
         buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Kronenansatz");
         if( buf != null){ ca = Float.valueOf(buf);}
   // Kronendurchmesser - Höhe am Stamm /( Value Overwrite)
      buf = ((ForesterTree)(f1.foresterTrees.get(i))).getDataValue("Kronendurchmesser");
```

```
         if( buf != null){ cd = Float.valueOf(buf);}
         }
         if (status < 3 && status >= 0)(
            [ b21:Baum(id, alter, h, bhd, art, ca  )]
          {
            b21.setTransform (x, y, z); b21[maximalerKronenRadius] = cd /2.0f;
            b21.descriptionIndex = i; b21.id = id;
          }
         )
      )
   ];
]


//-------------------------------------------------------------------------
//                      Abspeichern des Bestandes
//-------------------------------------------------------------------------

private  void aktualisiereForesterDescription(ForesterDescription f1)
// Aktualisieren der Forester Description mit den Simulationsergebnissen
[
   {
   for((* b:Baum *)){
      if (b.descriptionIndex > 0){
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("Alter", Integer.toString(b.baumAlter));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("Hoehe", Float.toString(b.baumHoehe));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("D13", Float.toString(b.d13));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("Kronenansatz", Float.toString(b.kronenAnsatz));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("Kronendurchmesser",
            Float.toString(b.maximalerKronenRadius*2.0f));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("Kronendurchmesser",
            Float.toString(b.maximalerKronenRadius*2.0f));
         ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex)))
            .setDataValue("BiomasseC", Float.toString(b.cMasseGesamt));

         if (b.nadelMasseG <= 0.0f) {
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).status = 3; }
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).stemSize[0]= (b.d13/2.0f);
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).stemSize[2]= (b.d13/2.0f);
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).stemSize[1]= b.baumHoehe;
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).crownPosition[1] =
   b.kronenAnsatz;
   // Multiplikation mit 2 erfolgt zu Korrektur eines Darstellungsfehlers der Kronen im
   Virtual Forester Client Version 0.90
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).crownSize[0]=
       b.maximalerKronenRadius*2.0f;
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).crownSize[1]=
      (b.baumHoehe - b.kronenAnsatz)*2.0f;
   ((ForesterTree)(f1.foresterTrees.get(b.descriptionIndex))).crownSize[2]=
      b.maximalerKronenRadius*2.0;
         }
      }

   }
]

protected  void abspeichern(ForesterDescription f1, String url)
// Methode zu Abspeichern einer Forester-Description
{
   for (apply(1)) aktualisiereForesterDescription(f1); // Die
   foresterconnection.ForesterDescription wird mit den Modeldaten aktualisiert
   f1.speichereForesterDescription( url);
}
```

```
//-------------------------------------------------------------------------------
//          Erzeugen der Baumsegmente und Initialisieren der Baumgrunddaten
//-------------------------------------------------------------------------------
private void setzeBaumGrundfarbe(int farbe)
//Servicefunktion zum Setzen der Baumgrundfarbe
[
   b:Baum ::> {
      b.setColor(farbe);
   }
]

private  void erzeugeBaumSegmente()
// der Baum wird aus einzelnen Segmenten zusammengesetzt
[
   b:Baum(bnr, alter, stemlength, bhd, typ, ka) ==>  b.(setName("Baum" + bnr))
   {
      float tmp = 0.0f, rest = 0.0f, initiale_segment_leaenge = 0.0f;
      int count = 0;
      // ermitteln der Segmentlänge: Segmentlänge muss < vorgegebener Maximalwert sein
      // Segmentlänge = baumHöhe / (i * baumalter)  : für i= 1 ...x bis Segmentlänge <
   maximale Segmentlänge
      do {
         count += b.baumAlter;
         initiale_segment_leaenge = stemlength / count;
         } while (initiale_segment_leaenge > Definitionen.MAX_SEGMENTLAENGE[b.baumArt]);
   }
   // zeichnen der Segmente als Tochtergraphen des Baumes
   [
      for( int i = 1; i <= b.baumAlter ; i++) (
         s:Segment(bnr, i, initiale_segment_leaenge, initiale_segment_leaenge *
            (i -1)).(setName("Segment" + bnr + "_" + i))
         { s.top = s.base + s.length; }
      )
   ];
]

//-------------------------------------------------------------------------
//               Nadelmassen im Segment
// ------------------------------------------------------------------------
public void initialisiereSegmenteFuerJahresDurchlauf()
{
//berechnet die Nadelmassen für alle Segmente

   for(apply(1))  kalkuliereBaumBetafunktionKorrekturintegrale();
   for(apply(1))  kalkuliereSegmenteNadelMasse();
   for(apply(1))  ermittleKronenAnsatzSegment();
}

public  void kalkuliereBaumBetafunktionKorrekturintegrale()
// berechnet die Korrekturintegrale (Betafunktion) für die Betaverteilung der
   Nadeldichten
[
   // 1. schritt 1 Nullsetzung der Variablen
   b:Baum ::> {
      for(int i = 0 ; i<7; i++) b.betafunktion_korrekturintegral_pro_jahrgang[i] = 0.0f;
   }
   // 2. Berechnung der Korrekturintegrale der Bäume für die einzelnen Nadeljahrgänge
   für alle segmente
   s:Segment -ancestor-> b:Baum  ::> {
      if (b.maxNadeljahrgang >= 1) { b.betafunktion_korrekturintegral_pro_jahrgang[0] +=
         s.calculateSegmentBetaValue( b.a1, b.b1, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
      if (b.maxNadeljahrgang >= 2) { b.betafunktion_korrekturintegral_pro_jahrgang[1] +=
         s.calculateSegmentBetaValue( b.a2, b.b2, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
      if (b.maxNadeljahrgang >= 3) { b.betafunktion_korrekturintegral_pro_jahrgang[2] +=
         s.calculateSegmentBetaValue( b.a3, b.b3, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
      if (b.maxNadeljahrgang >= 4) { b.betafunktion_korrekturintegral_pro_jahrgang[3] +=
         s.calculateSegmentBetaValue( b.a4, b.b4, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
      if (b.maxNadeljahrgang >= 5) { b.betafunktion_korrekturintegral_pro_jahrgang[4] +=
         s.calculateSegmentBetaValue( b.a5, b.b5, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
```

```
      if (b.maxNadeljahrgang >= 6) { b.betafunktion_korrekturintegral_pro_jahrgang[5] +=
         s.calculateSegmentBetaValue( b.a6, b.b6, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
      if (b.maxNadeljahrgang >= 7) { b.betafunktion_korrekturintegral_pro_jahrgang[6] +=
         s.calculateSegmentBetaValue( b.a7, b.b7, s.base, s.base + s.length, b.beta_q,
         b.beta_p); }
   }
]


public  void kalkuliereSegmenteNadelMasse()
//berechnet die Nadeldmasse pro Segment und Jahrgang auf Basis der Jahrgangsnadelmasse
   und der Betaverteilung zur Nadeldichte
[
   s:Segment -ancestor-> b:Baum ::> {
      s[nadelMasseGesamt] = 0.0f; // nullsetzung
      b[countSegmente] ++;  // zähler für die segmente
      if (b.maxNadeljahrgang >= 1 && b.betafunktion_korrekturintegral_pro_jahrgang[0] >
         0.0f) { s[nadelMasse1] = b.nadelMasse1 * s.calculateSegmentBetaValue( b.a1,
         b.b1, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[0];  s[nadelMasseGesamt] +=
         s.nadelMasse1; }
      if (b.maxNadeljahrgang >= 2 && b.betafunktion_korrekturintegral_pro_jahrgang[1] >
         0.0f) { s[nadelMasse2] = b.nadelMasse2 * s.calculateSegmentBetaValue( b.a2,
         b.b2, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[1];  s[nadelMasseGesamt] +=
         s.nadelMasse2; }
      if (b.maxNadeljahrgang >= 3 && b.betafunktion_korrekturintegral_pro_jahrgang[2] >
         0.0f) { s[nadelMasse3] = b.nadelMasse3 * s.calculateSegmentBetaValue( b.a3,
         b.b3, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[2];  s[nadelMasseGesamt] +=
         s.nadelMasse3; }
      if (b.maxNadeljahrgang >= 4 && b.betafunktion_korrekturintegral_pro_jahrgang[3] >
         0.0f) { s[nadelMasse4] = b.nadelMasse4 * s.calculateSegmentBetaValue( b.a4,
         b.b4, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[3];  s[nadelMasseGesamt] +=
         s.nadelMasse4; }
      if (b.maxNadeljahrgang >= 5 && b.betafunktion_korrekturintegral_pro_jahrgang[4] >
         0.0f) { s[nadelMasse5] = b.nadelMasse5 * s.calculateSegmentBetaValue( b.a5,
         b.b5, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[4];  s[nadelMasseGesamt] +=
         s.nadelMasse5; }
      if (b.maxNadeljahrgang >= 6 && b.betafunktion_korrekturintegral_pro_jahrgang[5] >
         0.0f) { s[nadelMasse6] = b.nadelMasse6 * s.calculateSegmentBetaValue( b.a6,
         b.b6, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[5];  s[nadelMasseGesamt] +=
         s.nadelMasse6; }
      if (b.maxNadeljahrgang >= 7 && b.betafunktion_korrekturintegral_pro_jahrgang[6] >
         0.0f) { s[nadelMasse7] = b.nadelMasse7 * s.calculateSegmentBetaValue( b.a7,
         b.b7, s.base, s.base + s.length, b.beta_q, b.beta_p) /
         b.betafunktion_korrekturintegral_pro_jahrgang[6];  s[nadelMasseGesamt] +=
         s.nadelMasse7; }
      if (s.nadelMasseGesamt > 0.0f)  s.color = F_BENADELT; else s.color = F_UNBENADELT;
   }

]

public  void ermittleKronenAnsatzSegment()
//Ermiiteln das Segment in dem der Kronenansatzpunkt liegt
[
   // Nullsetzung der Baumeigenschaft Kronenansatzsegment
   b:Baum ::> {
      b[kronenAnsatzSegment] = 0;
   }

   { float h = 0.0f;}
   s:Segment -ancestor-> b:Baum,(s.nadelMasseGesamt > 0.0f) ::> {
      if (b.kronenAnsatzSegment == 0) h = b.baumHoehe +1;
      if ( s.top < h) { h = s.top; b[kronenAnsatzSegment] = s.snr;}
   }

]
```

```
//-------------------------------------------------------------------------
//                              Beschattung
//-------------------------------------------------------------------------
// Erzeugung eines Bestandesgürtel zur Absicherung der Randeffekte.
// wird vor der Beschattungberechnung gestartet - muss nacher wieder gelöscht werden
// Bezugspunkt für alle Bestände ist der Nullpunkt (Mutterknoten)
// Referenzpunkt der Bestände ist die linke untere Ecke
// Ist der Hauptbestand veschoben, werden die Schattenkopien um dieselbe Distanz
   verschoben
// dies wird (durch mitte.x + xseite und mitte.y + yseite gewährleistet)

 public void erzeugeRandbestaendeAlsKopieDesHauptbestandes()
 // Anlegen der Bestandeskopien
{
   Point3d referenzpunkt;
   [
      Nullpunkt [ best:Bestand(mitte, xseite, yseite, aktiv) ],(aktiv == true) ==>
      Nullpunkt [ best ]

      {referenzpunkt = new Point3d(mitte.x + xseite , mitte.y + yseite, mitte.z);}

      [ besta:((Bestand)cloneSubgraph(best))]
      {  besta.setTransform(mitte.x + xseite , mitte.y + yseite, mitte.z);
         besta.referenzpunkt = referenzpunkt; besta.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x + xseite , mitte.y, mitte.z);}
      [ bestb:((Bestand)cloneSubgraph(best))]
      {  bestb.setTransform(mitte.x + xseite , mitte.y, mitte.z);
         bestb.referenzpunkt = referenzpunkt; bestb.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x + xseite , mitte.y - yseite, mitte.z);}
      [ bestc:((Bestand)cloneSubgraph(best))]
      {  bestc.setTransform(mitte.x + xseite , mitte.y - yseite, mitte.z);
         bestc.referenzpunkt = referenzpunkt; bestc.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x - xseite , mitte.y + yseite, mitte.z);}
      [ bestd:((Bestand)cloneSubgraph(best))]
      {  bestd.setTransform(mitte.x - xseite , mitte.y + yseite, mitte.z);
         bestd.referenzpunkt = referenzpunkt; bestd.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x - xseite , mitte.y , mitte.z);}
      [ beste:((Bestand)cloneSubgraph(best))]
      {  beste.setTransform(mitte.x - xseite , mitte.y , mitte.z);
         beste.referenzpunkt = referenzpunkt; beste.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x - xseite , mitte.y - yseite, mitte.z);}
      [ bestf:((Bestand)cloneSubgraph(best))]
      {  bestf.setTransform(mitte.x - xseite , mitte.y - yseite, mitte.z);
         bestf.referenzpunkt = referenzpunkt; bestf.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x , mitte.y - yseite, mitte.z);}
      [ bestg:((Bestand)cloneSubgraph(best))]
      {  bestg.setTransform(mitte.x , mitte.y - yseite, mitte.z);
         bestg.referenzpunkt = referenzpunkt; bestg.aktiv = false;
      }

      {referenzpunkt = new Point3d(mitte.x  , mitte.y + yseite, mitte.z);}
      [ besth:((Bestand)cloneSubgraph(best))]
      {  besth.setTransform(mitte.x  , mitte.y + yseite, mitte.z);
         besth.referenzpunkt = referenzpunkt; besth.aktiv = false;
      }
      {};
   ]
}
public  void setzeFarbmarkerFuerRandInaktivenBestand()
//Servicefunktion zur farblichen Markierung von Bestandeskopien
[
   b:Baum -ancestor-> best:Bestand, (best.aktiv == false) ::>
```

```
      {  b.setColor(NICHT_AKTIV);}

]

public  void deleteRandBestaende()
// Methode zum Entfernen des Bestandesgürtels rund um den Haupbestand.
// Steuerung erfolgt über die Eigenschaft aktiv des jeweiligen Bestandes
[
   best:Bestand(referenzpunkt, xseite, yseite, aktiv), ( aktiv == false) ==>> ;
]


private  void kalkuliereExterneBeschattung( float winkel)
// Berechnung der Beschattung durch Segmente der Nachbarbäume
{
   int alt = 0;
   long t1 = System.currentTimeMillis();
   [
   //gehe über alle Segmente des Hauptbestandes
   s:Segment -ancestor-> best:Bestand,( s.nadelMasseGesamt > 0.0f && best.aktiv == true)
   ::>
   {
      s.externeBeschattung = 0.0f;
      // ermitteln der beschattenden Segmente (a) für das aktuelle Segment (s)
      for ((* b:Baum (-->)* a:Segment , ( s.getParent() != b && b.getLastChild() in
         cone(s, false, winkel)),( a.nadelMasseGesamt > 0.0f && a in cone(s, false,
         winkel)) *))
         {
            s[externeBeschattung] += ( a.nadelMasseGesamt *
               b.lichtTransmissionsKoeffizient / (distanceSquared(a, s) +
               b.kalkuliereNadelMasseVerteilungKorrekturTerm() ));
         };
         if (s.externeBeschattung > 0.0f) s[color] = F_BESCHATTET;
      }
   ]
   t1 = System.currentTimeMillis() - t1;
   println("   Rechenzeit: "  + t1  + " für externe Beschattung");
}

private  void setzeFarbmarkerFuerBeschatteteSegmente()
//Servicefunktion zur farblichen Markierung beschatteter Segmente
[
   s:Segment, (s.externeBeschattung > 0.0f && s.nadelMasseGesamt > 0.0f) ::>
   {
      s[color] =  F_BESCHATTET;
   }
]

private  void kalkuliereSelbstBeschattung()
//Berechnung der Beschattung durch eigne Segmente des Baumes
{
   long t1 = System.currentTimeMillis();
   [
   //gehe über alle benadelten Segmente dieses Baumes
   s:Segment -ancestor-> best:Bestand, (s.nadelMasseGesamt > 0.0f && best.aktiv == true)
   ::>
      {
      s.selbstBeschattung = 0.0f;
      // ermitteln der beschattenden Segmente für das aktuelle Segment
      for ((* b:Baum (-->)* a:Segment , ( s.getParent() == b), (   a.snr >= s.snr) *))
         {
         s[selbstBeschattung] += ( a.nadelMasseGesamt * b.lichtTransmissionsKoeffizient
            / (distanceSquared(a, s)+ b.kalkuliereNadelMasseVerteilungKorrekturTerm()));
         };
      }
   ]
   t1 = System.currentTimeMillis() - t1;
   println("   Rechenzeit: "  + t1  + " für selbst Beschattung");
}

private  void addiereGesamtBaumBeschattung(int lbnr)
[
//gehe über alle Segmente dieses Baumes
   s:Segment, (s.bnr == lbnr ) ::>
   {
      s[gesamtBeschattung] = s.externeBeschattung + s.selbstBeschattung;
```

```
   }
]


//-------------------------------------------------------------------------
//         PHOTOSYNTHESE
//-------------------------------------------------------------------------
private void kalkuliereBaumPhotosynthese()
[
//kalkuliert die Bruttophotosntheseleistung getrennt für die einzelnen Jahrgänge

// Nullsetzung aller Baum-Assimlatespeicher
   b:Baum ::> {   b[bruttoAssimilateSpeicher] = 0.0f;}

//gehe über alle benadelte Segmente (geodnet nach Bäumen)
   s:Segment -ancestor-> b:Baum, (s.nadelMasseGesamt > 0.0f) ::>
   {
      s[assimilateSpeicher] = 0.0f;
         switch(b.verwendetePhotosyntheseFunktion) {
            case 1: s.relativeProduktionsQuoteBeschattet  =
               s.kalkuliereProduktionsQuoteBeschattet1(b.pfreundtKoeffizient_A,
               b.pfreundtKoeffizient_B); break;
            default: s.relativeProduktionsQuoteBeschattet =
               s.kalkuliereProduktionsQuoteBeschattet0(b.beerLambertKoeffizient_A,
               b.beerLambertKoeffizient_B); break;
      }
      for(int i = 1 ; i <= b.maxNadeljahrgang; i++){
         switch(i){
            case 1: s[assimilateSpeicher] += s.nadelMasse1 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[0] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 2: s[assimilateSpeicher] += s.nadelMasse2 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[1] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 3: s[assimilateSpeicher] += s.nadelMasse3 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[2] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 4: s[assimilateSpeicher] += s.nadelMasse4 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[3] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 5: s[assimilateSpeicher] += s.nadelMasse5 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[4] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 6: s[assimilateSpeicher] += s.nadelMasse6 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[5] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
            case 7: s[assimilateSpeicher] += s.nadelMasse7 *
               b.nadelJahrgangReativePhotosyntheseKapazitaet[6] *
               b.produktionsRateUnbeschattet * s.relativeProduktionsQuoteBeschattet;
               break;
         }
      }
      b[bruttoAssimilateSpeicher] += s.assimilateSpeicher;
   }
]


private  void addiereNettoPhotosynthese()
// Addieren der Photosynthese - Respiration = Nettophotsynthese apparent
[
   b:Baum ::> {   b[nettoAssimilateSpeicher] = b.bruttoAssimilateSpeicher -
   b.gesamtRespiration;
      b.kalkuliereProduktionsEffizienz();
      b.kalkuliereRelativeProduktionsEffizienz();
   }
]
```

```
//-------------------------------------------------------------------------------
//                                  Respiration
//-------------------------------------------------------------------------------

private  void kalkuliereRespiration()
//Steuerung der Respirationsberechnung
{
    for(apply(1)) kalkuliereBaumNadelRespiration();
    for(apply(1)) kalkuliereBaumwWurzelRespirationMethode2();
    for(apply(1)) kalkuliereBaumStammRespirationMethode2();
    for(apply(1)) kalkuliereBaumAstRespirationMethode2();
    for(apply(1)) addiereBaumRespiration();
}

private  void addiereBaumRespiration()
[
// Addieren der einzelnen rRespirationsraten
   b:Baum ::> {   b[gesamtRespiration] = b.nadelRespiration + b.astRespiration +
   b.stammRespiration + b.wurzelRespiration;}
]

private void kalkuliereBaumNadelRespiration()
// Nadelrespiration erfolgt proportional zur Nadelmasse über alle Nadeljahrgänge
[
   // Nullsetzung der rRespirationswerte
   b:Baum ::> {   b[nadelRespiration] = 0.0f;}

   //gehe über alle Segmente geodnet nach Bäumen
   s:Segment -ancestor-> b:Baum ::>
   {
      s[nadelRespiration] = 0.0f;
      s[nadelRespiration] += s.nadelMasseGesamt * b.nadelRespirationRate;
      b[nadelRespiration] += s.nadelRespiration;
   }
]

private  void kalkuliereBaumwWurzelRespirationMethode1()
// Methode 1 nach Sloboda & Pfreundt (1989) führt zu überhöhten Werten
[
   {float dmax = 0.0f, tmp = 0.0f;}
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[wurzelRespiration] = 0.0f;
      dmax = b.d13 * 100 * b.baumHoehe / ( b.baumHoehe - 1.3f);
      b[wurzelRespiration] = ( b.feinWurzelMasseG + b.grobWurzelMasseG ) / dmax /
         b.holzteileRespirationKoeffizient_A * (float)
         Math.log((b.holzteileRespirationKoeffizient_A * dmax +
         b.holzteileRespirationKoeffizient_B) /  b.holzteileRespirationKoeffizient_B);
   }
]

private  void kalkuliereBaumwWurzelRespirationMethode2()
// Alternative Methode zur Wurzelrespiration in Abhängigkeit der Wurzelmasse
[
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[wurzelRespiration] = 0.0f;
      b[wurzelRespiration] = ( b.feinWurzelMasseG + b.grobWurzelMasseG ) *
         b.wurzelRespirationrateMethode2;
   }
]
private  void kalkuliereBaumStammRespirationMethode1()
// Methode 1 nach Sloboda & Pfreundt (1989)führt zu überhöhten Werten
[
   {float admax = 0.0f;}
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[stammRespiration] = 0.0f;
      admax = b.d13 * 100 * b.baumHoehe/ ( b.baumHoehe - 1.3f) *
         b.holzteileRespirationKoeffizient_A;
      b[stammRespiration] = ( 3 * b.stammMasseG / (admax * admax * admax))
         * (((admax  * 0.5 ) - b.holzteileRespirationKoeffizient_B) * admax
         + log(( admax + b.holzteileRespirationKoeffizient_B) /
```

```
         b.holzteileRespirationKoeffizient_B) * b.holzteileRespirationKoeffizient_B
         *b.holzteileRespirationKoeffizient_B) ;
   }
]

private  void kalkuliereBaumStammRespirationMethode2()
// Alternative Methode zur Stammrespiration
// Bezuggröße ist Respirationsrate pro m^2 Stammoberfläche

[
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[stammRespiration] = 0.0f;
      b[stammRespiration] =  b.stammOberflaeche * b.stammRespirationrateMethode2;
   }
]

private  void kalkuliereBaumAstRespirationMethode1()
// Methode 1 nach Sloboda & Pfreundt (1989) führt zu überhöhten Werten
[
   {float admax = 0.0f;}
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[astRespiration] = 0.0f;
      admax = b.d13 * 100 * 0.1f + 0.5f *  b.holzteileRespirationKoeffizient_A;
      b[astRespiration] = b.astMasseG / admax  *log((  admax +
         b.holzteileRespirationKoeffizient_B) /  b.holzteileRespirationKoeffizient_B);
   }
]

private  void kalkuliereBaumAstRespirationMethode2()
// Alternative Methode zur Astrespiration
// Bezuggröße ist eine Respirationsrate kgC Biomasse der Äste

[
   // Nullsetzung der Respirationswerte
   b:Baum ::> {
      b[astRespiration] = 0.0f;
      b[astRespiration] =  b.astMasseG * b.astRespirationrateMethode2;
   }
]


//-------------------------------------------------------------------------
//                          Allokation
//-------------------------------------------------------------------------

private  void verteileAssimilateAufKompartimente()
// Verteilung des Nettoasssimilatespreichers auf die Baumkompartimente
{
   float tmp = 0.0   ;
   [
      // Nullsetzung der Respirationswerte
      b:Baum ::> {
         // 1. Schritt Energiebedarf für das Wachrums = Wachstumsrespiration
         b[wachstumsRespiration] = b[nettoAssimilateSpeicher] *
            b.wachstumsRespirationRate;
         b[nettoAssimilateSpeicher] -= b[wachstumsRespiration];

         // 2. Schritt Feinwurzelwachstum (proportional zur Bruttoproduktion
         b[feinWurzelMasseTot]= b.feinWurzelMasseG * b.feinWurzelUeberlebensrate;
         b[feinWurzelMasseNeu] = b.kalkulierefFeinwurzelWachstum();
         b[feinWurzelMasseG] = b[feinWurzelMasseTot] +b[feinWurzelMasseNeu];
         b.nettoAssimilateSpeicher -= b[feinWurzelMasseNeu];

         // 3. Schritt Nadelwachtum
         if(b.nettoAssimilateSpeicher > 0.0f) {
            b[nadelMasseNeu] = b.kalkuliereNadelWachstum();
            b.nettoAssimilateSpeicher -= b[nadelMasseNeu];
         }
         else {
         // Was passiert wenn Speicher bereits leer. Funktion für Notuaustrieb einfügen?
            b[nadelMasseNeu] = 0.0f;
         }

         // 4. Schritt Astmassenwachtum und Absterben von Ästen
         b[astMasseTot] = b.astMasseG * (1- b.astUeberlebensRate);
```

```
         b[astMasseNeu]  = b.kalkuliereAstWachstum();
         b[astMasseG] = b[astMasseTot] + b[astMasseNeu] ;
         b.nettoAssimilateSpeicher -= b[astMasseNeu] ;

         // 4. Schritt Grobwurzelmassenwachstum und absterben
         b[grobWurzelMasseTot] = b.grobWurzelMasseG * (1- b.grobWurzelUeberlebensRate);
         b[grobWurzelMasseNeu]  = b.kalkuliereGrobwurzelWachstum(b.grobWurzelMasseTot);
         b[grobWurzelMasseG] =  b.grobWurzelMasseTot + b.grobWurzelMasseNeu;
         b.nettoAssimilateSpeicher -= b.grobWurzelMasseNeu;

         // 5. Schritt Stammmasse (der rest der Assimilate)
         b[stammMasseNeu] = Math.max( b.nettoAssimilateSpeicher , 0.0f);
         b[stammMasseG] += b[stammMasseNeu];
         b[nettoAssimilateSpeicher] -= b[stammMasseNeu];
      }
   ]
}

//-------------------------------------------------------------------------
//                          Struktur wachsen lassen
//-------------------------------------------------------------------------
public void ermittleNeuenKronenAnsatz() {
// Berechnen des neuen Kronenansatzsegmentes
   [
      b:Baum ::> {
      float hh = b.baumHoehe;
      for((* s:Segment, (s.bnr == b.bnr && (s.assimilateSpeicher - s.nadelRespiration) >
         0.0f) *))
      {
         if (  s.base < hh)
            { hh = s.base ;}
      }

      if(hh > b[kronenAnsatz]){
         println("Baum" + b.bnr  + " Kronenansatz von: "  + b.kronenAnsatz + "auf: " +
            hh + " gesetzt.");
         b[kronenAnsatz] = hh;
         if( b.maxNadeljahrgang <= 1) b[a1] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 2) b[a2] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 3) b[a3] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 4) b[a4] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 5) b[a5] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 6) b[a6] = b[kronenAnsatz];
         if( b.maxNadeljahrgang <= 7) b[a7] = b[kronenAnsatz];
         }
      }
   ]
}

private  void aktualisiereBetaFunktionswerte()
// Aktualisierung der Daten für die Nadeldichte für die nächste Wachstumsperiode
[
   b:Baum ::>
   { b.updateBetaFunctionValues();}
]


public  void uebertrageBaumdatenInsNaechsteJahr()
// Aktualisieren der Baumdaten für die nächste Wachstumsperiode
[
   b:Baum ::> {
      if (b.maxNadeljahrgang >= 7) {   b.a7 = b.a6; b.b7 = b.b6; b.nadelMasse7 =
         b.nadelMasse6 * b.nadelJahrgangUeberlebensrate[6];}
      if (b.maxNadeljahrgang >= 6) {  b.a6 = b.a5; b.b6 = b.b5; b.nadelMasse6 =
         b.nadelMasse5 *b.nadelJahrgangUeberlebensrate[5];}
      if (b.maxNadeljahrgang>= 5) {  b.a5 = b.a4; b.b5 = b.b4; b.nadelMasse5 =
         b.nadelMasse4 * b.nadelJahrgangUeberlebensrate[4];}
      if (b.maxNadeljahrgang >= 4) {  b.a4 = b.a3; b.b4 = b.b3; b.nadelMasse4 =
         b.nadelMasse3 * b.nadelJahrgangUeberlebensrate[3];}
      if (b.maxNadeljahrgang>= 3) {  b.b3 = b.a2; b.b3 = b.b2; b.nadelMasse3 =
         b.nadelMasse2 * b.nadelJahrgangUeberlebensrate[2];}
      if (b.maxNadeljahrgang >= 2) {  b.a2 = b.a1; b.b2 = b.b1; b.nadelMasse2 =
         b.nadelMasse1 * b.nadelJahrgangUeberlebensrate[1];}
      if (b.maxNadeljahrgang >= 1) {  b.a1 = b.a1; b.b1 = b.baumHoehe; b.nadelMasse1 =
         b.nadelMasseNeu * b.nadelJahrgangUeberlebensrate[0];}
      b.nadelMasseNeu = 0.0f;
      b[nadelMasseG] = b.addiereNadelJahrgangMassen();
```

```
      b[baumAlter]++;
      b[cMasseGesamt]= b.addiereKompartimentMassen();
   }
]


public void berechneHoehenwachstumsBaumdaten()
// Berechnung des Höhenwachstums
{

   float sigmaHoehe[] = new float[Definitionen.MAXBAUMARTEN];
   float sh[] = new float[Definitionen.MAXBAUMARTEN];
   float sh2[] = new float[Definitionen.MAXBAUMARTEN];
   int n[] = new int[Definitionen.MAXBAUMARTEN];

   for( int i=0; i< Definitionen.MAXBAUMARTEN; i++){ sigmaHoehe[i] = 0.0f; sh[i] = 0.0f;
   sh2[i] = 0.0f; n[i] = 0;}

   [
      //berechnung der Varianz der Höhe
      b:Baum ::> {
         sh[b.baumArt] += b.baumHoehe;
         sh2[b.baumArt] += b.baumHoehe * b.baumHoehe;
         n[b.baumArt] ++;
      }
   ]
   for( int i=0; i< Definitionen.MAXBAUMARTEN; i++){
      // sigmaHoehe = Standardabweichung der Höhen
      //= Wurzel aus Varianz der Höhen =Wurzel aus ( E((X-mü)^2) oder E(X^2)-E(x)^2 )
      if(n[i]  != 0f) sigmaHoehe[i] = Math.sqrt((sh2[i]/n[i]) - pow(sh[i]/n[i],2));
   }
   // Kalkulation des Höhenzuwachses und des Durchmessergewinns
   [
      b:Baum ::> {
         // wenn keine neue Stammmasse, dann auch kein Höhenwachstum.
         if(b.stammMasseNeu > 0.0f) {
         b.baumHoehe += b.kalkuliereBaumHoehenZuwach(sigmaHoehe[b.baumArt]);
         b[d13] = b.kalkuliereStammDurchmesser();
         b[stammOberflaeche] = b.kalkuliereStammOberflaeche();
         b[maximalerKronenRadius] += b.kalkuliereKronenRadiusZuwachs();
         }
      }
   ]
}

public  void anfuegeSegmente()
//Anfügen neuer Segmente
[
   {Baum b; float tmp = 0.0f; int count = 0; float laengenzuwachs = 0.0f;}

   s:Segment, ( s == s.getParent().getLastChild()) ==> s
   {
      count = 0; laengenzuwachs = 0.0f;
      b =  (Baum) s.getParent();
      laengenzuwachs = b.baumHoehe - s.top;
      if(laengenzuwachs > 0.0001f) {
         do {
            count++;
            tmp = laengenzuwachs / count;
            } while (tmp > Definitionen.MAX_SEGMENTLAENGE[b.baumArt]);
      }
   }

   for(int j = 1; j <= count; j++) (
      news:Segment(s.bnr, s.snr + j, tmp, s.top)
      {
         news.setName("Baum" + s.bnr + "_" + (s.snr+j));
         news.top = news.base + news.length;;
      }
   );
]
```

```
//*****************************************************************************
//        Ab hier keine wichtigen Modellteile : Nur Ausgabemethoden
//*****************************************************************************

//-----------------------------------------------------------------------------
// Methoden zum Visulalisieren eines Beschattungskonus
//-----------------------------------------------------------------------------
public void Kontrollkonus(){
   for (apply(1)) ausgabe.zeichneKontrollkonus(
      ForesterUtils.getParameterFromDialog(
            "Baum Auswahl",
            "Bitte geben Sie eine Baumnummer an",
            "1",
            1),
      ForesterUtils.getParameterFromDialog(
            "Segment Auswahl",
            "Bitte geben Sie eine Segmentnummer an",
            "1",
            1),
      ForesterUtils.getParameterFromDialog(
            "Winkel Auswahl",
            "Bitte geben Sie den Öffnungswinkel des Konus an",
            "60.0",
            60f)
            );
}

public  void deleteKonus()
//Entfernt den Kontrollkonus
[
   c:Cone ==>>;
]

//-----------------------------------------------------------------------------------------
// Block mit Servicefunktionen zur Ausgabe von Werten oder zum Zeichnen von Charts
//-----------------------------------------------------------------------------------------
public  void zeichneNadeldichteChart(){

   for(apply(1)) ausgabe.zeichneNadeldichteChart(
      ForesterUtils.getParameterFromDialog(
         "Nadeljahrgang Auswahl",
         "Bitte wählern Sie einen Nadeljahrgang für die Grafik (0 = GesamttNadelmasse)",
         Integer.toString(beobachteterNadeljahrgang),
          beobachteterNadeljahrgang));
   }

public  void zeichnerRealtiveProduktionsQuoteChart(){
// Servicefunktion für Chart Ausgabe der Produktionquote
              ausgabe.zeichneRealtiveProduktionsQuoteChart(
              ForesterUtils.getParameterFromDialog(
                 "Baum Auswahl",
                 "Bitte geben Sie eine Baumnummer an",
                 "1",
                 1),
                 ForesterUtils.getParameterFromDialog(
                 "Ab Kronenansatz?",
                 "ja = true",
                 "true",
                 true)
                 );
}
public void zeichneProduktionsStaerkeChart(){
// Servicefunktion für Chart Ausgabe der Produktionsquote
          for(apply(1))    ausgabe.zeichneProduktionsStaerkeChart(
              ForesterUtils.getParameterFromDialog(
                 "Baum Auswahl",
                 "Bitte geben Sie eine Baumnummer an",
                 "1",
                 1),
                 ForesterUtils.getParameterFromDialog(
                 "Ab Kronenansatz?",
                 "ja = true",
                 "true",
```

```
                  true)
                  );
}

public void zeichneBeschattungChart(){
// Servicefunktion für Chart Ausgabe der Beschattungswerte
   int baum = ForesterUtils.getParameterFromDialog(
            "Baum Auswahl",
            "Bitte geben Sie eine Baumnummer an, 0 für alle",
            "0",
            0);
            if (baum == 0) {
             for(apply(1)) ausgabe.zeichneBeschattungChart(
               ForesterUtils.getParameterFromDialog(
                  "Beschattung Auswahl",
                  "Bitte wählern Sie den Typ der Beschattung für die Grafik aus.(bs =
   Selbst-, be = Externe, defualt = Gesamtbeschattung)",
                  "bg",
                  "bg")
                  );
            }
            else
               {
                for(apply(1)) ausgabe.zeichneBeschattungChart(
               ForesterUtils.getParameterFromDialog(
                  "Beschattung Auswahl",
                  "Bitte wählern Sie den Typ der Beschattung für die Grafik aus.(bs =
   Selbst-, be = Externe, defualt = Gesamtbeschattung)",
                  "bg",
                  "bg"),
                  baum,
                  ForesterUtils.getParameterFromDialog(
                  "Ab Kronenansatz?",
                  "ja = true",
                  "true",
                  true)
                  );
               }
}


public void ausgabeBetaKorrekturIntegrale(){
// Servicefunktion für das Debugging
 for(apply(1)) ausgabe.ausgebenBetafunktion_korrekturintegral_pro_jahrgang(
         ForesterUtils.getParameterFromDialog(
            "Baum Auswahl",
            "Bitte geben Sie eine Baumnummer an",
            "1",
            1)
      );
}

//-----------------------------------------------------------------------------
//                          Krone zeichnen
//-----------------------------------------------------------------------------


public void zeichneKroneNachNadeldichte(){
// Starten der Kronenerzeugung nach Nadelmasse
   [
      b:Baum ::>
      { ermittleKronenStuetzpunkteNachNadeldichte(b.bnr);}
   ]
    for(apply(1)) zeichneKrone2();

}

public void zeichneKroneNachKronenform(){
// Starten der Kronenerzeugung nach Kronenform
   [
      b:Baum ::>
      { b.ermittleKronenStuetzpunkteNachKronenform();}
   ]
    for(apply(1)) outputKronenStuetzPunkte();
    for(apply(1)) zeichneKrone2();

}
```

```
private void zeichneKrone2() {
// zeichnet eine Krone  direkt als Rotationskörper
   [
      b:Baum, (b.nadelMasseG > 0.0f) ==> b [
         {
            float data[] = new float[b.kronenStuetzPunkte[0].length * 2];
            for(int i = 0; i < b.kronenStuetzPunkte[0].length; i++){
               data[(i*2)] = b.kronenStuetzPunkte[1][i];
               data[(i*2)+1] = b.kronenStuetzPunkte[0][i];

            }
            VertexListImpl punkte = new VertexListImpl(data, 2 );
            BSplineOfVertices spline = new BSplineOfVertices(punkte, 1, false, false);
         }
         Krone(b.bnr, b.kronenAnsatz, b.baumHoehe)
         Surface(new SwungSurface(spline, new Circle(1), 0,1))
      ];
   ]
}

public  void deleteKrone()
//entfernt die alten Krone
[
   k:Krone ==>>;
]

protected void ermittleKronenStuetzpunkteNachNadeldichte(int bnr)
//berechnet Werte für das Zeichnen einer Krone nach Nadelmassen
[
   { float inc = 0.0f;}
   b:Baum, (b.bnr == bnr) ::> {
      inc = (b.baumHoehe - b.kronenAnsatz) / ( b.kronenStuetzPunkte[0].length -1);
      b.kronenStuetzPunkte[0][0] = b.kronenAnsatz;
      b.kronenStuetzPunkte[1][0] = 0.01f;
      b.kronenStuetzPunkte[0][b.kronenStuetzPunkte[0].length -1] = b.baumHoehe;
      b.kronenStuetzPunkte[1][b.kronenStuetzPunkte[1].length -1] = 0.01f;

      for( int i = 1; i < b.kronenStuetzPunkte[0].length - 1; i++){
         b.kronenStuetzPunkte[0][i] = ( i * inc) + b.kronenAnsatz;
         b.kronenStuetzPunkte[1][i] = 0.01f;
      }
   }

   { float h = 0.0f;}
   s:Segment -ancestor-> b:Baum, (b.bnr == bnr) ::> {
      for( int i = 1; i < b.kronenStuetzPunkte[0].length -1; i++){
         if ( b.kronenStuetzPunkte[0][i] > s.base && b.kronenStuetzPunkte[0][i] <
   s.top){
            b.kronenStuetzPunkte[1][i] = s.nadelMasseGesamt;
         }
      }
   }

]

// Hilfsmethode zur Kontrollausgabe der Kronenstützpunkte
public  void outputKronenStuetzPunkte(){
   [
      b:Baum::> {
         println("Baum: " + b.bnr + " ka: " + b.kronenAnsatz + " Hoehe: " + b.baumHoehe
          + " Maximaler Kronenradius: " + b.maximalerKronenRadius);
         for(int i = 0; i < b.kronenStuetzPunkte[0].length; i++){
            print("" + i + "- x: " + b.kronenStuetzPunkte[0][i] + " y: " +
               b.kronenStuetzPunkte[1][i]);
            println("");
         }
         println("");
      }

   ]
}
// ende wald.rgg
```

10.2.3 Code baum.rgg

```
//------------------------------------------------------------------------
//                       Baumklasse
//------------------------------------------------------------------------


class Baumklasse extends Sphere
{
// Bereich 1: eins baumartenspezifische Eigenschaften: werden durch den Konstruktor
   gesetzt und aus der Klasse definitionen abgeleitet. Für Kommentare sie
   definitionen.rgg

      public int maxNadeljahrgang = 0;
      public float[] kompartimentAnteile = new float[5];
      public float[] nadelJahrgangUeberlebensrate  = new float[7];
      public float[] nadelJahrgangReativePhotosyntheseKapazitaet  = new float[7];
      public float[] mittelStammAltersHoehenkurve  = new float[7];

      public float [] formzahl_Koeffizienten = new float[7];

      public float nadelMasseVerteilungKorrekturfaktor  = 0.0f;
      public float[] kronenformParameter = new float [4];

      // Photosyntheseleistung unbeschattet
      public float produktionsRateUnbeschattet ;

      // Respirationsparameter - kgC/kgC
      public float nadelRespirationRate;
      public float holzteileRespirationKoeffizient_A;
      public float holzteileRespirationKoeffizient_B;
      public float astRespirationrateMethode2;
      public float wurzelRespirationrateMethode2;
      public float wachstumsRespirationRate;

      // Respirationsparameter - kgC/m^2 Stammoberfläche
      public float stammRespirationrateMethode2;

   // Allokation
      public float feinWurzelAllocationsRate;
      public float feinWurzelUeberlebensrate;
      public float nadelWachstumEffizinzanpassungKoeffizient_A;
      public float nadelWachstumEffizinzanpassungKoeffizient_B;

      public float nadelWachstumAssimilateMinimalRate;
      public float nadelWachstumAssimilateMaximalRate;
      public float nadelWachstumAssimilateEinflussFaktorExp;

      public float astWachstumRate;
      public float astUeberlebensRate;
      public float grobWurzelWachstumRate;
      public float grobWurzelUeberlebensRate;
   // wachstum
      public float hoehenZuwachsKorrelationsKoeffizient  ;
      public float hoehenZuwachsVariationsKoeffizient ;


      // unechte Formzahl f: für vol = f * pi/4 * d^2 * h
      // die Formzahl wird errechnet
      public float formzahl;
      public float stammDichte;
      const float cQuote;

      public  int verwendetePhotosyntheseFunktion;
      public  float beerLambertKoeffizient_A;
      public  float beerLambertKoeffizient_B;
      public  float pfreundtKoeffizient_A;
      public  float pfreundtKoeffizient_B;
      public  float maekelaKoeffizient_A;
      public  float maekelaKoeffizient_B;
      public  float maekelaKoeffizient_C;

      public  float lichtTransmissionsKoeffizient;
// q wert betafunktion (b-z)^q // b wert betafunktion (z -a)^p
      public  float beta_q = 2.5;
      public  float beta_p = 2.5;
```

```
      // a1 = untere Grenze beta funktion für Nadeldichte Jahrgang 1
      // b7 = obere Grenze beta Funktion für Nadeldichte Jahrgang 7

      public  float a1 = 0.0f;
      public  float a2 = 0.0f;
      public  float b2 = 0.0f;
      public  float a3 = 0.0f;
      public  float b3 = 0.0f;
      public  float a4 = 0.0f;
      public  float b4 = 0.0f;
      public  float a5 = 0.0f;
      public  float b5 = 0.0f;
      public  float a6 = 0.0f;
      public  float b6 = 0.0f;
      public  float a7 = 0.0f;
      public  float b7 = 0.0f;

//-------------------------------------------------------------------------
// Bereich 2: editierbare Eigenschaften werden durch den Konstruktor gesetzt
//
//-------------------------------------------------------------------------

      public @Editable int baumArt = 0;
      public @Editable int baumAlter = 0;
      public @Editable float baumHoehe = 0.0f;
      public @Editable float d13 = 0.0f;

// wird berechnet durch baumHoehe * kaRelativ

      public @Editable float kronenAnsatz = 0.0f;
      public @Editable float maximalerKronenRadius = 0.0f;
      public @Editable float cMasseGesamt = 0.0f;

// Bereich 3: editierbare Eigenschaften welche durch das Programm während der Simulation
   berechnet werden

// gesamt Nadelmasse und Nadelmassen der Nadeljahrgänge 1.7 C-Trockenewicht in kgC
      public @Editable float nadelMasseG = 0.0f;
      public @Editable float nadelMasse1 = 0.0f;
      public @Editable float nadelMasse2 = 0.0f;
      public @Editable float nadelMasse3 = 0.0f;
      public @Editable float nadelMasse4 = 0.0f;
      public @Editable float nadelMasse5 = 0.0f;
      public @Editable float nadelMasse6 = 0.0f;
      public @Editable float nadelMasse7 = 0.0f;

      // c-Trockengewicht der Komopartimente kgC
      public @Editable float astMasseG = 0.0f;
      public @Editable float stammMasseG = 0.0f;
      public @Editable float feinWurzelMasseG = 0.0f;
      public @Editable float grobWurzelMasseG = 0.0f;

   // gesamte erzeugte Trockenmasse kgC
      public @Editable float bruttoAssimilateSpeicher = 0.0f;

   // Assimilate nach Erhaltungsatmung aber vor Wachstumsrespiration kgC
      public @Editable float nettoAssimilateSpeicher = 0.0f;

// Respiration der Kompartimente im aktuellen Jahr kgC
      public @Editable float wachstumsRespiration = 0.0f;
      public @Editable float nadelRespiration = 0.0f;
      public @Editable float astRespiration  = 0.0f;
      public @Editable float wurzelRespiration = 0.0f;
      public @Editable float stammRespiration = 0.0f;
      public @Editable float gesamtRespiration = 0.0f;

// Neue Masse der Kompartimente im aktuellen Jahr kgC
      public @Editable float feinWurzelMasseNeu = 0.0f;
      public @Editable float nadelMasseNeu = 0.0f;
      public @Editable float astMasseNeu = 0.0f;
      public @Editable float grobWurzelMasseNeu = 0.0f;
      public @Editable float stammMasseNeu = 0.0f;

// Abgestorbene Masse der Kompartimente im neuen Jahr kgC
      public @Editable float astMasseTot = 0.0f;
      public @Editable float stammMasseTot = 0.0f;
      public @Editable float feinWurzelMasseTot = 0.0f;
```

```
      public @Editable float grobWurzelMasseTot = 0.0f;

      public @Editable float produktionsEffizienz = 0.0f;
      public @Editable float relativeProduktionsEffizienz = 0.0f;

// temporärer Zwischenspeicher zur Ausgabe von Charts
      public @Editable float tmpDouble = 0.0f;

// Bereich 4:  nicht editierbare Variablen
      public float stammOberflaeche = 0.0f;
//    public float stammVolumen = 0.0f;
      public int kronenAnsatzSegment = 0;
      public int countSegmente = 0;
      public float[] betafunktion_korrekturintegral_pro_jahrgang  =
         {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};

      public float[][] kronenStuetzPunkte  = new float[2][30];


// Bereich 5 Variablen für die Verbindung zur Forester Description

// Index der Arraylist der Foresterconnection.ForesterDescription
      public int descriptionIndex = -1;
// entspricht foresterconnection.ForesterTree.id wenn in der Description gesetzt

      public int id = 0;   //

//---------------------------------------------------------------------------
//                Methoden
//---------------------------------------------------------------------------
// Konstructor der Baumklasse
   Baumklasse(int age, float h, float bhd, int art, float ka)
      {
         super();
         baumAlter = age;
         baumArt = art;
         System.arraycopy(Definitionen.KOMPARTIMENT_ANTEILE[this.baumArt],0,
            kompartimentAnteile, 0, 5);
         System.arraycopy(Definitionen.NADELJAHRGANG_UEBERLEBENSRATE[this.baumArt],0,
            nadelJahrgangUeberlebensrate, 0, 7);
//System.arraycopy(Definitionen.NADELMASSE_VERTEILUNG_KORREKTURFAKTOR[this.baumArt],0,
   nadelMasseVerteilungKorrekturfaktor, 0, 7);
         System.arraycopy(Definitionen.NADELJAHRGANG_REL_PHOTO_KAPAZITAET[this.baumArt],
            0,  nadelJahrgangReativePhotosyntheseKapazitaet, 0, 7);
         System.arraycopy(Definitionen.MITTELSTAMM_ALTERS_HOEHENKURVE[this.baumArt], 0,
            mittelStammAltersHoehenkurve, 0, 7);
         System.arraycopy(Definitionen.FORMZAHL_KOEFFITZIENTEN[this.baumArt], 0,
            formzahl_Koeffizienten, 0, 7);
         System.arraycopy(Definitionen.KRONENFORM_PARAMETER[this.baumArt], 0,
            kronenformParameter, 0, 4);
         nadelMasseVerteilungKorrekturfaktor =
               Definitionen.NADELMASSE_VERTEILUNG_KORREKTURFAKTOR[this.baumArt];

         baumHoehe = h; //Gesamthöhe des Baumes
         d13 = bhd; // Brusthöhendurchmesser D13
         kronenAnsatz = ka;   // Kronenansatzpunkt in Relation der Baumhöhe angegeben
         maximalerKronenRadius = Definitionen.KRONENDURCHMESSERDEFAULTWERT[baumArt];

         maxNadeljahrgang = Definitionen.MAX_NADELJAHRGANG[this.baumArt];
         pfreundtKoeffizient_A = Definitionen.PFREUNDT_COEFF_A[this.baumArt];
         pfreundtKoeffizient_B = Definitionen.PFREUNDT_COEFF_B[this.baumArt];
         beerLambertKoeffizient_A = Definitionen.BEER_LAMBERT_COEFF_A[this.baumArt];
         beerLambertKoeffizient_B = Definitionen.BEER_LAMBERT_COEFF_B[this.baumArt];
         maekelaKoeffizient_A = Definitionen.MAEKELA_COEFF_A[this.baumArt];
         maekelaKoeffizient_B = Definitionen.MAEKELA_COEFF_B[this.baumArt];
         maekelaKoeffizient_C = Definitionen.MAEKELA_COEFF_C[this.baumArt];
         verwendetePhotosyntheseFunktion =
            Definitionen.F_FUNKTION_BAUMARTEN_DEFAULT[this.baumArt];
         lichtTransmissionsKoeffizient =
            Definitionen.LICHT_TRANSMISSIONS_KOEFFIZIENT[this.baumArt];
         produktionsRateUnbeschattet =
            Definitionen.PRODUKTIPONSRATE_UNBESCHSATTET[this.baumArt];
         nadelRespirationRate = Definitionen.NADEL_RESPIRATION_RATE[this.baumArt];
         holzteileRespirationKoeffizient_A =
            Definitionen.RESPIRATION_COEFF_A[this.baumArt];
         holzteileRespirationKoeffizient_B =
            Definitionen.RESPIRATION_COEFF_B[this.baumArt];
```

```
        stammRespirationrateMethode2 =
           Definitionen.STAMM_RESPIRATIONRATE_METHODE_2[this.baumArt];
        astRespirationrateMethode2  =
           Definitionen.AST_RESPIRATIONRATE_METHODE_2[this.baumArt];
        wurzelRespirationrateMethode2  =
           Definitionen.WURZEL_RESPIRATIONRATE_METHODE_2[this.baumArt];
        wachstumsRespirationRate =
           Definitionen.WACHSTUMSRESPIRATION_RATE[this.baumArt];
        feinWurzelAllocationsRate =
           Definitionen.FEIN_WURZEL_ALLOCATIONS_RATE[this.baumArt];
        feinWurzelUeberlebensrate =
           Definitionen.FEIN_WURZEL_UEBERLEBENS_RATE[this.baumArt];
        nadelWachstumEffizinzanpassungKoeffizient_A =
           Definitionen.NADEL_WACHSTUM_EFFIZINZANPASSUNG_COEFF_A[this.baumArt];
        nadelWachstumEffizinzanpassungKoeffizient_B =
           Definitionen.NADEL_WACHSTUM_EFFIZINZANPASSUNG_COEFF_B[this.baumArt];
        nadelWachstumAssimilateMinimalRate =
           Definitionen.NADEL_WACHSTUM_ASSIMILATE_MINIMALRATE[this.baumArt];
        nadelWachstumAssimilateMaximalRate =
           Definitionen.NADEL_WACHSTUM_ASSIMILATE_MAXIMALRATE[this.baumArt];
        nadelWachstumAssimilateEinflussFaktorExp =
           Definitionen.NADEL_WACHSTUM_ASSIMILATE_EINFLUSS_FAKTOR_EXP[this.baumArt];

        astWachstumRate = Definitionen.AST_WACHSTUM_RATE[this.baumArt];
        astUeberlebensRate = Definitionen.AST_UEBERLEBENS_RATE[this.baumArt];
        grobWurzelUeberlebensRate =
           Definitionen.GROB_WURZEL_UEBERLEBENSRATE_RATE[this.baumArt];
        hoehenZuwachsKorrelationsKoeffizient  =
           Definitionen.HOEHENZUWACHS_CORR_COEFF[this.baumArt];
        hoehenZuwachsVariationsKoeffizient  =
           Definitionen.HOEHENZUWACHS_VARIATIONS_COEFF[this.baumArt];
        stammDichte = Definitionen.STAMMDICHTE[this.baumArt];
        formzahl = kalkuliereFormzahl();
        cQuote = Definitionen.C_QUOTE;

        stammMasseG = kalkuliereStammCMasse();
        cMasseGesamt =  stammMasseG / this.kompartimentAnteile[Definitionen.STAMM];
              nadelMasseG = cMasseGesamt *
        this.kompartimentAnteile[Definitionen.NADEL];
        astMasseG = cMasseGesamt * this.kompartimentAnteile[Definitionen.AST];
        feinWurzelMasseG = cMasseGesamt *
           this.kompartimentAnteile[Definitionen.FEINWURZEL];
        grobWurzelMasseG = cMasseGesamt *
           this.kompartimentAnteile[Definitionen.GROBWURZEL];
        stammOberflaeche = this.kalkuliereStammOberflaeche();
        grobWurzelWachstumRate = grobWurzelMasseG / (grobWurzelMasseG + stammMasseG);
        // initialisiert ka und höhe der nadeljahrgänge
        initialisiereBetaFunctionValues();
        //  startwerte für die NadelJahrgangsmassen setzen
        initialisiereNadeljahrgangCMassen();
           ermittleKronenStuetzpunkteNachKronenform();
     }


  private void initialisiereNadeljahrgangCMassen()
  // initialisiert die Jahrgangs-Nadelmassen anhand der GesamtNadelBiomasse und der
  NadelJahrgang-Überlebensraten
     {
        float summe = 1.0f, a = 1.0f;
        a = 1.0f;
        for (int i=1; i < this.maxNadeljahrgang; i++) {
           a = a * this.nadelJahrgangUeberlebensrate[i];
           summe += a;
        }
        for (int i=1; i <= this.maxNadeljahrgang; i++) {
           switch (i){
     case 1: nadelMasse1 = nadelMasseG / summe;  break;
     case 2: nadelMasse2 = nadelMasse1 * this.nadelJahrgangUeberlebensrate[1]; break;
     case 3: nadelMasse3 = nadelMasse2 * this.nadelJahrgangUeberlebensrate[2]; break;
     case 4: nadelMasse4 = nadelMasse3 * this.nadelJahrgangUeberlebensrate[3]; break;
     case 5: nadelMasse5 = nadelMasse4 * this.nadelJahrgangUeberlebensrate[4]; break;
     case 6: nadelMasse6 = nadelMasse5 * this.nadelJahrgangUeberlebensrate[5]; break;
     case 7: nadelMasse7 = nadelMasse6 * this.nadelJahrgangUeberlebensrate[6]; break;
           }
        }
     }
```

```
public float addiereNadelJahrgangMassen()
      // Berchnet die Gesamt-Nadelmasse durch Addition der Jahrgangs-Nadelmassen
      {
         float summe = 0.0f;
         for (int i=1; i <= this.maxNadeljahrgang; i++) {
            switch (i) {
            case 1: summe += this.nadelMasse1 ; break;
            case 2: summe += this.nadelMasse2 ; ;break;
            case 3: summe += this.nadelMasse3 ; break;
            case 4: summe += this.nadelMasse4 ; break;
            case 5: summe += this.nadelMasse5 ; break;
            case 6: summe += this.nadelMasse6 ; break;
            case 7: summe += this.nadelMasse7 ; break;
            }
         }
            return summe;
      }

      public float addiereKompartimentMassen()
      // Addiert die Kopartimentmassen
      {
      float summe = 0.0f;
      summe = this.nadelMasseG + this.astMasseG + this.stammMasseG +
         this.feinWurzelMasseG + this.grobWurzelMasseG;
      return summe;
      }


public void initialisiereBetaFunctionValues()
// initialisiert die Startwerte für Nadelverteilung: a werte werden für alle
   Nadeljahrgänge auf den Kronenansatz gesetzt, b werte werden der Baumhöhe der
   vergangenen Wachstumsperioden angeglichen, Baumhöhe t-1 = Baumhöhe t -
   durchschnittlicher Zuwachs(bis zu Zeitpunkt t)
{
   for (int i=1; i <= this.maxNadeljahrgang; i++) {
     setBetaFunctionValue(i, kronenAnsatz, baumHoehe - ((i-1)*(baumHoehe / baumAlter)));
   }
}

public void updateBetaFunctionValues()
// aktualisiert die Betafunktionswerte
      {
      if( this.maxNadeljahrgang >= 1) setBetaFunctionValue(1, this.a1, this.b1);
      if( this.maxNadeljahrgang >= 2) setBetaFunctionValue(2, this.a2, this.b2);
      if( this.maxNadeljahrgang >= 3) setBetaFunctionValue(3, this.a3, this.b3);
      if( this.maxNadeljahrgang >= 4) setBetaFunctionValue(4, this.a4, this.b4);
      if( this.maxNadeljahrgang >= 5) setBetaFunctionValue(5, this.a5, this.b5);
      if( this.maxNadeljahrgang >= 6) setBetaFunctionValue(6, this.a6, this.b6);
      if( this.maxNadeljahrgang >= 7) setBetaFunctionValue(7, this.a7, this.b7);
      }

private void setBetaFunctionValue(int nadeljahrgang, float valueA, float valueB )
// setzt die Nadelgrenzen pro jahrgang
{
   if ( nadeljahrgang > 0 && nadeljahrgang <= this.maxNadeljahrgang){
      // Kronenansatz ist minamaler Wert für a
      if(valueA < this.kronenAnsatz) valueA = this.kronenAnsatz;
      //baumhöhe ist maximaler wert für a
      if (valueA > this.baumHoehe) valueA = this.baumHoehe;
      // Kronenansatz ist minamaler Wert für b
      if(valueB < this.kronenAnsatz) valueB = this.kronenAnsatz;
      if (valueB > this.baumHoehe) valueB = this.baumHoehe;
            switch (nadeljahrgang) {
            case 1: this[a1] = valueA; this.b1 = valueB; break;
            case 2: this.a2 = valueA; this.b2 = valueB; break;
            case 3: this.a3 = valueA; this.b3 = valueB; break;
            case 4: this.a4 = valueA; this.b4 = valueB; break;
            case 5: this.a5 = valueA; this.b5 = valueB; break;
            case 6: this.a6 = valueA; this.b6 = valueB; break;
            case 7: this.a7 = valueA; this.b7 = valueB; break;
            } //  end switch
         }  // end if
      } //end method
```

```
public float kalkuliereNadelMasseVerteilungKorrekturTerm()
// berechnet den Additionsterm für die Korrektur der Distanz der beschattenden
   Nadelmasse. Berücksichtigt den Korrekturfaktor für ihre Verteilung im Kronenraum

{
   return ((this.maximalerKronenRadius * this.maximalerKronenRadius) /
      this.nadelMasseVerteilungKorrekturfaktor);
}

public void ermittleKronenStuetzpunkteNachKronenform()
//berechnet Werte für das Zeichnen der Krone nach Kronenform aus Pretsch 2001
{
      float height = height = (this.baumHoehe - this.kronenAnsatz) ;
      float i0 = height * this.kronenformParameter[0];
      float a = this.maximalerKronenRadius / Math.pow(i0, this.kronenformParameter[1]);
      float b =   this.kronenformParameter[2];
      float radiusKronenansatz = this.maximalerKronenRadius *
         this.kronenformParameter[3];
      float d = (radiusKronenansatz - this.maximalerKronenRadius) / (height - i0);
      float c = this.maximalerKronenRadius - (d*i0);

      //Schattenkrone
      this.kronenStuetzPunkte[0][0] = this.kronenAnsatz;
      this.kronenStuetzPunkte[1][0] = radiusKronenansatz;

      this.kronenStuetzPunkte[0][1] = this.baumHoehe - i0;
      this.kronenStuetzPunkte[1][1] = c+ i0*d;
      float distanz = 0.0f;
      float anzahlPunkte = this.kronenStuetzPunkte[0].length;
      for ( int i = 2; i < anzahlPunkte ; i++){
            distanz = ( ( i0 / (anzahlPunkte - 2))  * ( (anzahlPunkte -1)- i));
            this.kronenStuetzPunkte[0][i] = this.baumHoehe - distanz ;
            this.kronenStuetzPunkte[1][i] = Math.max(0.01f, a * Math.pow(distanz, b));
      }
}

private float kalkuliereFormzahl()
// berechnet die Formzahl für den Baum
{
   float formzahl = 1.0f;
   float d = this.d13 * 100;
   formzahl = formzahl_Koeffizienten[0]
   + formzahl_Koeffizienten[1]/ d
   + formzahl_Koeffizienten[2]/ this.baumHoehe
   + formzahl_Koeffizienten[3] / (d * d)
   + formzahl_Koeffizienten[4]/ (d * this.baumHoehe)
   + formzahl_Koeffizienten[5]/ ((d * this.baumHoehe)*( d *this.baumHoehe))
   + formzahl_Koeffizienten[6] * Math.log(d) * Math.log(d);
   return formzahl;
}

private float kalkuliereStammCMasse()
// berechnet die Stammmasse nach Höhe, d113 und Formzahl
{
      return this.d13 * this.d13 * this.baumHoehe * Math.PI / 4 * this.stammDichte *
   this.cQuote * this.formzahl;
}

public float kalkuliereStammOberflaeche()
{
   return this.d13 * this.baumHoehe * Math.PI  * Math.sqrt(this.formzahl);
}

public float altersHoehenkurveMittestamm(float alter)
{
// berechnet die Höhe des Bestandesmitelstammes in abhängigkeit des alters
   float a = 0.0f, b= 0.0f, c= 0.0f;
   if (alter < this.mittelStammAltersHoehenkurve[0]){
      a = this.mittelStammAltersHoehenkurve[1];
      b= this.mittelStammAltersHoehenkurve[2];
      c= this.mittelStammAltersHoehenkurve[3];}
   else{
      a = this.mittelStammAltersHoehenkurve[4];
      b= this.mittelStammAltersHoehenkurve[5];
```

```
      c= this.mittelStammAltersHoehenkurve[6];}
      return a + b * alter + c * alter * alter;
}

public  float kalkuliereStammDurchmesser()
   {
      return  Math.sqrt(this.stammMasseG / ( this.baumHoehe * Math.PI / 4 *
   this.stammDichte * cQuote * formzahl)) ;

   }

public float kalkuliereBaumHoehenZuwach(float sigmaH)
// berechnet den höhenzuwacht von alter t zu t +1 auf basis dreier komponenten
//1. der zuwachs des mittelstammes
//2. +- zuwachs in abhängigkeit der bisherigen höhendifferenz zum mittelstamm des baumes

//3. +- eine Zufallskomponente
{
   float ht = this.altersHoehenkurveMittestamm(this.baumAlter);
   float mittelStammZuwach = this.altersHoehenkurveMittestamm(this.baumAlter +1)-  ht;
// Variationskoeffizient = Standardabweichung durch Mittelwert
//Rückrechnung auf Standardabweichung
   float sdeltah = this.hoehenZuwachsVariationsKoeffizient  * mittelStammZuwach ;

   float hoehenAbhaengigerZuwachs =  sdeltah / sigmaH * ( this.baumHoehe - ht);
   float zufallsZuwachs = random(-sdeltah, sdeltah);

   return mittelStammZuwach + this.hoehenZuwachsKorrelationsKoeffizient  *
      hoehenAbhaengigerZuwachs + (1-this.hoehenZuwachsKorrelationsKoeffizient  ) *
      zufallsZuwachs;
   }

public float kalkuliereKronenRadiusZuwachs()
// berechnet den Zuwachs der Kronenausdehnung (Radius)
// nimmt einen durchschnittliche jährlichen Zuwachs
{
   return (this.maximalerKronenRadius / this.baumAlter );
 }

public void kalkuliereProduktionsEffizienz()
// Berechnung der Produktionseffizienz pro Kilogramm Nadelmasse
{
   if (this.nadelMasseG > 0.0f){
      this.produktionsEffizienz = this.bruttoAssimilateSpeicher / this.nadelMasseG;
   }
   else {this.produktionsEffizienz = 0.0f; }
}

public void kalkuliereRelativeProduktionsEffizienz()
// Berechnung der Produktionseffizienz im Verhältnis zu unbeschatteten Verhältnissen
{
   if (this.nadelMasseG > 0.0f){
      this.relativeProduktionsEffizienz =  this.bruttoAssimilateSpeicher
         /this.nadelMasseG / this.produktionsRateUnbeschattet ;
   }
   else {relativeProduktionsEffizienz = 0.0f; }
}

public float kalkulierefFeinwurzelWachstum()
   {
      return Math.max(Math.min (this.bruttoAssimilateSpeicher *
         feinWurzelAllocationsRate, this.nettoAssimilateSpeicher), 0.0f);
   }
public float kalkuliereNadelWachstumNAchOle()
// Nadelwachstum Methode 1
   {
      return this.nettoAssimilateSpeicher * Math.max
      (this.nadelWachstumEffizinzanpassungKoeffizient_A +
      this.nadelWachstumEffizinzanpassungKoeffizient_B *this.produktionsEffizienz, 0.0);
   }

public float kalkuliereNadelWachstum()
// Berechnet Nadelwachstum mit Methode 2
   {
       return this.nettoAssimilateSpeicher * Math.min
          (this.nadelWachstumAssimilateMinimalRate  /
          Math.pow(this.relativeProduktionsEffizienz,
```

```
            this.nadelWachstumAssimilateEinflussFaktorExp),
            this.nadelWachstumAssimilateMaximalRate);
   }

public float kalkuliereAstWachstum()
// Berechnung der neuen Astmasse im aktuellen Jahr
   {
      return  this.nadelMasseNeu * this.astWachstumRate ;
   }

public float kalkuliereStammHoehenWachstum(float sigmaH)
// Berechnung des Höhenwachstums

   {
      return  this.kalkuliereBaumHoehenZuwach(sigmaH); ;
   }

public float kalkuliereGrobwurzelWachstum(float death)
// Berechnung der neuen Grobwurzelmasse im aktuellen Jahr
   {
      if(this.nettoAssimilateSpeicher - death > 0.0f ) {
         return  ((this.nettoAssimilateSpeicher - death) * (
            this.grobWurzelWachstumRate)) + death ;
      }
      else {
         return  Math.max(this.nettoAssimilateSpeicher,0.0f);
      }
   }

}
// Ende der Baumklasse (baum.rgg).
```

10.2.4 Code segment.rgg

```
import de.grogra.ext.cpfg.*;
import static java.lang.Math.*;

//------------------------------------------------------------------------
//                      Segmentklasse
//------------------------------------------------------------------------
class Segmentklasse extends F
{

   // temporärer Zwischenspeicher zur Ausgabe von Charts
   public @Editable float tmpDoubleX = 0.0f;
   public @Editable float tmpDoubleY = 0.0f;

   public @Editable float nadelMasse1 = 0.0f;
   public @Editable float nadelMasse2 = 0.0f;
   public @Editable float nadelMasse3 = 0.0f;
   public @Editable float nadelMasse4 = 0.0f;
   public @Editable float nadelMasse5 = 0.0f;
   public @Editable float nadelMasse6 = 0.0f;
   public @Editable float nadelMasse7 = 0.0f;
   public @Editable float nadelMasseGesamt = 0.0f;
   public @Editable float externeBeschattung = 0.0f;
   public @Editable float selbstBeschattung = 0.0f;
   public @Editable float gesamtBeschattung = 0.0f;
   public @Editable float  assimilateSpeicher = 0.0f;
   public @Editable float  nadelRespiration = 0.0f;
   public @Editable float  top = 0.0f;
   public float relativeProduktionsQuoteBeschattet = 0.0f;

   Segmentklasse(float len){
      super(len)
   }


// Methoden ************************************


   public float kalkuliereProduktionsQuoteBeschattet0(float a, float b)
   //berechnet die Produktionsrate (in Abhängigkeit der beschattenden Biomasse) relativ
   zu unbeschatteten Bedingungen. Gefittete Funktion nach Beer-Lambert-Ansatz
   {
      float tmp = 0.0f;
      tmp = Math.max(a*Math.exp(-b*this.gesamtBeschattung),0.0f);
      return tmp;
   }


   public float kalkuliereProduktionsQuoteBeschattet1(float a, float b)
   //berechnet die Produktionsrate(in Abhängigkeit der Beschattenden Biomasse) relativ
   zu unbeschatteten bedingungen. Funktion nach Sloboda & Pfreundt (1989)
   {
      float tmp = 0.0f;
      tmp = Math.max(1 - a * pow(this.gesamtBeschattung,b),0.0f);
   // println("" + this.gesamtBeschattung + " " + tmp);
      return tmp;
   }


public float kalkuliereProduktionsQuoteBeschattet2(float a, float b, float c)
// berechnet die Produktionsrate  (in Abhängigkeit der beschattenden Biomasse) relativ
   zu unbeschatteten Bedingungen. Funktion nach F-Funktion von Mäkela beschrieben in
   Kellomäki et al. 1980. Achtung: nur experimentell. Benötigt Biomassewerte pro Hektar
   (ungewichtet)
{
   float tmp = 0.0f;
   tmp = (c/ (1 + Math.exp(-2* (Math.log10(this.gesamtBeschattung) - a )/ b))) + 1 - c;
   // println("" + this.gesamtBeschattung + " " + tmp);
      return tmp;
}
```

```
public float calculateSegmentBetaValue(float a, float b,float base, float top, float
   beta_q, float beta_p)
   {
// kalkuliert den absoluten Wert der Betaverteilung (ncht normiert für ein sSegment duch
   lineare iInterpolation zwischen p wWert der Basis und p Wert top
      float p = 0.0f;
      float length = 0.0f;

      if (b < base) return 0.0f;   //ff  wenn Grenzwerte der bBetaverteilung mitten im
   Segment liegen
      if (b < top) top = b;
      if (a > top) return 0.0f;
      if (a > base) base = a;
      length = top - base;
      if ( length == 0.0) return 0.0f;
      p = ( ( ( pow((base - a) , beta_p) * pow((b - base), beta_q))) +
          ( ( pow((top  - a) , beta_p) * pow((b - top ), beta_q))) ) / 2   ;
      return p * length;
   }


}
// Ende segment.rgg
```

10.2.5 Code httpstartup.rgg

```
//---------------------------------------------------------------------------
//         Klasse mit Methoden für den Modellstart über den http-Server
//---------------------------------------------------------------------------

import de.grogra.ext.cpfg.*;
import static java.lang.Math.*;
import java.io.*;
import de.grogra.imp.net.*;
import de.grogra.util.*;
import de.grogra.pf.registry.*;
import java.util.regex.*;
import java.util.*;
import forester.http.formdata.*;
import forester.description.*;
import forester.groimp.*;

public static HttpMultipartFormData verarbeiteHttpInput(Object info){

   HttpResponse resp = (HttpResponse) info;
   passBoundary ();
   String contentType   = (resp.getRequest()).getHeaderField("content-type");
   // Auslesen der boundary
   String boundary = contentType.substring(contentType.indexOf("boundary=")+9);
   // Auslesen des Content-Bereichs
   String content = new String((resp.getRequest()).getContent(), "UTF-8");
   println(content);
   //Auslesen der Formfelder aus dem content Bereich des geposteten HTTP-Stream
   HttpMultipartFormData n1 = new HttpMultipartFormData(content, "--" + boundary);
   return n1;
}

public static String createFilename(ForesterDescription foDescIn){
   String tmp;
   tmp = ( foDescIn.getFileName() + "_" + System.currentTimeMillis() + ".txt");
   return tmp;
}

public static String createAnswer(Object info, HttpMultipartFormData n1,
   ForesterDescription foDescIn ){
   String server  = "http://ufgb985:58080";
   String modelout = "deliverscene.gsz";
   StringBuffer buf = new StringBuffer ();

   String filename = "";
   HttpResponse resp = (HttpResponse) info;

   // Erstellen der Http Antwort
   if ( foDescIn == null){
      // Fall 1 +2 Keine Forester Description wurde mitgesendet
      if (n1.getFieldContent("form") == null)
         {   // Antwort an Forester server
            buf.append("url=\n");
            buf.append("status=wrong\n");
            resp.setContent ("text/text", "UTF-8", buf.toString());
         }
         else
         {
            // Antwort an Browser
            buf.append("<html><body>\nurl=\n<br>status=wrong\n<br>keine Forester-
               Description gesendet<br></body></html>");
            resp.setContent ("text/html", "UTF-8", buf.toString());
         }
      }
   else
      {
      // Fall 3 + 4 Forester Description wurde mitgesendet
      filename = ( foDescIn.getFileName() + "_" + System.currentTimeMillis() + ".txt");
         if ((n1.getFieldContent("form")) == null)
         {
         // Antwort an Forester
            buf.append("url=" + server + "/open?" + modelout + ";" + filename + "\n");
            buf.append("status=ok\n");
            resp.setContent ("text/text", "UTF-8", buf.toString());
         }
         else
```

```
         {
          //Antwort an Browser
            buf.append("<html><body>\n");
            buf.append( "<form action=\"" + server + "/open?" + modelout + "\"
               method=\"post\" enctype=\"multipart/form-data\" >");
            buf.append("<input name=\"datei\" value=\"" + filename + "\"
               type=\"text\">");
            buf.append("<input name=\"submit\" value=\"Datei anfordern\"
               type=\"submit\">");
            buf.append("</form></body></html>");
            resp.setContent ("text/html", "UTF-8", buf.toString());
         }
      }
   // Senden der Antwort
   resp.send (true);
   return filename;
}

public static ForesterDescription
      readForesterDescriptionFormfield(HttpMultipartFormData n1)
// erstellt das Forester description Objekt anhand wenn eine Datei im Upload
   {
      for(int i = 0; i < n1.getNumberOfFields(); i++) {
         if(n1.fields[i].getType() == "file" && ! n1.fields[i].getFileName().equals(""))
         {
            return new ForesterDescription(n1.fields[i].getFieldContent(),
               n1.fields[i].getFileName() );
         }
      }
      return null;
   }


public static int readSimulationPeriode(HttpMultipartFormData n1, String key,
            int standardSimulationsPeriode )
// Auslesen der Zahl der Simulationsperioden aus einen mitgesendeten Formularfeld:
// name des Formularfeldes wird der Methode als key übergeben.
{
      // Simulationsperiode auslesen
      String tmp;
      // Defaultsetzung mit Standardwert als Parameter übergeben
      int periode = standardSimulationsPeriode;
      tmp =  n1.getFieldContent(key);
      if (tmp != null){
         Scanner s = new Scanner(tmp);
         if (s.findWithinHorizon("(\\d+)", 0) != null)
         {
            MatchResult result = s.match();
            for (int i=1; i<=result.groupCount(); i++) {
               periode =  Integer.valueOf(result.group(i));
            }
         }
      }
      return periode;
   }
// ende httpstartup.rgg
```

10.2.6 Code ausgabe.rgg

```
import static java.lang.Math.*;

const DatasetRef nadeldichte = new DatasetRef ("Nadeldichte");
const DatasetRef beschattung = new DatasetRef ("Beschattung");
const DatasetRef produktionsquote = new DatasetRef ("Produktionsquote");
const DatasetRef produktionsStaerke = new DatasetRef ("Produktionsstärke");
const DatasetRef tmp = new DatasetRef ("tmp");

//-----------------------------------------------------------------------------
//       CHARTS
//-----------------------------------------------------------------------------

public static void zeichneNadeldichteChart(int jahrgang)
// Chart der Nadelmassen pro Segment
{
   [
      s:Segment ::> {
         s.tmpDoubleX = (( s.top - s.base) /2) + s.base;
         switch(jahrgang){
            case 1: s.tmpDoubleY = s.nadelMasse1;    break;
            case 2: s.tmpDoubleY = s.nadelMasse2;    break;
            case 3: s.tmpDoubleY = s.nadelMasse3;    break;
            case 4: s.tmpDoubleY = s.nadelMasse4;    break;
            case 5: s.tmpDoubleY = s.nadelMasse5;    break;
            case 6: s.tmpDoubleY = s.nadelMasse6;    break;
            case 7: s.tmpDoubleY = s.nadelMasse7;    break;
            default: s.tmpDoubleY = s.nadelMasseGesamt; break;
         }
      }
   ]
   zeichneSegmentChart(nadeldichte);
}


public static void zeichneRealtiveProduktionsQuoteChart(int baum, boolean ka)
// Chart der relativen Produktionsstärke
{
   [
      s:Segment -ancestor-> b:Baum, ( s.bnr == baum) ::> {
         s.tmpDoubleX = (( s.top - s.base) /2) + s.base;
         s.tmpDoubleY = s.relativeProduktionsQuoteBeschattet;
      }
   ]
   zeichneSegmentChartBaum(produktionsquote, baum, ka);

}

public static void zeichneProduktionsStaerkeChart(int baum, boolean ka)
// Chart der Produktionsstärke
{
   [
      s:Segment -ancestor-> b:Baum, ( s.bnr == baum ) ::> {
         s.tmpDoubleX = (( s.top - s.base) /2) + s.base;
         s.tmpDoubleY = s.assimilateSpeicher / ( s.top - s.base);
      }
   ]
   zeichneSegmentChartBaum(produktionsStaerke, baum, ka);
}

public static void zeichneBeschattungChart(String typ)
// Chart der Beschattung
{
   [
      s:Segment ::> {
         s.tmpDoubleX = (( s.top - s.base) /2) + s.base;
         if(typ.equals("bs")){ s.tmpDoubleY = s.selbstBeschattung;}
            else {
               if (typ.equals("be"))
                     s.tmpDoubleY = s.externeBeschattung;
               else  s.tmpDoubleY = s.gesamtBeschattung;
            }
      }
   ]
   zeichneSegmentChart(beschattung);
}
```

```
public static void zeichneBeschattungChart(String typ, int baum, boolean ka)
// Chart der Beschattung für einen Baum
{
   [
      s:Segment, (s.bnr==baum)::> {
         s.tmpDoubleX = (( s.top - s.base) /2) + s.base;
         if(typ.equals("bs")){ s.tmpDoubleY = s.selbstBeschattung;}
            else {
               if (typ.equals("be"))
                     s.tmpDoubleY = s.externeBeschattung;
               else  s.tmpDoubleY = s.gesamtBeschattung;
            }
      }
   ]
   zeichneSegmentChartBaum(beschattung, baum, ka);
}

private static void zeichneSegmentChart(DatasetRef tmp)
// Zeichnen eines Charts aus der Variable tempDouble für alle Bäume
{
   int maxseg = 0;
   int countbaum = 0;
   [           // regel zum bestimmen der maximalen segmentzahl
      {int nr = 0;}
      b:Baum ::> {
         countbaum ++;
         nr = ((Segment) b.getLastChild() ).snr;
         if(maxseg <= nr) maxseg = nr;
      }
   ]
   // erzeugen eines 2-d array nach baum und segment
   float[][][] val = new float[countbaum][maxseg][2];
   int[] nrs = new int[countbaum];
   // initalisieren des arrays
   for(int i=0; i< countbaum; i++)
   for (int j=0 ; j< maxseg; j++){
      val[i][j][0]=0.0f;
      val[i][j][1]=0.0f;
   }
   [     // regel zum auslesen der werte
      {int i = 0;}
      b:Baum ::> {
         for ((* s:Segment, (s.bnr == b.bnr) *)) {
            val[i][s.snr-1][0]=s.tmpDoubleX;
            val[i][s.snr-1][1]=s.tmpDoubleY;
         }
         nrs[i]=b.bnr;
         i++;
      }
   ]

   // initialisieren des charts
   tmp.clear ();

   Dataseries dt;
   //setzen der Datenreihen
   for (int i = 0; i < countbaum; i++) tmp.setColumnKey(i, "B" + nrs[i]);
   // setzen der datenpunkte
   for (int j = 0; j < maxseg; j++){
      dt = tmp.addRow();

      for (int i = 0; i < countbaum; i++)
      {
      // println("ausgabe21:" + i + "  " + val[i][j][0] + "  " + val[i][j][1]);
         dt.set(i, val[i][j][0], val[i][j][1]);
      }
   }
   chart (tmp, de.grogra.pf.ui.ChartPanel.XY_PLOT);
}
```

```
private static void zeichneSegmentChartBaum(DatasetRef tmp,int bnr,boolean kronenansatz)
// Zeichnen eines Charts aus der Variable tempDouble für alle Bäume
{
   int maxseg = 0;
   int startsegment = 1;
   [
// Regel zum bestimmen der maximalen Segmentzahl
      b:Baum, (b.bnr == bnr) ::> {
         maxseg = ((Segment) b.getLastChild() ).snr;
         if(kronenansatz == true) {startsegment = b.kronenAnsatzSegment;}
      }
   ]
// Erzeugen eines 2-d Array nach Baum und Segment
   float[][] val = new float[maxseg][2];
// initalisieren des Arrays
   for (int j=0 ; j< maxseg; j++){
      val[j][0]=0.0f;
      val[j][1]=0.0f;
   }
   [
// Regel zum Auslesen der Werte
      s:Segment, (s.bnr == bnr ) ::> {
            val[s.snr-1][0]=s.tmpDoubleX;
            val[s.snr-1][1]=s.tmpDoubleY;
      }
   ]
// initialisieren des Charts
   tmp.clear ();
   Dataseries dt;

//setzen der Datenreihen
   tmp.setColumnKey(0, "B" + bnr);
// setzen der Datenpunkte
   for (int j = startsegment-1; j < maxseg; j++){
      dt = tmp.addRow();
      dt.set(0, val[j][0], val[j][1]);
   }
   chart (tmp, de.grogra.pf.ui.ChartPanel.XY_PLOT);
}
//------------------------ Kontrollkonus ---------------------------------
public static void zeichneKontrollkonus(int lbnr, int lsnr, float winkel)
// zeichnet einen nach oben offenen Konus für ein gegebenes Segment ein.
// Höhe des Konus = Baum.baunHoehe
// Öffnung des Konus = winkel
{
   float ba = 0.0f;
   float sa = 0.0f;
   [
   // zeichnen eines cones für das übergebene Segment lbnr:lsnr
      s1:Segment, (s1.bnr == lbnr &&  s1.snr == lsnr ) ==> s1 [ c:Cone ]
      {
         ba = ( (Baum) s1.getParent()).baumHoehe;
         c[length] = (s1.base - ba );
         //c.setAxis(0.0f, 0.0f, (sa -ba));
         c.setTransform(0,0, s1.base);
         c.setRadius(tan((winkel)*DEG)* (ba-sa));
         c.setOpen(true);
      };
   ]

}
//*****************************************************************************/
// Konsolenausgaben
//*****************************************************************************/
public static void ausgabeBetaBaumSummen()
{
   [
      {
      float ddg;
      }
      b:Baum ::> {
         ddg = 0.0f;
         for((* s:Segment, (s.bnr == b.bnr) *)) {  ddg += s.nadelMasseGesamt; }
         println("Baum " + b.bnr + " summe nadelMasseGesamt: " + ddg);
      }
   ]
}
```

10.3 deliverscene.gsz

10.3.1 delivery.rgg

```
import static java.lang.Math.*;
import java.io.*;
import de.grogra.imp.net.*;
import de.grogra.util.*;
import de.grogra.pf.registry.*;
import java.util.regex.*;
import java.util.*;

import forester.http.formdata.*;
import forester.description.*;
import forester.groimp.*;

String modelOutputDir = ForesterUtils.leseHttpServerBasisDirectoryAusPreferences() +
   "\\" + "modeloutput\\";
String server  = "http://ufgb982:58080";
String modelout = "deliverscene.gsz";


//-------------------------------------------------------------------------------
//                       Startup durch eine ForesterSzene per HTTP Servers
//-------------------------------------------------------------------------------
protected void startup ()
{
   super.startup ();
   HttpResponse resp = HttpResponse.get (workbench());
   System.out.println (resp);
   if (resp != null)
   {
      runLater (resp);
   }
}
// Methode run wird von runLater aus startup aufgerufen.
// Findet nur Verwendung wenn die Simulation über den HTTP-Server gestartet wird.
protected void run (Object info)
{
   boolean inputstatus = false;
   String tmp;
   String filename ;
   HttpResponse resp = (HttpResponse) info;
   StringBuffer buf = new StringBuffer ();
   passBoundary ();
   String contentType   = (resp.getRequest()).getHeaderField("content-type");
   String boundary = contentType.substring(contentType.indexOf("boundary=")+9);
   // Auslesen des Content-Bereichs
   String content = new String((resp.getRequest()).getContent(), "UTF-8");
   //Auslesen der Formfelder aus dem content Bereich des geposteten HTTP-Stream
   HttpMultipartFormData n1 = new HttpMultipartFormData(content, "--" + boundary);

   filename = n1.getFieldContent("datei");
   String path = ( modelOutputDir + filename );
   println(path);
   //{{{ Check if file is valid
   tmp = ForesterUtils.readFileInString(path);

   if(tmp != null){
      buf.append(tmp + "\n");
      buf.append("status=ready\n");
   }
   else {
      buf.append("status=wait\n");
      println("wait");
   }
   resp.setContent ("text/plain", "UTF-8", buf.toString());
   resp.send (true);


   closeWorkbench ();
}

protected void init(){}
// Ende delivery.rgg
```

10.4 Package forester.http.formdata

10.4.1.1 Klasse HttpMultipartFormData

```
package forester.http.formdata;

/*
 * Copyright (C) 2002 - 2006 Forstliche Biometrie und Informatik Universität Göttingen
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 */

/**
 * Die Klasse zerlegt den Content eines mit HTTP Post gesendeten
 * Multipart/Formdata Formulares in seine Bestandteile und enthält ein Array mit
   Objekten
 * der Klasse <code>HttpFormField</code> für die einzelnen gesendeten Formularfelder
 *</p>
 *<pre>
 * nicht benannte Formularfelder werden nicht erfasst
 *
 * Einschränkungen:
 * Die Klasse {@link forester.http.formdata.HttpFormField HttpFormField }erfasst
   gegenwärtig noch keine Felder zum Upload
 * von Dateien mit mehreren Dateien.
 *
 * Spezifikationen HTML 4.01
 * http://www.w3.org/TR/1999/REC-html401-19991224
 * </pre>
 *
 * @author Dirk Lanwert Stand 12/2006
 */


public class HttpMultipartFormData {
   /** HTML Boundary String des Formulars */
   private String boundary;
   /** Inhalt der gesamten Formulars*/
   private String content;
   /** Anzahl der ermittelten Formuularfelder */
      private int numberOfFields;
   /** Array mit HttpFormFiled Objekten */
   public HttpFormField[] fields;

   /**
    * Constructor der Klasse
    * @param content
    * @param boundary
    */
   public HttpMultipartFormData(String content , String boundary){

      this.boundary = boundary;
      this.content = content;
      this.numberOfFields = Math.max( countBoundaries()-1,0);
      this.fields = new HttpFormField[this.numberOfFields];
      extractFields();
   }
```

```
/* Zählen der Vorkommen von Boundaries */
private int countBoundaries(){
   int pos = 0; int count = 0;
   do {
      pos  = this.content.indexOf(this.boundary, pos) ;
      pos += this.boundary.length();
      count ++;
   } while ( pos >= this.boundary.length());
   count--;
   return  count;
   }
   /* Extrahieren der einzelnen Parts aus dem HTTP Content anhand der Boundaries */
   private void extractFields(){
      int pos1 = 0; int pos2 = 0;
      for ( int i = 0; i < this.numberOfFields; i++) {
         pos1  = this.content.indexOf(this.boundary, pos2) ;
      pos1 += this.boundary.length();
         pos2  = this.content.indexOf(this.boundary, pos1) ;

/* Erzeugen der einzelnen HttpFormFiled-Objekte für die einzelnen Content-Parts.
   Achtung: vor der Boundary steht immer ein newline. Dieser wird mit trim() eleminiert.
   */
         this.fields[i] = new HttpFormField(this.content.substring(pos1, pos2-2));
      }
   }

/**Durchsucht das Array mit HttpFormField-Objekten nach einem Feld mit pasendem Namen.
 * @param key gibt den zu suchenden Namen des Formfeldes an.
 * @return gibt den ermittelten Inhalt der Formularfeldes zurück.
 * null wird zurückgegeben, wenn ein passendes Formularfeld nicht gefunden wird.
 */
   public String getFieldContent(String key){
      String ret ;
      for ( int i = 0; i < this.numberOfFields; i++){
         if((this.fields[i].getName()).equalsIgnoreCase(key)){
          ret = ((this.fields[i].getFieldContent()));
          return ret;
         }
      }
      return null;
   }
   public String getFieldContentType(String key){
      String ret ;
      for ( int i = 0; i < this.numberOfFields; i++){
         if((this.fields[i].getName()).equalsIgnoreCase(key)){
          ret = ((this.fields[i].getContentType()));
          return ret;
         }
      }
      return null;
   }
   public String getFieldContentDisposition(String key){
      String ret ;
      for ( int i = 0; i < this.numberOfFields; i++){
         if((this.fields[i].getName()).equalsIgnoreCase(key)){
          ret = ((this.fields[i].getContentDisposition()));
          return ret;
         }
      }
      return null;
   }
   public String getFieldType(String key){
      String ret ;
      for ( int i = 0; i < this.numberOfFields; i++){
         if((this.fields[i].getName()).equalsIgnoreCase(key)){
          ret = ((this.fields[i].getType()));
          return ret;
         }
      }
      return null;
   }
   public int getNumberOfFields() {
      return this.numberOfFields;
   }
}
// Ende Klasse HttpMultipartFormData
```

10.4.1.2 Klasse HttpFormField

```
package forester.http.formdata;
/*
 * Copyright (C) 2002 - 2006 Forstliche Biometrie und Informatik Universität Göttingen
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 */

/**
 * <p>Die Klasse zerlegt den als String übergebenen Content-Part eines
 * HTML-Formlfields in die entsprechenden Parameter.</p>
 * <p>
 * <pre>
 * Folgende werden für die Typen von Feldern gesetzt:
 *
 * für Formularfelder wie textinput,textarea, select, radio, checkbox:
 * contentDisposition
 * contentType = "text/plain" (default)
 * name
 * type = "text"
 * fieldContent
 *
 * für Datei-Upload: <br>
 * contentDisposition
 * contentType
 * name
 * filename
 * type = "file"
 * fieldContent
 *
 *  für unbekannte Elemente wir type = "unknown" gesetzt
 *
 * Einschränkungen:
 * Die Klasse HttpFormField erfasst gegenwärtig noch keine Felder zum Upload
 * von Dateien mit mehreren Dateien.
 *
 *
 * Spezifikationen HTML 4.01
 * http://www.w3.org/TR/1999/REC-html401-19991224
 * </pre>
 * </p>
 *   @author Dirk Lanwert Stand 12/2006
 */

public class HttpFormField {
   /** contentDisposition Element des Formularfeldes */
   private String contentDisposition;
   /** Name des Formilarfeldes */
   private String name;
   /** Name des gesendeten Files */
   private String fileName;
   /** Typ des Formularfeldes ("file" oder "text") */
   private String type;
   /** HTML contentType des Formularfeldes Default = "text/plain" */
   private String contentType;
   /** Inhalt des Formularfeldes (nach einem Trim())*/
   private String fieldContent;
```

```
/**
 * Konstructor der Klasse
 * @param inbuf
 */
   public HttpFormField(String inbuf){

      int pos1 = inbuf.indexOf("Content-Disposition:");
      int pos2 = inbuf.indexOf("\r\n", pos1);

      if(pos1 >= 0 && pos2 >= pos1){
         this.contentDisposition=inbuf.substring(pos1+21, pos2);
      }
      else {
         pos2 = 0; pos1 = 0;
      }

      pos1 = inbuf.indexOf("Content-Type:", pos2);
      if (pos1 >= 0){
         pos2 = inbuf.indexOf("\r\n", pos1);
         this.contentType=inbuf.substring(pos1+14, pos2);
      }
      else {
         this.contentType = "text/plain";
      }

      pos1 = inbuf.indexOf("\r\n\r\n", pos2);
      this.fieldContent=inbuf.substring(pos1+4);
      analyseContentDisposition();
   }

   /* Zerlegen des ContentDisposition Strings */
   /* Nur für form-data Elemente */
   /* legt auch den Typ fest */

      private void analyseContentDisposition(){
         int pos1 = 0; int pos2 = 0;
         this.type = "unknown";
         pos1 = this.contentDisposition.indexOf("form-data;");
         if(pos1 != -1) {
            /*Ermitteln des Namens*/
            pos1 = this.contentDisposition.indexOf("name=\"", pos1);
            if(pos1 != -1) {
               pos2 = this.contentDisposition.indexOf("\"", pos1+6);
               if(pos2 != -1) {this.name=this.contentDisposition.substring(pos1+6,
   pos2); pos1 = pos2;  }
            }
            /* Ermitteln des Filenamens wenn vorhanden*/
            pos1 = this.contentDisposition.indexOf("filename=\"", pos1);
            if(pos1 != -1) {
               pos2 = this.contentDisposition.indexOf("\"", pos1+10);
               if(pos2 != -1) {
                  this.fileName=this.contentDisposition.substring(pos1+10, pos2);
                  pos1 = pos2;
                  this.type = "file";
                  }
            }
            else
               this.type = "text";
         }
      }

      public String getContentDisposition(){ return this.contentDisposition;}
      public String getName(){ return this.name;}
      public String getFileName(){ return this.fileName;}
      public String getType(){ return this.type;}
      public String getContentType(){ return this.contentType;}
      public String getFieldContent(){ return this.fieldContent;}
}
// Ende Klasse HttpFormField
```

10.5 Package forester.description

10.5.1.1 Klasse ForesterDescription

```
package forester.description;

/* Copyright (C) 2002 - 2006 Forstliche Biometrie und Informatik Universität Göttingen
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License
* as published by the Free Software Foundation; either version 2
* of the License, or any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*/


import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import forester.groimp.ForesterUtils;

/**
 * Eine Foresterdescription ist eine in VRML geschriebene Wald Bestandesszene mit
   spezifischen Prototypdefinitionen für
 * das Virtual Forester Projekt des Institutes Für Forstliche Biometrie und Informatik
   an der Universität Göttingen.
 *
 * Die vorliegende Klasse ist die Basis zur Behandlung von Foresterdescription.
 *
 * Einschränkungen:
 * Der vorliegende Konstruktor benötigt zur Zeit eine vollstädenige Foresterdescription
   anhand derer ein entsprechendes Objekt
 * dieser Klasse instanziiert wird. Dieses Objekt kann anschließend manipuliert werden.
 * Die Neukonstruktion eines Foresterdescription Objektes ist noch nicht implementiert.
 *
 * @author Dirk Lanwert Stand 12/2006
 *
 */

public class ForesterDescription {
   /** Enthält den Definitionskopf der VRML Forester Description*/
private String foresterDefinitionString;
   /** Anzahl der Bäume im Description Objekt*/
private int       numberOfTrees;
   /** Array der Baumobjekte im Description Objekt*/
public ArrayList foresterTrees;
   /** Basisname der Description */
private String fileName;
private int forecastPeriode = 0;

   /**
    * Konstruktor für Forester Description-Objekte anhand einer vorhandenen Description
    * @param inbuf  - VRML Datei Inhalt gemäß Foresterdescription Prototyp
    * @param filename - Basistname der VRML Datei
    */
   public ForesterDescription(String inbuf, String filename) {


      int pos1 = 0; int pos2 = 0;
      String buf;
      this.fileName = ForesterUtils.extractFileName(filename);
      this.foresterTrees = new ArrayList(50);

   // System.out.println("description angekommen");
      pos2= inbuf.length();
      pos1 = inbuf.indexOf("TREE {");
         if(pos1 >= 0) { pos2 = pos1;}
```

```
            this.foresterDefinitionString = inbuf.substring(0, pos2);

            while( pos1 >= 0) {
               pos1 = inbuf.indexOf("{", pos1);
               pos2 = inbuf.indexOf("}", pos1);
               buf = inbuf.substring(pos1, pos2+1);
               this.foresterTrees.add(new ForesterTree(buf));
               this.numberOfTrees ++;
               pos1 = inbuf.indexOf("TREE {", pos2);
            }
      }

      /**
       * Methode für das Zusammenstellen der VRML Description im Forester Prototype Format
       * @return String mit Foresterdescription
       */
      public String getForesterDescription(){

               StringBuffer buffer = new StringBuffer();
               buffer.append(this.foresterDefinitionString + "\n");
               for(int i = 0; i < this.numberOfTrees; i++){
               buffer.append( ((ForesterTree)(foresterTrees.get(i))).getTreeString() +
                  "\n");
               }
               buffer.append("]\n}\n");
               return buffer.toString();
            }



      /**
       * Methode zum Speichern einer VRML Description im Forester Prototype Format
       * @param filename
       * @throws IOException
       */
      public void speichereForesterDescription( String filename)throws IOException {
            FileWriter w = new FileWriter (filename);
            w.write (this.getForesterDescription());
            w.flush();
      //    System.out.println("Filename write:" + filename);

         }

      public int getNumberOfTrees(){ return this.numberOfTrees;}
      public String getFileName(){ return this.fileName;}
      public int getForecastPeriode(){ return this.forecastPeriode;}
}
// Ende Klasse ForesterDescription
```

10.5.1.2 Klasse ForesterTrees

```
package forester.description;

/* Copyright (C) 2002 - 2006 Forstliche Biometrie und Informatik Universität Göttingen
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License
* as published by the Free Software Foundation; either version 2
* of the License, or any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*/

import java.util.Scanner;
import java.util.regex.MatchResult;

/**
 * Die ForesterTree Klasse behandelt Baum Elemente einer {@link
   forester.description.ForesterDescription Forester Discription}.
 * <p>
 * <pre>
 * Die Baumobjekte enthalten Egenschaften gemäß des Forester Prototyps:
 * Die Syntax und Semantik der Prototyp Definition ist zu beachten
 *
 * <b>Besonders zu beachten ist:</b>
 * Die Arrays labels[] & data[] sind von gleicher Länge. numberOfData = numberOfLabels
 * Ihre Inhalte sind paarweise aneinander gebunden
 * labels[1] = Identifier einer Information z.B. Baumalter
 * data[1] = enthält die entsprechende Information z.B. 108 (Jahre)
 *
 * <b>Wichtig</b>
 * Nicht alle Elemente müssen für einen Forester Tree definiert sein.
 * Pflichtelmente sind für Objekte dieser Klasse mit Defaultwerten definiert
 * Weitergehende Elemente sind nicht vordefiniert.
 * Die Schlüssigkeit der resultierenden Description ergibt sich für die Elemente
 * ( species, tapperSystem, stemtexture, und croneTexture) nur dann, wenn genutzte
 * Werte auch im Definitionskopf (Prototypendefinition) vorhanden sind.
 *
 * Es wird an dieser Stelle keine Überprüfung der gesamten VRML Szene vorgenommen.
 *
 * </pre>
 *
 * @author Dirk Lanwert Stand 12/2006
 *
 */
public class ForesterTree {
   /** <pre>Identifier des Baumes
    *  Defaultwert 0 </pre>*/
   public int id = 0;
   /** <pre> Status des Baumes: 5 Zustände sind vordefiniert:
    * 1=normal, 2=posetive selecion, 3= negative selecion, 4= cutted, 5=eleminated
    *  Defaultwert 1 </pre>*/
   public int status = 1;
   /** <pre> Position des Baumes (x Richtung, Höhenkoordinate, y Richtung)
    *  Defaultwert {0,0,0} </pre>*/
   public float[] position = {0,0,0};
   /** <pre> Radius des Stammes (x, Höhen, y)
    *  Defaultwert {0,1,0} </pre>*/
   public float[] stemSize = {0,1,0};
   /** <pre>Position des Kronenansatzes (x Richtung, Höhenkoordinate, y Richtung)<p>
    * Bezieht sich relativ auf den Stammfuss (position)
    *  Defaultwert {0,1,0} </pre>*/
   public float[] crownPosition = {0,1,0};
   /** Radius der Krone (x, Höhen, y) Defaultwert {1,1,1} </pre>*/
   public float[] crownSize = {1,1,1};
   /** <pre>Verformung der Krone (Reduktionsfaktor 0-1 für 6 Achsen (Laufrichtung von 9
   Uhr mit dem Uhrzeiger )Defaultwert {1,1,1,1,1,1} </pre>*/
   public float[] crownValues = {1,1,1,1,1,1};
   /** Baumart  */
```

```
   public String species;
   /** Array für frei definierbare Baumdaten (Bezeichner der Daten)*/
   private String labels[];
   /** Anzahl der Felder im labels Array*/
   public int numberOfLabels = 0;
   /** Array für frei definierbare Baumdaten (daten)*/
   private String data[];
   /** Anzahl der Felder im data Array*/
   public int numberOfData = 0;
   /** Texturangabe für den Stamm */
   public String stemTexture;
   /** texturangabe für die Krone */
   public String crownTexture;
   /** Schaftsystem für die Darstellung des Stamms */
   public int taperSystem;

   ForesterTree() {
// später zu ergänzen wenn neue Bäume hinzugefügt werden sollen
   }

   /**
    * Konstruktor eine Baumelementes mit Übergabe der Baumwerte
    * in Form eines vollständigen tree elementes aus einer Forester Description
    * im VRML Forester Prototyp Format
    * @param inbuf
    */
   public ForesterTree(String inbuf) {
      readLabels(inbuf);
      readData(inbuf);
      readPosition(inbuf);
      readStemSize(inbuf);
      readCrownPosition(inbuf);
      readCrownSize(inbuf);
      readCrownValues(inbuf);
      readSpecies(inbuf);
      readId(inbuf);
      readStatus(inbuf);
      readCrownTexture(inbuf);
      readStemTexture(inbuf);
      readTaperSystem(inbuf);
   }

   /**
    * Methode zum Auslesen des Baum Labels Bereichs der Forester Description Datei
    * @param inbuf
    */
   private void readLabels(String inbuf){
      int pos1 = 0; int pos2 = 0; String buf;
      pos1 = inbuf.indexOf("labels [");
         if(pos1 >= 0) {
            pos1 = inbuf.indexOf("[", pos1);
            pos2 = inbuf.indexOf("]", pos1);
            buf=inbuf.substring(pos1 +1, pos2);
            buf = buf.trim().substring(1);
            String[] result = (buf.trim()).split("\\s+\"");
            this.numberOfLabels = result.length;
            this.labels = new String[result.length];
            for (int i=0; i < result.length; i++) {
               this.labels[i] = result[i].substring(0,(result[i].indexOf("\"")));
            }
         }
   }

   /**
    * Ausgabe der Labelelmente auf der Standard Konsole
    *
    */
   public void printLabels(){
      System.out.println("Ausgabe " + this.numberOfLabels +" Baum " + this.id);
      for(int i = 0; i < this.numberOfLabels; i++)
      System.out.println("---" + this.labels[i] + "---");
   }
   /**
    * Methode zum Auslesen des Baum Data Bereichs der Forester Description Datei
    *
    * @param inbuf
    */
```

```
private void readData(String inbuf){
      int pos1 = 0; int pos2 = 0; String buf;
      pos1 = inbuf.indexOf("data [");
         if(pos1 >= 0) {
            pos1 = inbuf.indexOf("[", pos1);
            pos2 = inbuf.indexOf("]", pos1);
            buf=inbuf.substring(pos1 +1, pos2);
            buf = buf.trim().substring(1);
            String[] result = (buf.trim()).split("\\s+\"");

            this.numberOfData = result.length;
            this.data = new String[result.length];
               for (int i=0; i < result.length; i++) {
                  this.data[i] = result[i].substring(0,(result[i].indexOf("\"")));
               }
         }
   }

   /**
    * Ausgabe der Dataelmente auf der Standard Konsole
    *
    */
public void printData(){
      System.out.println("Ausgabe " + this.numberOfData +" Baum " + this.id);
      for(int i = 0; i < this.numberOfData; i++)
      System.out.println("---" + this.data[i] + "---");
   }

   /**
    * Methode zum Auslesen und Setzen des Statuswertes
    * @param inbuf
    */
private void readStatus(String inbuf){
      Scanner s = new Scanner(inbuf);
      if (s.findWithinHorizon("\\s+status\\s+(\\d+[.]*\\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.status=  Integer.valueOf(result.group(i));
         }
      }
      s.close();
   }

   /**
    * Methode zum Auslesen und Setzen des ID Wertes
    * @param inbuf
    */
private void readId(String inbuf){
      Scanner s = new Scanner(inbuf);
      if (s.findWithinHorizon("\\s+ID\\s+(\\d+[.]*\\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.id =  Integer.valueOf(result.group(i));
         }
      }
      s.close();
   }

   /**
    * Methode zum Auslesen und Setzen des ID Wertes
    * @param inbuf
    */
   private void readTaperSystem(String inbuf){
      Scanner s = new Scanner(inbuf);
      if (s.findWithinHorizon("\\s+shaftSystem\\s+(\\d+[.]*\\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.taperSystem =  Integer.valueOf(result.group(i));
         }
      }
      s.close();
   }
```

```
   /**
    * Methode zum Auslesen und Setzen der Positions Werte
    * @param inbuf
    */
private void readPosition(String inbuf){
   Scanner s = new Scanner(inbuf);
         if
   (s.findWithinHorizon("\\s+position\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\
   d*)\\s+", 0) != null)
         {
            MatchResult result = s.match();
            for (int i=1; i<=result.groupCount(); i++) {
                  this.position[i-1]=  Float.valueOf(result.group(i));
            }
         }
         s.close();
      }

   /**
    * Methode zum Auslesen und Setzen der StemSize Werte
    * @param inbuf
    */
private void readStemSize(String inbuf){
      Scanner s = new Scanner(inbuf);
      if
         (s.findWithinHorizon("\\s+stem_size\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d
         +[.]*\\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.stemSize[i-1]=  Float.valueOf(result.group(i));
         }
      }
      s.close();
   }

   /**
    * Methode zum Auslesen und Setzen der Kornen Positions Werte
    * @param inbuf
    */
private void readCrownPosition(String inbuf){
      Scanner s = new Scanner(inbuf);
      if
   (s.findWithinHorizon("\\s+crown_position\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d+
   [.]*\\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.crownPosition[i-1]=  Float.valueOf(result.group(i));
         }
      }
      s.close();
   }
   /**
    * Methode zum Auslesen und Setzen der Kronen Size Werte
    */
private void readCrownSize(String inbuf){
      Scanner s = new Scanner(inbuf);
      if
   (s.findWithinHorizon("\\s+crown_size\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*
   \\d*)\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.crownSize[i-1]=  Float.valueOf(result.group(i));
         }
      }
      s.close();
   }



/**
 * Methode zum Auslesen und Setzen der Kronenverformungswerte Werte
 * @param inbuf
 */
```

```
   private void readCrownValues(String inbuf){
      Scanner s = new Scanner(inbuf);
      if
   (s.findWithinHorizon("\\s+crown_values\\s+\\[\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+
   (\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+(\\d+[.]*\\d*)\\s+\\]\\s+", 0)
   != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.crownValues[i-1]=  Float.valueOf(result.group(i));
         }
      }
      s.close();
   }

   /**
    * Methode zum Auslesen und Setzen des Baumarten Wertes
    * @param inbuf
    */
   private void readSpecies(String inbuf){
      Scanner s = new Scanner(inbuf);
      if (s.findWithinHorizon("\\s+Species\\s+\"(\\w+)\"\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.species =  result.group(i);
         }
      }
      s.close();
   }

   /**
    * Methode zum Auslesen und Setzen des stem_texture Wertes
    * @param inbuf
    */

   private void readStemTexture(String inbuf){
      Scanner s = new Scanner(inbuf);
      if (s.findWithinHorizon("\\s+stem_texture\\s+\"(\\w+[.]*\\w*)\"\\s+", 0) != null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.stemTexture = result.group(i);
         }
      }
      s.close();
   }

/**
    * Methode zum Auslesen und Setzen des crown_texture Wertes
    * @param inbuf
 */

   private void readCrownTexture(String inbuf){
      Scanner s = new Scanner(inbuf);
         if (s.findWithinHorizon("\\s+crown_texture\\s+\"(\\w+[.]*\\w*)\"\\s+", 0) !=
   null)
      {
         MatchResult result = s.match();
         for (int i=1; i<=result.groupCount(); i++) {
               this.crownTexture =  result.group(i);
         }
      }
      s.close();
   }



   /**
    * Methode zum Zusammenstellen eines Tree strings für eine VRML Forester Description
    * @return VRML Tree String im Forester Prototyp Format
    */
   public String getTreeString(){
      StringBuffer buffer = new StringBuffer();
      buffer.append("TREE {");
      buffer.append(" ID " + this.id);
      buffer.append(" status " + this.status);
```

```
      buffer.append(" labels [");
      for(int i = 0; i < this.numberOfLabels; i++) {
         buffer.append(" \"" + this.labels[i] + "\"" ); }
      buffer.append("  ]");
      buffer.append(" data [");
      for(int i = 0; i < this.numberOfData; i++) {
         buffer.append(" \"" + this.data[i] + "\"" ); }
      buffer.append("  ]");
      buffer.append(" position " + this.position[0] + " " + this.position[1] + " " +
         this.position[2]);
      buffer.append(" crown_position " + this.crownPosition[0] + " " +
         this.crownPosition[1] + " " + this.crownPosition[2]);
      buffer.append(" stem_size " + this.stemSize[0] + " " + this.stemSize[1] + " " +
         this.stemSize[2]);
      buffer.append(" crown_size " + this.crownSize[0] + " " + this.crownSize[1] + " " +
         this.crownSize[2]);
      buffer.append(" crown_values [ " + this.crownValues[0]  + " " +
         this.crownValues[1]  + " " + this.crownValues[2]  + " " + this.crownValues[3]
         + " " + this.crownValues[4]  + " " + this.crownValues[5] + " ] ");
      buffer.append(" Species \"" + this.species + "\"");
      if(stemTexture != null) {
         buffer.append(" stem_texture \"" + this.stemTexture + "\""); }
      if(crownTexture != null) {
         buffer.append(" crown_texture \"" + this.crownTexture + "\""); }
      buffer.append(" }" );
      return buffer.toString();
   }

   /**
    * Durchsucht das labels[] Array nach einer enstprechenden Übereinstimmung mit key
    * und ermittelt den zugehörigen data[] Wert. labels[] und data[] sind parallel
   angelegt.
    * @param label
    * @return data value des data[] elementes welches zum übergebenen Label passt.
    * null wird zurückgegeben, wenn kein passendes Label gefunden wird.
    */
   public String getDataValue(String key){
      String ret;
      for ( int i = 0; i < this.numberOfLabels; i++){
         if(this.labels[i].equalsIgnoreCase(key)){
            ret = ((this.data[i]));
            return ret;
         }
      }
      return null;
   }

   /**
    * Durchsucht das labels[] Array nach einer enstprechenden Übereinstimmung mit key
    * und setzt den zugehörigen data[] Wert. labels[] und data[] sind parallel angelegt.
    * @param label
    * @param value
    * @return -1 bei Misserfolg, index des Arrays bei Erfolg
    */
   public int setDataValue(String key, String value){
      int ret = -1;
      for ( int i = 0; i < this.numberOfLabels; i++){
         if(this.labels[i].equalsIgnoreCase(key)){
          ((this.data[i])) = value; ret = i;

         }
      }
      return ret;
   }
}
// Ende Klasse ForesterTrees
```

10.6 Package forester.groimp

10.6.1.1 Klasse ForesterUtils

```
package forester.groimp;
/* Copyright (C) 2002 - 2006 Forstliche Biometrie und Informatik Universität Göttingen
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 */

import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import de.grogra.pf.registry.Item;
import de.grogra.pf.registry.Registry;
import de.grogra.pf.ui.Window;
import de.grogra.pf.ui.Workbench;
import static de.grogra.rgg.Library.*;
import de.grogra.util.Utils;
;
/**
 * Diese Klasse enthält statische Methoden zur einfacheren Anbindung an Groimp.<br>
 * <pre>
 * genutzt werden:
 * {@link de.grogra.util.Utils  de.grogra.util.Utils} und
 * {@link de.grogra.pf.registry de.grogra.pf.registry.*}
 * </pre>
 * <p>
 * @author Dirk Lanwert Stand 12/2006
 *
 */

public class ForesterUtils {

/**
 * Methode zum Auslesen des Groimp Http Server Basis Verzeichnisses aus den Preferenzes
 * @return Pfad des Basisverzeichnisses
 */
public static String leseHttpServerBasisDirectoryAusPreferences(){

   String base = null;
   Item item = Item.resolveItem(Registry.current(), "/http/commands/open");
   base =   (String) Utils.getObject(item, "directory");
   if (base == null)
   {
      base = System.getProperty ("user.home");
   }
   return base;
}

public static String leseUserBasisDirectoryAusRegistry(){

   String base = null;
   base = System.getProperty ("user.home");
   return base;
}

/**
 * Methode zum Extrahieren des Dateinamens aus einem Pfad
 * nutzt split("\\\\") , noch nicht getestet
 * @param in
 * @return  den letzten Abschnittes des Pfades
 */
```

```
public static String extractFileName2(String in){
   String out;
   System.out.println("Filename in:" + in);
   String[] result = (in.trim()).split("\\\\");
   if (result != null){
      out = result[(result.length -1)];
      System.out.println("Filename out:" + out);
      return out;
      }
   else {
      return in;
      }
   }

/**
 * Methode zum Extrahieren des Filenamens aus einem Pfad
 * nutzt lastIndexOf(), getestet
 * @param in
 * @return  den letzten Abschnittes des Pfades
 */
public static String extractFileName(String in){
   String out;
   int pos;
// System.out.println("Filename in:" + in);
    pos = in.lastIndexOf("\\");

   if (pos != -1){
      out = in.trim().substring(pos+1);
      return out;
      }
   else {
      pos = in.lastIndexOf("/");
      if (pos != -1){
         out = in.trim().substring(pos+1);
         return out;
      }
      else {
         return in;
         }
      }
   }
/**
 * Liest eine Datei von der lokalen Platte in einen String
 * @param path
 * @return Inhalt der Datei oder null
 * @throws IOException
 */
public static String readFileInString(String path) throws IOException{

   StringBuffer buf = new StringBuffer ();;
   String ret;
   int i;
   File file = new File(path);
// Check if file is valid
   if(file.exists()){
      System.out.println("FileExists");
      FileReader r = new FileReader(path);
      do {
         i = r.read();
         if (i >= 0) buf.append((char) i);
      } while(i != -1);
   ret = buf.toString();
   return ret;
   }
   else
   {
   return null;
   }
}
public static boolean getParameterFromDialog(String title, String message, String
   initial, boolean defaultwert) {
   try {
      Workbench wrk = workbench();
      Window wd = wrk.getWindow();
      String tmp = wd.showInputDialog(title, message, initial);
      boolean wert = Boolean.valueOf(tmp);
      return wert;
```

```
   }
   catch(Exception e){
      return defaultwert;
   }
}
public static int getParameterFromDialog(String title, String message, String initial,
   int defaultwert)  {
   try {
      Workbench wrk = workbench();
      Window wd = wrk.getWindow();
      String tmp = wd.showInputDialog(title, message, initial);
      int wert = Integer.valueOf(tmp);
      return wert;
   }
   catch(Exception e){
      return defaultwert;
   }
}
public static String getParameterFromDialog(String title, String message, String
   initial, String defaultwert)  {
   try {
      Workbench wrk = workbench();
      Window wd = wrk.getWindow();
      String tmp = wd.showInputDialog(title, message, initial);
      return tmp;
   }
   catch(Exception e){
      return defaultwert;
   }
}
public static float getParameterFromDialog(String title, String message, String initial,
   float defaultwert)   {
   try {
      Workbench wrk = workbench();
      Window wd = wrk.getWindow();
      String tmp = wd.showInputDialog(title, message, initial);
      float wert = Float.valueOf(tmp);
      return wert;
   }
   catch(Exception e){
      return defaultwert;
   }
}
}
// Ende Klasse ForesterUtils
```

10.7 Vergrößerte Grafiken

In diesem Teil des Anhangs finden sie einige der Grafiken aus Kapitel 3.9 in vergrößerter Form.

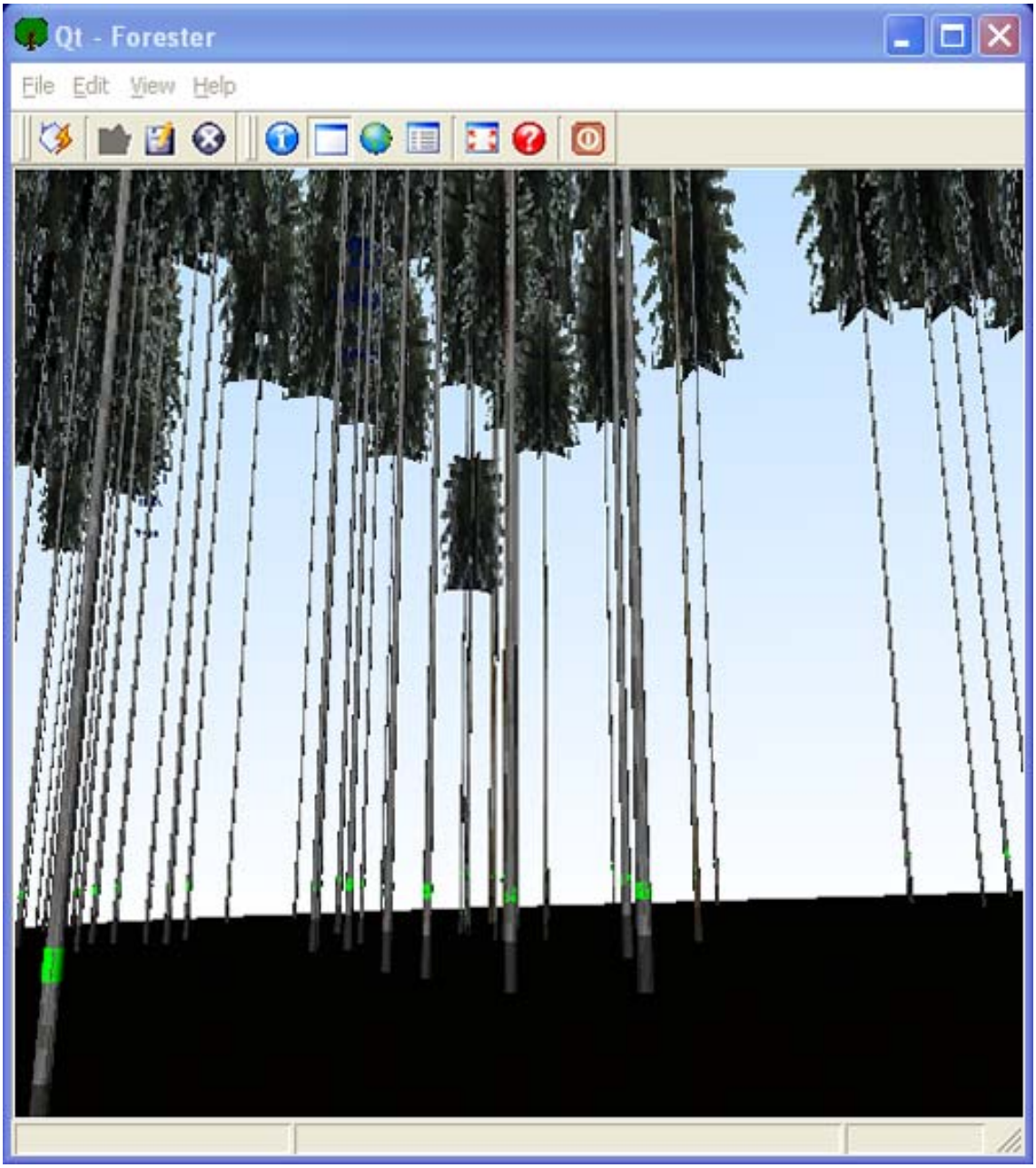


Abbildung 37: Darstellung einer abgebrochenen Krone im Virtual Forester Client als als Qualitätsmerkmal des Baumes. Die Texturen wurden durch den Quality Identifier hinzugefügt.

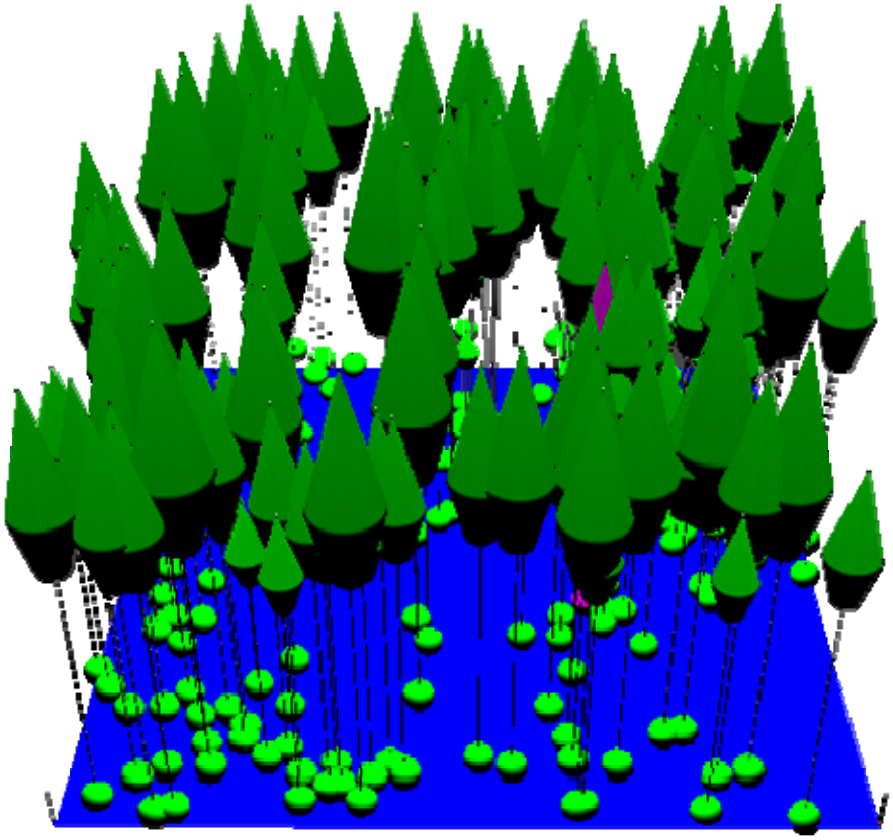


Abbildung 38: Räumliche Ausgangssituation der Variante A (169 Bäume auf 50x50 Meter). Baum 65 ist lila eingefärbt.

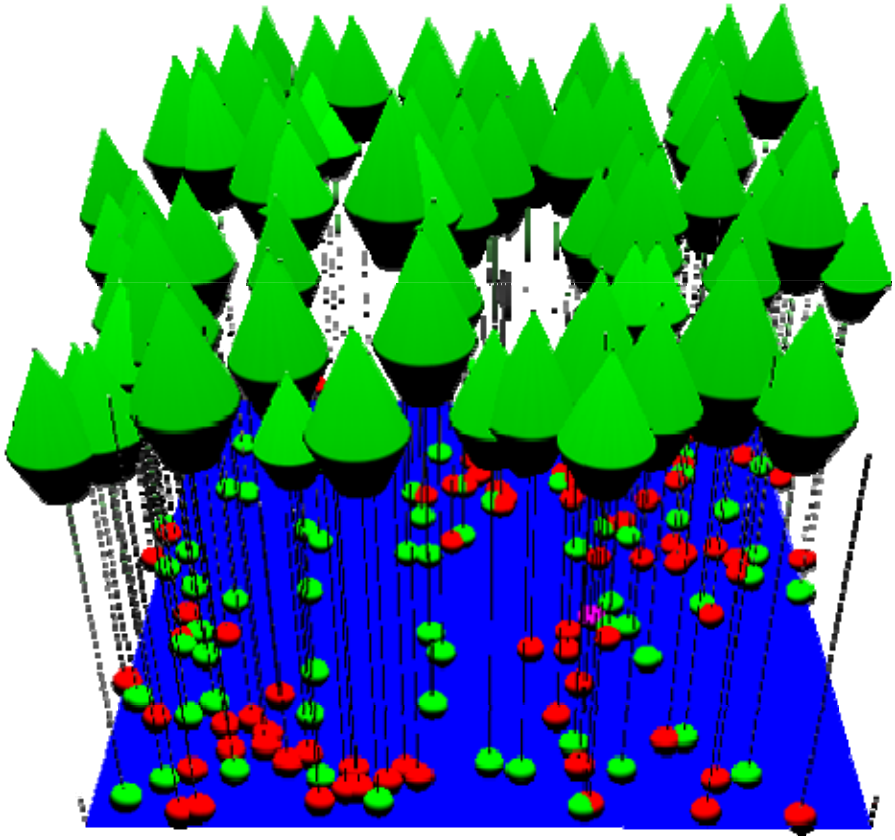


Abbildung 39: Räumliche Situation der nach der 20jährigen Simulation der Variante A. Rote Basispunkte markieren abgestorbene Bäume. Baum 65 ist abgestorben.

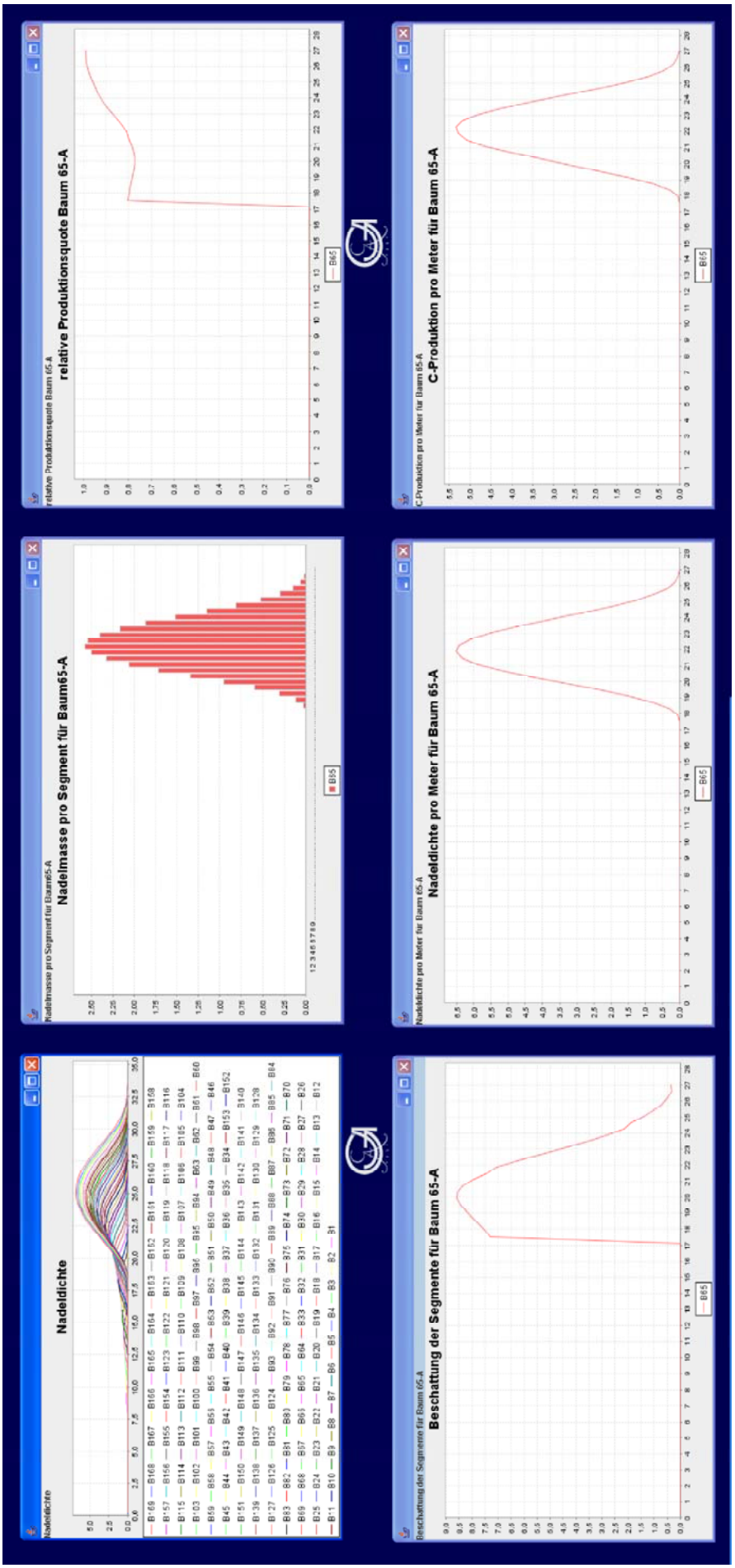


Abbildung 40: Charts der Ausgangssituation vor der 20jährigen Simulation. Die Nadelmasse für alle Segmente (o.l.) wird für alle Bäume des Bestandes angezeigt. Alle anderen Charts zeigen Werte für die Segmente des Baumes 65 an: Beschattungswerte kgC (u. l.), Nadeldichte kgC · m^{-1} (u. m.), Nadelmassen kgC (o. m.), relative Produktion (o. r.) und Produktiodichte kgC · m^{-1} (u. r.).

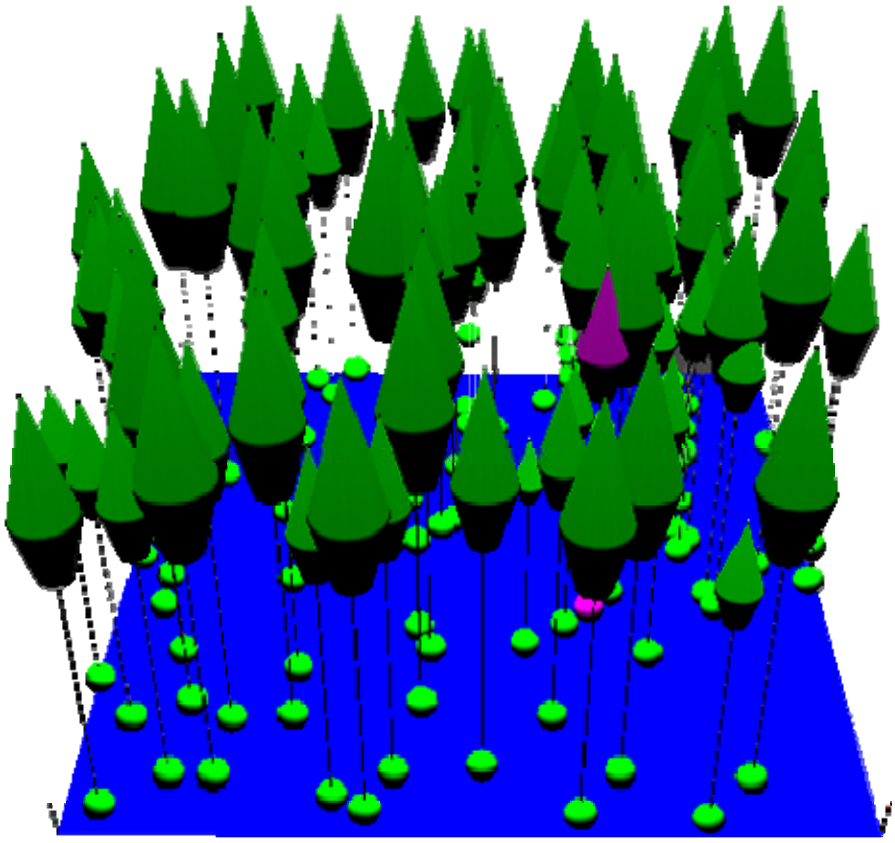


Abbildung 41: Räumliche Ausgangssituation der Variante B (93 Bäume auf 50x50 Meter). Baum 65 ist lila eingefärbt.

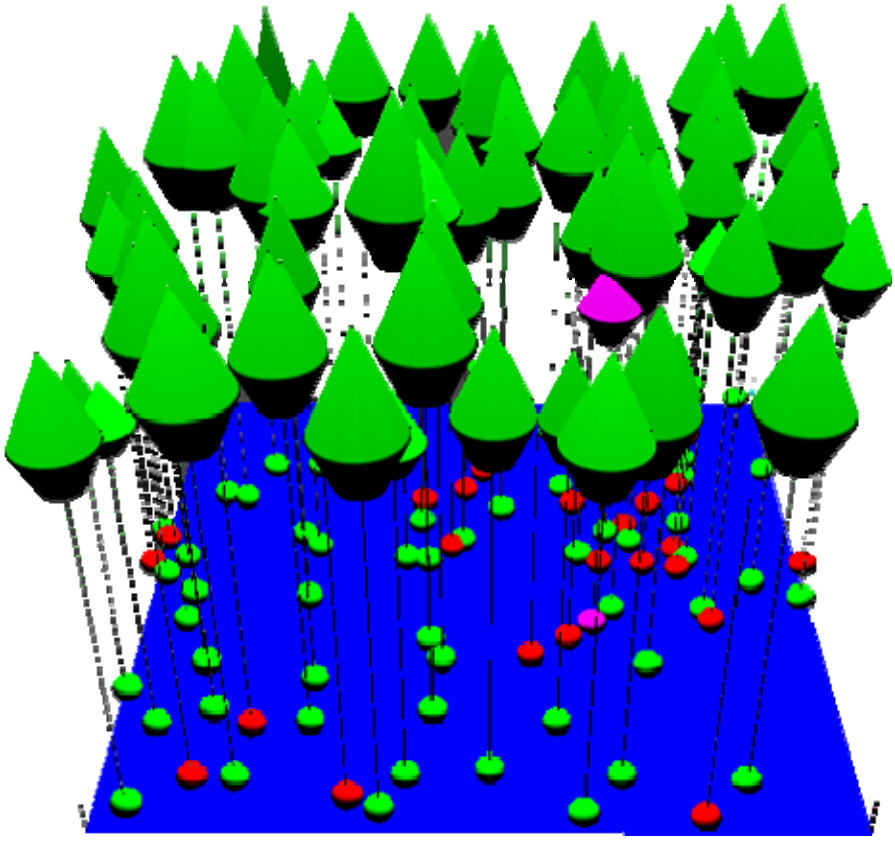


Abbildung 42: Räumliche Situation der Variante B nach der 20jährigen Simulation. Rote Basispunkte markieren abgestorbene Bäume. Baum 65 ist lila eingefärbt.

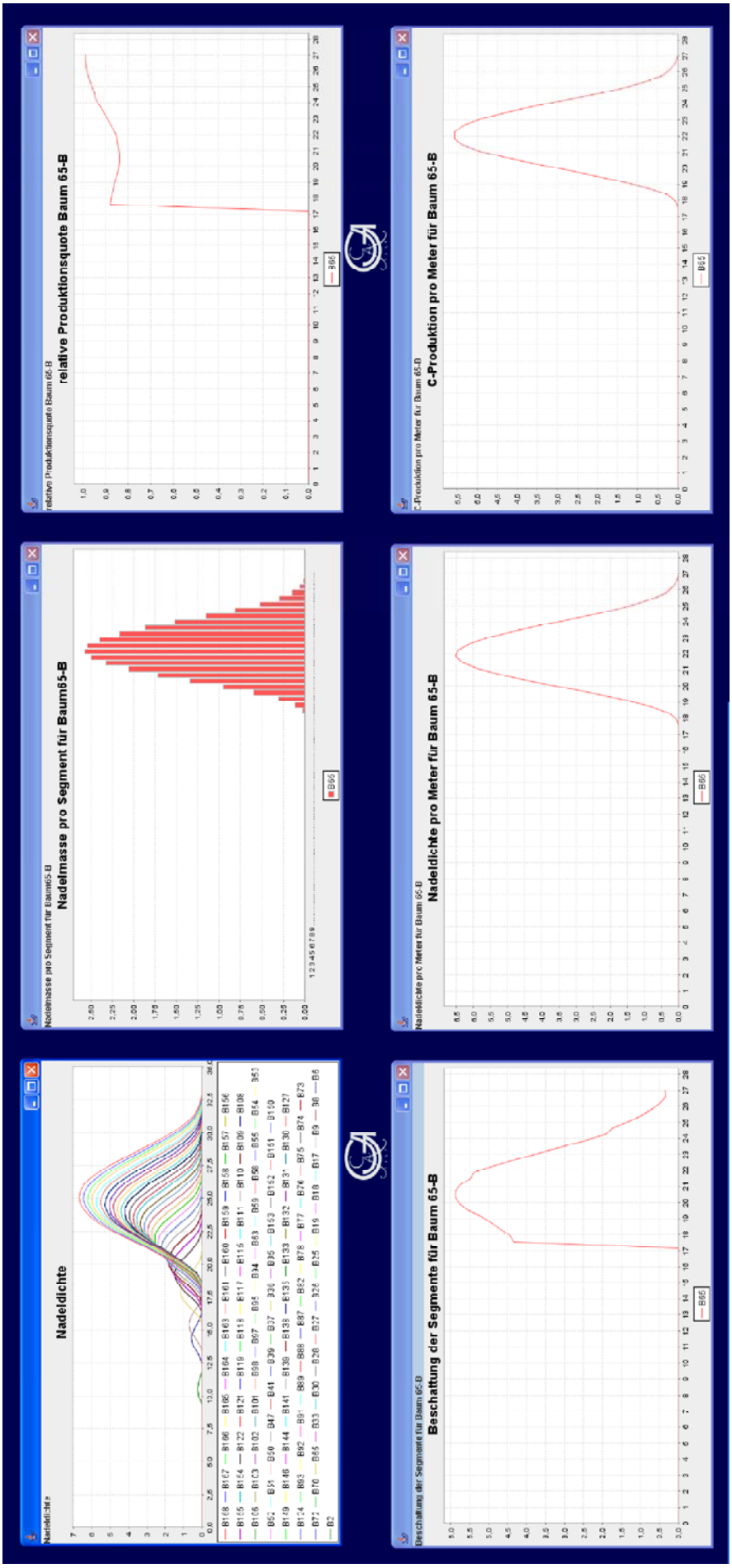


Abbildung 43: Charts der Ausgangssituation vor der Simulation. der variante B. Die Nadelmasse für alle Segmente (o.l.) wird für alle Bäume des Bestandes angezeigt. Alle anderen Charts zeigen Werte für die Segmente des Baumes 65 an: Beschattungswerte kgC (u. l.), Nadeldichte kgC · m^{-1} (u. m.), Nadelmassen kgC (o. m.), relative Produktion (o. r.) und Produktiodichte kgC · m^{-1} (u. r.).

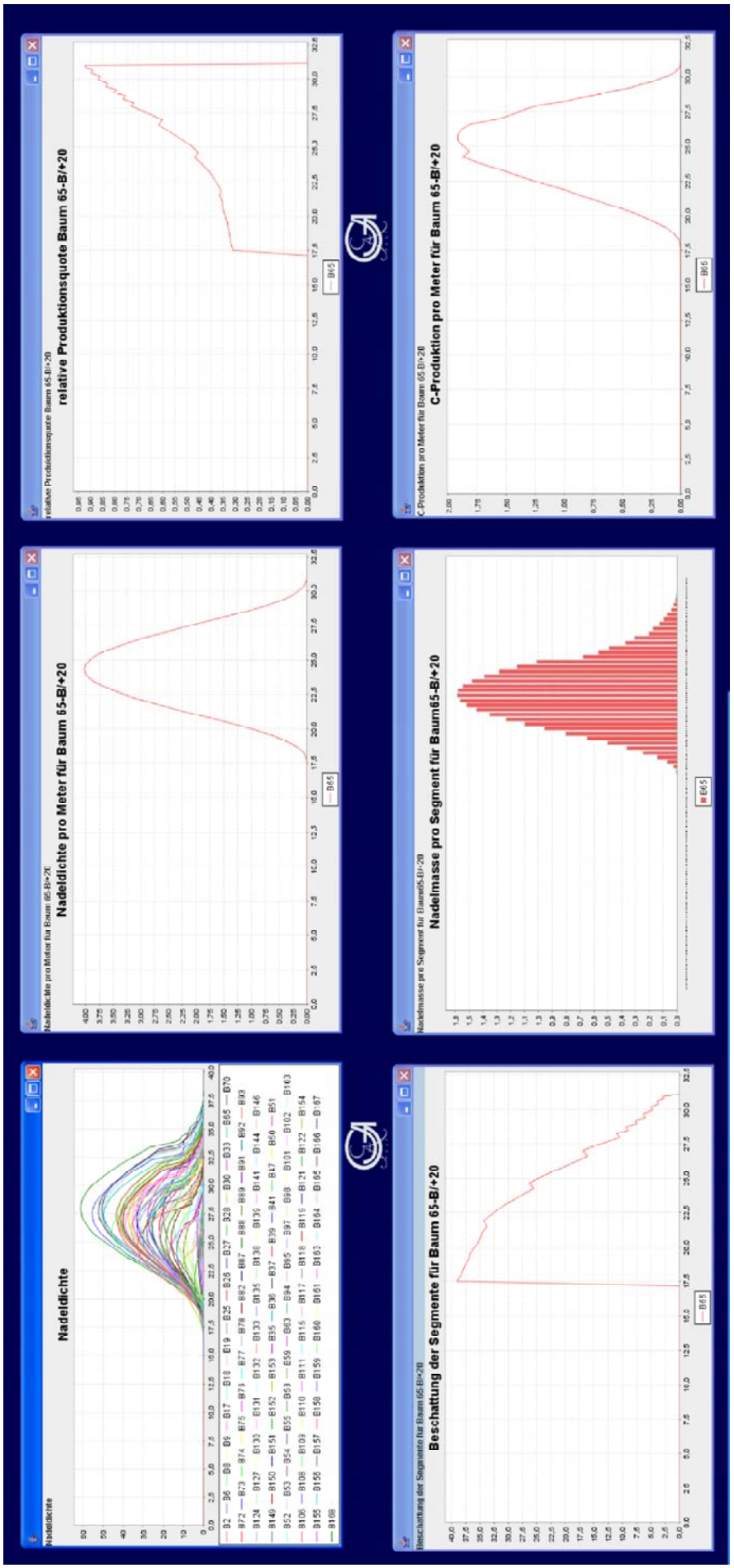


Abbildung 44: Charts der Situation nach der 20jährigen Simulation. der Variante B. Die Nadelmasse für alle Segmente (o.l.) wird für alle Bäume des Bestandes angezeigt. Alle anderen Charts zeigen Werte für die Segmente des Baumes 65 an: Beschattungswerte kgC (u. l.), Nadeldichte kgC · m^{-1} (u. m.), Nadelmassen kgC (o. m.), relative Produktion (o. r.) und Produktiodichte kgC · m^{-1} (u. r.).

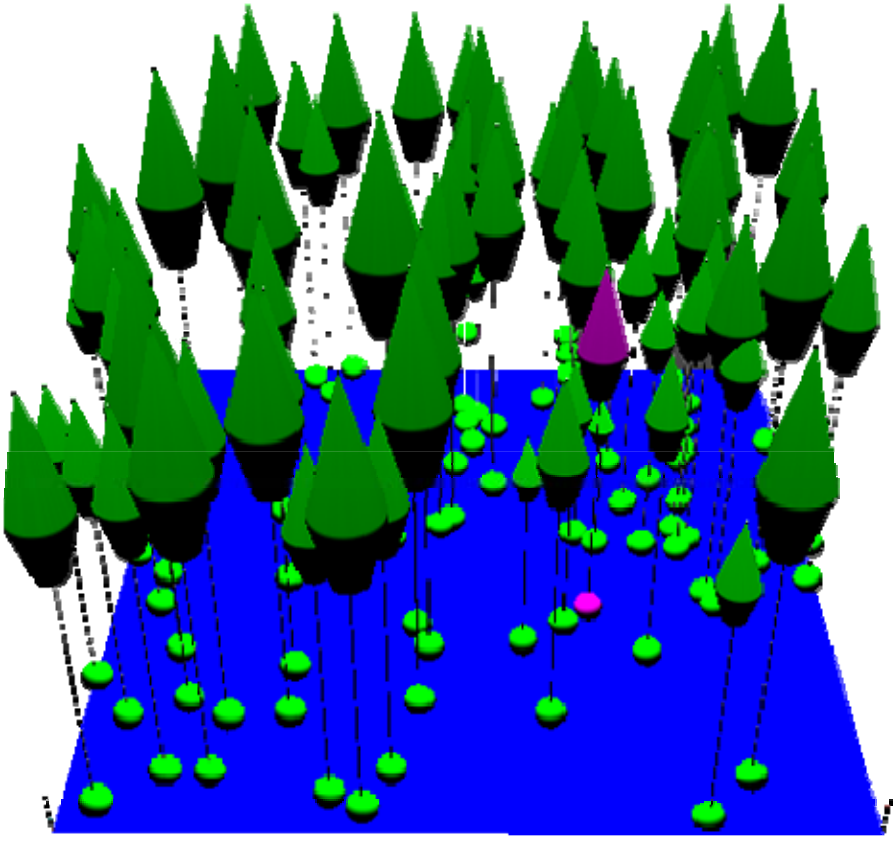


Abbildung 45: Räumliche Ausgangsituation der Variante C vor der Simulation (82 Bäume auf 50x50 Meter). Baum 65 (lila) wurde gezielt freigestellt.

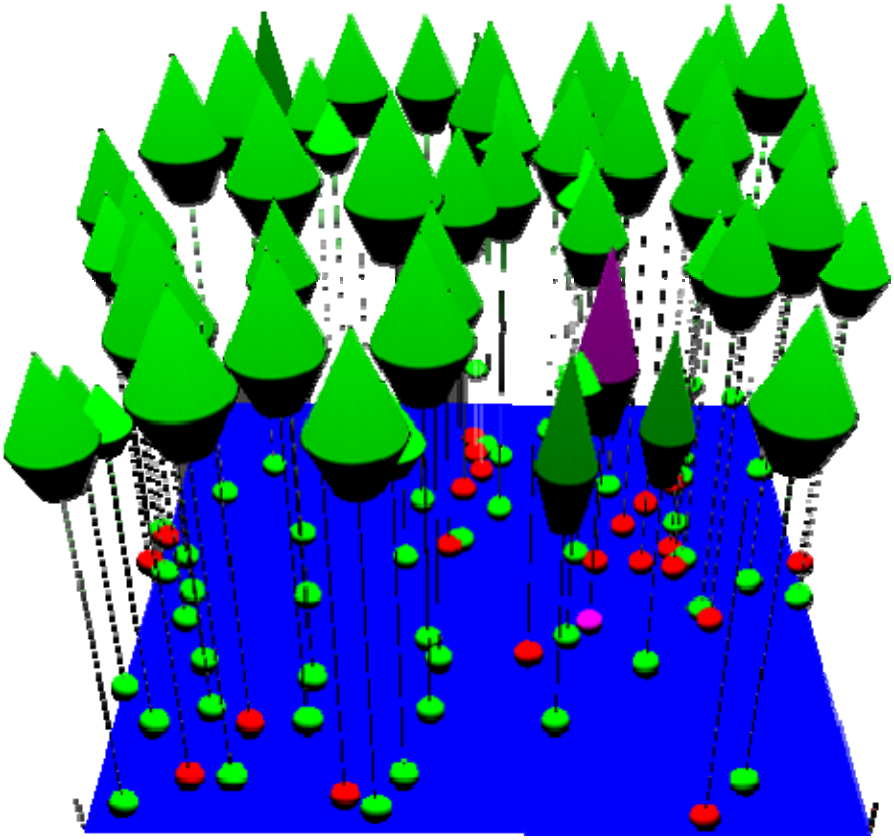


Abbildung 46: Räumliche Situation der Variante C nach der 20jährigen Simulation. Rote Basispunkte markieren abgestorbene Bäume. Baum 65 ist lila eingefärbt.

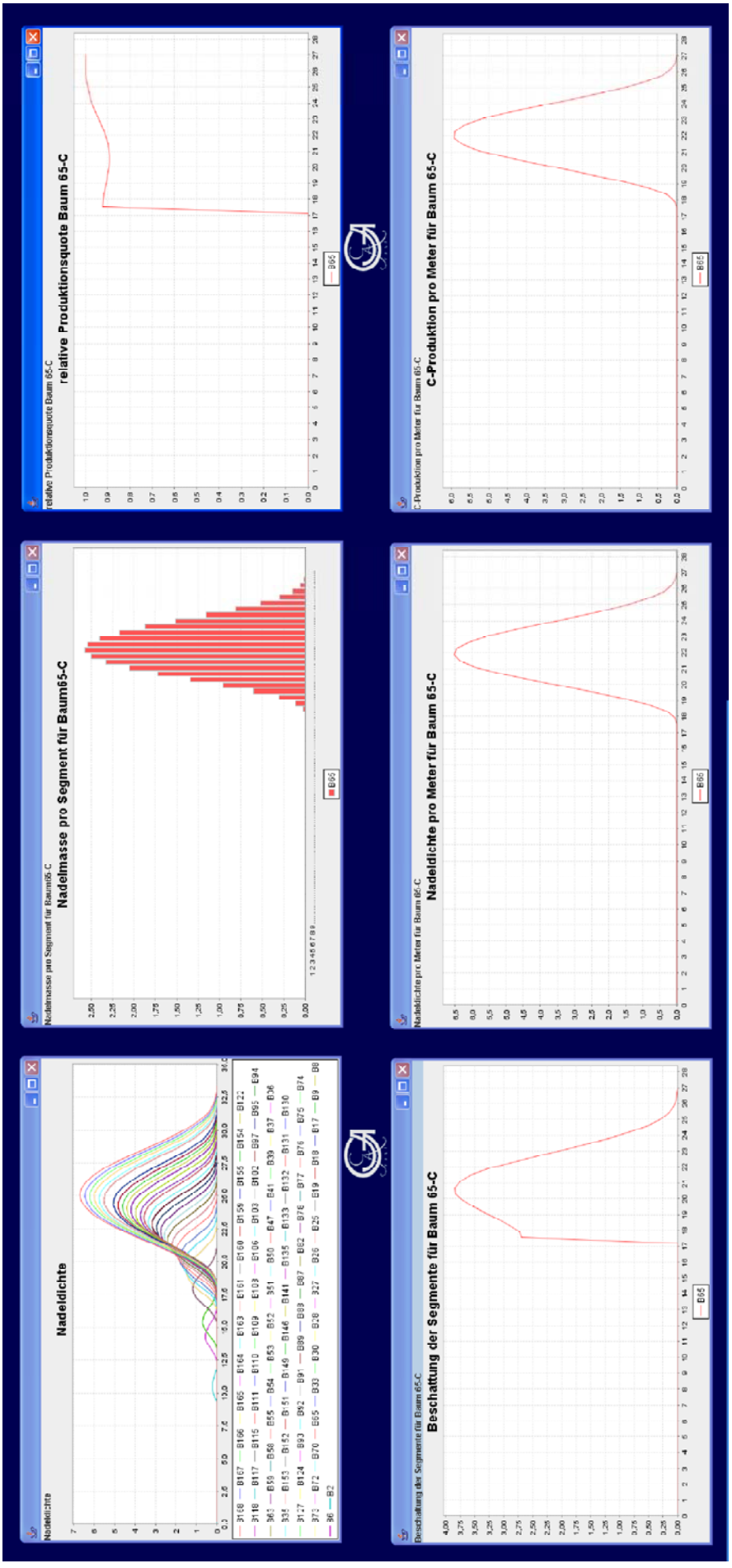


Abbildung 47: Charts der Ausgangssituation vor der Simulation. der Variante C. Die Nadelmasse für alle Segmente (o.l.) wird für alle Bäume des Bestandes angezeigt. Alle anderen Charts zeigen Werte für die Segmente des Baumes 65 an: Beschattungswerte kgC (u. l.), Nadeldichte kgC · m^{-1} (u. m.), Nadelmassen kgC (o. m.), relative Produktion (o. r.) und Produktiodichte kgC · m^{-1} (u. r.).

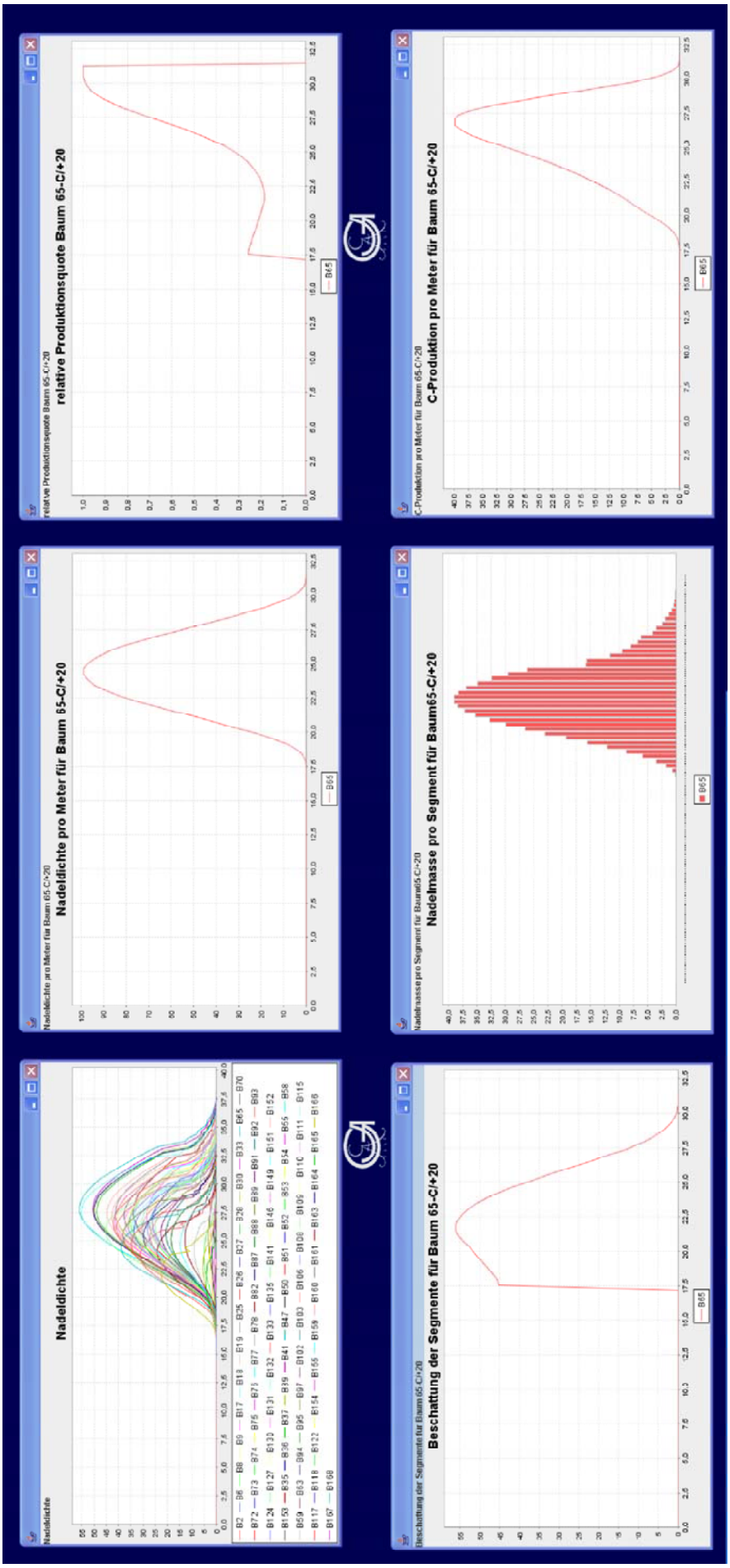


Abbildung 48: Charts Situation nach der 20jährigen Simulation. der Variante C. Die Nadelmasse für alle Segmente (o.l.) wird für alle Bäume des Bestandes angezeigt. Alle anderen Charts zeigen Werte für die Segmente des Baumes 65 an: Beschattungswerte kgC (u. l.), Nadeldichte kgC · m^{-1} (u. m.), Nadelmassen kgC (o. m.), relative Produktion (o. r.) und Produktiodichte kgC · m^{-1} (u. r.).