Interaktive Erzeugung fraktaler Landschaften auf Parallelrechnern

Diplomarbeit von Petra Buitmann

Universität Osnabrück

17. August 1995

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Verlauf der letzten Jahre hat die Computergraphik in der Datenverarbeitung einen immer bedeutenderen Stellenwert eingenommen. So ist sie bei der Arbeit in Berufen mit technischer Ausrichtung, wie z.B. bei der Erstellung von Bau- und Konstruktionsplänen oder der Simulation physikalischer Vorgänge nicht mehr wegzudenken. Einen erheblichen Aufschwung haben computergraphische Darstellungen aber vor allem in der graphischen Kunst, insbesondere bei der Erstellung von Filmen und Computerspielen erreicht. Hier werden mit dem Einsatz der Computergraphik bei der Erzeugung und Animation von Film- bzw. Spielobjekten große Erfolge erzielt.

Den meisten computergraphisch erzeugten Objekten haftet jedoch das Problem an, daß sie künstlich und damit unecht wirken. Eine Alternative bieten Fraktale. Aufgrund der ihnen innewohnenden Selbstähnlichkeit sind sie in der Lage, natürliche Objekte wie z.B. Berge, Bäume oder Wolken naturgetreu zu modellieren.

Da beim Einsatz der Computergraphik sehr komplexe Algorithmen zur Berechnung einer natürlich beleuchteten dreidimensionalen Szene verwendet werden, treten recht lange Laufzeiten auf. Bislang mußten daher teure Groß- bzw. Spezialrechner eingesetzt oder eine längere Rechenzeit in Kauf genommen werden.

Im Rahmen der hier vorliegenden Arbeit wird ein Computergraphikprogramm zur Erzeugung fraktaler Landschaften auf Parallelrechnern entwickelt. Die Komplexität der computergraphischen Algorithmen und die Unabhängigkeit der einzelnen Programmteile voneinander versprechen eine effiziente Parallelisierung und somit ein gutes Laufzeitverhalten. Als Basis der parallelen Programmierung dienen die einzelnen sequentiellen Programmteile, die gleichzeitig auf verschiedenen Prozessoren ablaufen und Daten untereinander austauschen. Vor dem Start des Programms können die Größe der Topologie und der zu verwendende Parallelrechner vom Benutzer ausgewählt werden.

Die Landschaften werden von fraktalen Bergen und beliebig vielen fraktalen Pflanzen gebildet, die vom Benutzer manipuliert werden können. Der Anwender besitzt die Möglichkeit, die Szene frei zu konfigurieren. Außerdem kann er den fokussierten Punkt, den Standpunkt des Betrachters, den darzustellenden Szenenausschnitt und die Position einer Lichtquelle festlegen.

Zur Eingabe und Veränderung der Daten durch den Benutzer wird eine interaktive Oberfläche entwickelt. Mit Hilfe dieser Oberfläche werden außerdem das sequentielle sowie das parallele Programm gestartet und das berechnete Bild in einem Fenster dargestellt.

Mit der Erzeugung fraktaler Berge und fraktaler Pflanzen befaßt sich das folgende Kapitel. Es umfaßt eine Einführung in das Thema Fraktale und durch Skizzen erläuterte Darstellungen der verwendeten Algorithmen. Da die fraktalen Pflanzen mit Hilfe von Lindenmayer-Systemen und einer Turtle-Graphik generiert werden, enthält dieses Kapitel ebenfalls einige allgemeine Anmerkungen zu diesen Themengebieten.

Das Kapitel 3 geht auf die Abbildung der dreidimensionalen Szene auf einen zweidimensionalen Bildschirm ein. Es werden die algorithmischen Grundlagen für die Darstellung einer 3D-Szene aus beliebigen Blickwinkeln vorgestellt und die verschiedenen Techniken zur Unterdrückung von Rückflächen, zum Abschneiden nicht sichtbarer Objektteile und zum Schattieren von Flächen erläutert.

Im Mittelpunkt des 4. Kapitels steht die Parallelisierung des in Kapitel 2 und 3 entwickelten sequentiellen Programms. Hier wird nach einer einleitenden Darstellung der verwendeten parallelen Hard- und Software die parallele Version des Programms dargestellt, die unter der Laufzeitumgebung Parix entwickelt wird. Das parallele Programm ist auf Topologien mit 4, 16, 32 und 64 Prozessoren lauffähig. Um die Effizienz der Parallelisierung zu ermitteln, werden die Meßergebnisse auf diesen Topologien mit dem Laufzeitverhalten des sequentiellen Programms verglichen.

Die Funktionsweise der interaktiven Oberfläche zur Bedienung des Programms erläutert Kapitel 5. Darin wird zu Beginn eine Darstellung des X Window Systems und des OSF/Motif-Widget-Sets vorgenommen, unter denen die Oberfläche implementiert wurde, und anschließend auf die Handhabung derselben eingegangen.

Eine Zusammenfassung sowie ein Ausblick auf mögliche Erweiterungen des Programms schliessen die Arbeit ab. Der darauf folgende Anhang enthält die Grammatik der verwendeten Lindenmayer-Systeme, eine Installationsanweisung des Programms, ein Beispiel für die Beschreibung einer fraktalen Szene sowie eine Aufzählung aller Programmquellen.

Kapitel 2

Fraktale

Ein *Fraktal* ist ein Objekt, das eine geometrische *Selbstähnlichkeit* aufweist, d.h. das gleiche Motiv wiederholt sich in stets kleinerem Maße. Wird ein beliebig kleiner Ausschnitt des Fraktals gewählt, so finden sich darin alle Elemente des Ganzen wieder.

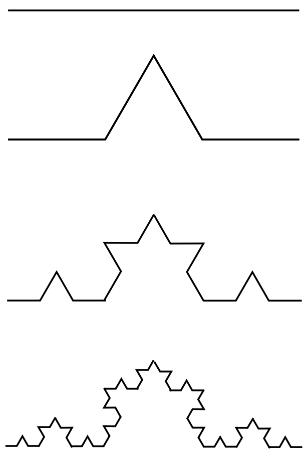

Abbildung 2.1: Koch'sche Kurve bis zur dritten Iteration

Die erste Definition des Begriffs *Fraktal* stammt von Benoît B. Mandelbrot [Man87]. Er bezeichnet damit eine Figur, deren Hausdorff-Besicovitch-Dimension echt die topologische Dimension übersteigt. Dabei darf diese Dimension gebrochen sein. Sie wird daher auch *fraktale Dimension* genannt. So besitzt z.B. die *Koch'sche Kurve* (siehe Abbildung 2.1) die fraktale Dimension D = 1,2618. Sie entsteht aus einem Liniensegment, dessen mittlerer Teil entfernt und durch ein gleichseitiges Dreieck ersetzt wird. Wird dieser Prozeß wiederholt, so erhält man eine Figur, die zwar aus einer Linie besteht, aber beinahe eine Fläche bildet. Von daher ist eine Dimension $1 < D < 2$ gerechtfertigt.

Einige auf ähnliche Weise gebildete Figuren, z.B. die *Lévy-Kurve* aus Abbildung 2.2, dürfte man laut dieser Definition allerdings nicht als Fraktale bezeichnen. Sie besitzt die fraktale und topologische Dimension $D = 1$, d.h. ihre fraktale Dimension ist nicht echt größer als ihre topologische Dimension. Anhand ihrer Struktur würde man sie aber als Fraktal bezeichnen wollen. Daher wurde der Begriff des Fraktals auf alle geometrischen Objekte, die Selbstähnlichkeit aufweisen, ausgeweitet. Bei der Koch'schen Kurve ist diese Selbstähnlichkeit sehr gut zu erkennen. Wird nach mehreren Iterationen ein Teil vergrößert, so wiederholt sich in diesem das ursprüngliche Motiv des unterteilten Liniensegments.

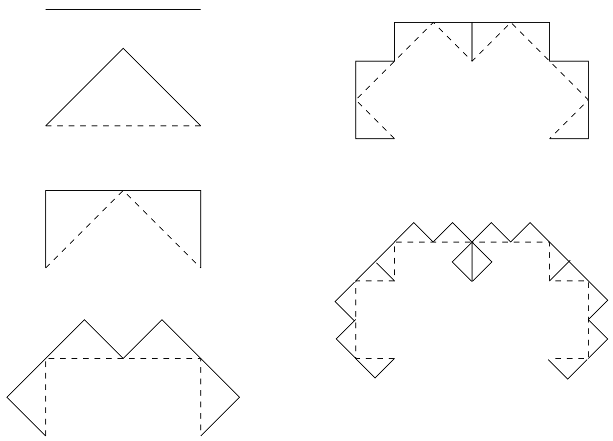

Abbildung 2.2: Lévy-Kurve

Große Ähnlichkeit mit der Natur weisen die *stochastischen Fraktale* auf. Sie werden erzeugt, indem kleine *Störungen* bei der Bildung des Objektes angebracht werden. Aufgrund der Gesetze der Wahrscheinlichkeitsrechnung besitzt aber ein solches Fraktal in allen Größenbereichen denselben Grad an Irregularität, d.h. der Begriff der *Selbstähnlichkeit* wird nicht verletzt, sondern erweitert. Zur vertiefenden Literatur des Themas *Fraktale* siehe [Pei88] und [La1/92, La2/92].

2.1 Erzeugung eines fraktalen Berges

Ein schneller Algorithmus zur Generierung eines fraktalen Berges besteht in der schrittweisen Verfeinerung eines Gitters von Höhenwerten (siehe Abbildung 2.3). Ausgangspunkt ist ein Quadrat beliebiger Größe, das als Grundfläche dient. Darauf entsteht ein Netz von Dreiecken, welches den Berg beschreibt.

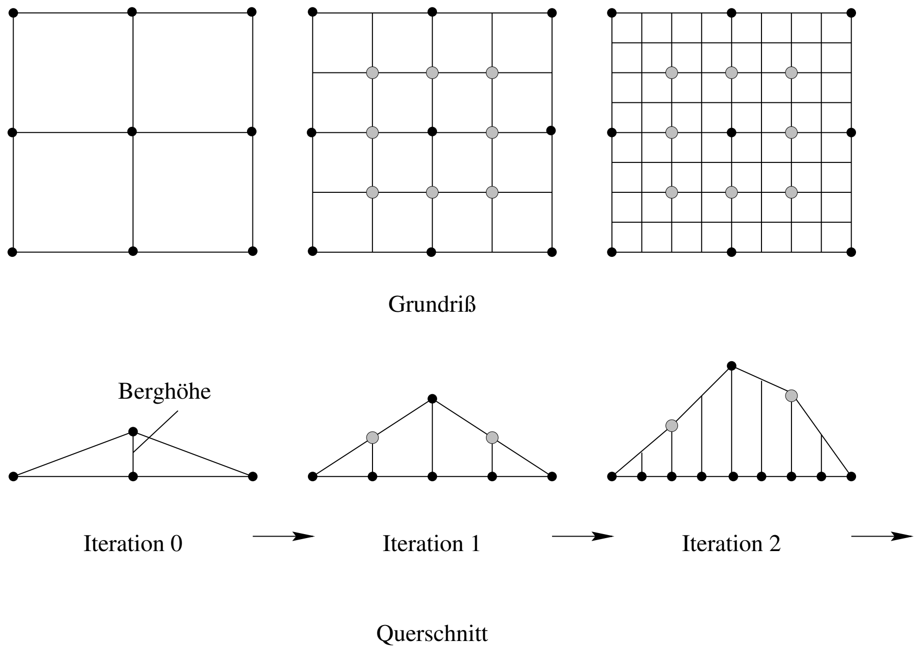

Abbildung 2.3: Verfeinerung eines Gitters von Höhenwerten

Zur Generierung des Gitternetzes wird zu Beginn ein Feld von 3×3 Punkten erzeugt, wobei dem Mittelpunkt ein beliebiger positiver Höhenwert (die sog. *Berghöhe*) zugewiesen wird. Durch den Wert der Berghöhe kann die spätere absolute Höhe des Berges, also die *Bergspitze*, beeinflußt werden. Alle Randpunkte erhalten den Wert Null, da der Berg zu allen Seiten flach auslaufen soll.

In einer frei wählbaren Anzahl von Iterationen wird nun das Gitter verfeinert, und es werden „Stützpunkte“ zur Beschreibung der Bergstruktur erzeugt. Auf diese Weise wird ein Feld von Höhenwerten über der quadratischen Grundfläche angelegt.

Dabei wird in jedem Schritt das Höhenfeld der Größe $(2^n+1) \times (2^n+1)$ zu einem Feld der Größe $(2^{n+1}+1) \times (2^{n+1}+1)$ erweitert, wobei n den Iterationsschritt bezeichnet. D.h., zwischen den bereits bestehenden Punkten werden $2^n \times 2^n$ weitere eingefügt (Abbildung 2.3). Nun wird zu den Höhenwerten der Punkte des „alten“ Gitters die oben erwähnte Berghöhe, multipliziert mit einem Zufallswert, addiert. Mit Hilfe des Zufallswertes wird die Regelmäßigkeit der Bergstruktur gestört, und der Berg wirkt natürlich.

Durch Interpolation der bereits berechneten Nachbarwerte erhält man die Höhenwerte der neu hinzugekommenen Punkte (siehe Abbildung 2.4). Dabei können die beiden Nachbarwerte mit unterschiedlichen Interpolationsgewichten in die Berechnung eingehen. Die Summe der Inter-

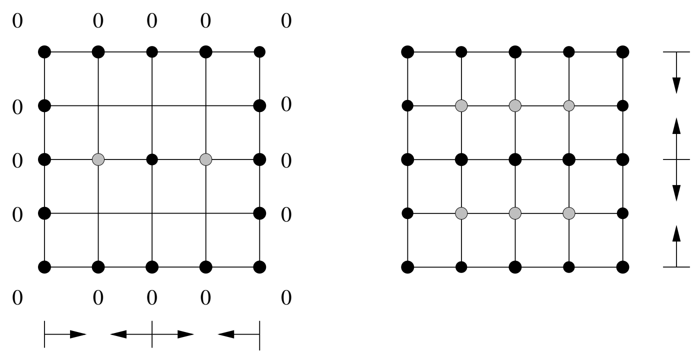

Abbildung 2.4: Interpolation neuer Höhenwerte

polationsgewichte beträgt 1. Nach jeder Iteration wird die *Berghöhe*, die auf die Höhenwerte addiert wird, verkleinert, um den Berg nicht zu spitz werden zu lassen.

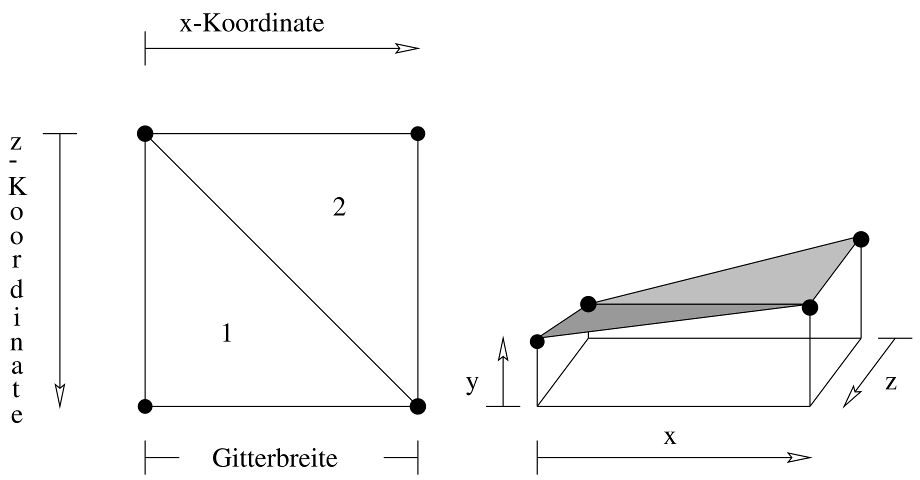

Abbildung 2.5: Umrechnung der Gitterwerte in 3D-Koordinaten

Das nach obigem Algorithmus entstandene zweidimensionale Feld von Höhenwerten ist nun in eine Menge von dreidimensionalen Punkten (*3D-Koordinaten*) umzurechnen, um die Daten des Berges plastisch darstellen zu können (siehe Abbildung 2.5).

Dazu wird der erste Feldindex, multipliziert mit der *Gitterbreite* als z-Koordinate und der zwei-

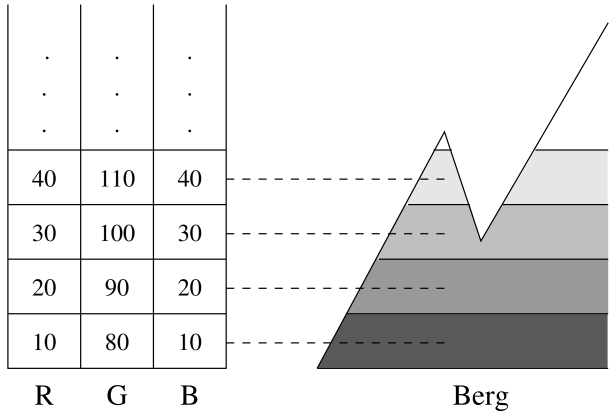

Abbildung 2.6: Abbildung der Farbwerte auf Berghöhen

te Feldindex, mit demselben Wert multipliziert, als x-Koordinate des Punktes festgelegt. Die Gitterbreite entspricht dem Abstand zwischen zwei Punkten des Gitters. Aus dem Feldeintrag, also dem Höhenwert, der durch den obigen Algorithmus berechnet wurde, wird die y-Koordinate gebildet.

Da der Berg später farblich dargestellt werden soll, muß jedem Punkt außer den 3 Koordinaten ein Farbwert zugewiesen werden. Dazu wird die Menge der Höhenwerte der Punkte ($0 \leq Höhe \leq Bergspitze$) auf ein Feld von 32 Farbwerten abgebildet (siehe Abbildung 2.6). Dieses Farbfeld enthält Einträge der Farben eines natürlichen Berges (von grün über braun und grau zu weiß). Der Bergspitze soll der höchste Weißwert (Schnee) und dem Fuß des Berges der niedrigste Grünwert (Vegetation) zugewiesen werden. Anhand der Bergspitze wird daher eine Spanne berechnet, die die Breite des Höhenbereiches angibt, der einem Farbeintrag zugeordnet wird. Nun wird für den Höhenwert (y-Koordinate) jedes Punktes der entsprechende Index im Farbfeld berechnet und dem Punkt der darin gespeicherte Farbwert zugewiesen. Bei den Farbeinträgen handelt es sich um sogenannte *RGB-Farbwerte* (siehe [Fel92] und [Vor94]).

Nachdem alle Punkte sowohl räumlich als auch farblich beschrieben wurden, wird aus jeweils drei dieser Punkte ein Dreieck gebildet, so daß der Berg in Form eines Gitternetzes über der quadratischen Grundfläche liegt.

Die fraktale Eigenschaft dieses Objektes ist zu erkennen, wenn ein vergrößerter Ausschnitt des Berges betrachtet wird (siehe Abbildung 2.7). Die Gitterstruktur bleibt erhalten und unterscheidet sich nur im Grad der Verfeinerung von dem Gesamtobjekt. Außerdem sind wie beim großen Berg „Bergspitzen“ und „Täler“ sichtbar. Da die Berechnung der Höhenwerte den Gesetzen des Zufalls unterworfen ist (*stochastisches Fraktal*), sind die Höhenstruktur des Ausschnitts und des gesamten Berges zwar unterschiedlich, aber die Selbstähnlichkeit bleibt erhalten. Sowohl der

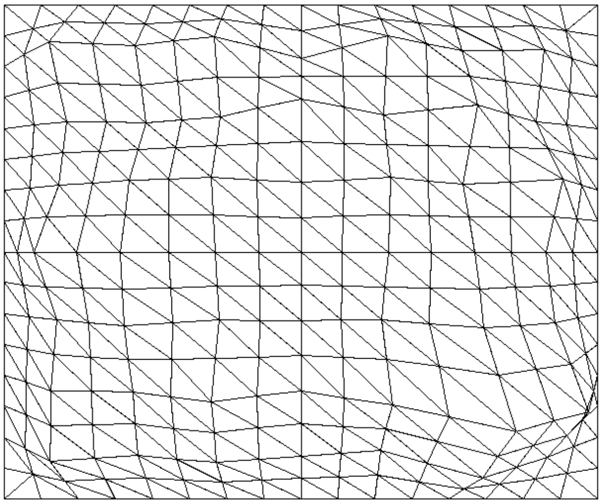

Abbildung 2.7: Fraktaler Berg als Gitterstruktur von oben betrachtet

große Berg als auch jeder Bergausschnitt bestehen aus einer Vielzahl von kleineren oder größeren Hügeln.

2.2 Erzeugung fraktaler Pflanzen

2.2.1 Lindenmayer-Systeme

Lindenmayer-Systeme — kurz auch *L-Systeme* genannt — wurden 1968 von dem Biologen *Aristid Lindenmayer* als theoretisches Hilfsmittel zur Untersuchung der Entwicklung multizellularer Organismen eingeführt. Die geometrische Interpretation der L-Systeme führte später dazu, daß dieses Konzept auch zur Erforschung höherer Pflanzen eingesetzt wurde. Es handelt sich hierbei um einen mathematischen Formalismus, dessen Idee darin besteht, aus einem einfachen Ausgangsobjekt ein komplexes Endobjekt zu schaffen, indem Teile anhand von sogenannten *Produktionen* ersetzt werden. Die Anwendung der Produktionen geschieht parallel. Ein Beispiel für das Ergebnis eines L-Systems ist die bereits erwähnte Koch-Kurve in Abbildung 2.1. Dort werden in jedem Schritt gleichzeitig alle geraden Linien durch Linien, deren Mittelstück ein gleichseitiges Dreieck ohne Grundlinie ist, ersetzt.

Formal beschrieben wird ein Lindenmayer-System durch ein Tripel $< V,\ \omega,\ P >$. Dabei bezeichnet V ein Alphabet, d.h. eine Menge von Buchstaben, V^* die Menge aller Wörter, die aus V gebildet werden können und V^+ die Menge aller nicht leeren Wörter aus V. Weiterhin ist $\omega \in V^+$ das initiale Objekt, genannt *Axiom* und $P \subset V \times V^*$ die Menge aller *Produktionen*, die zur Ableitung benutzt werden. Eine *Produktion* $(a, \chi) \in P$ wird mit $a \to \chi$ bezeichnet. Ihr erstes Argument wird *Vorgänger*, ihr zweites *Nachfolger* genannt. Bei der Ableitung wird zu jedem Buchstaben eine Produktion gesucht, bei der dieser Buchstabe Vorgänger ist, und er wird durch seinen Nachfolger ersetzt. Ist zu einem Buchstaben keine Produktion zu finden, so wird auf ihn die *identische Produktion* $a \to a$ angewendet.

Ein Beispiel für ein L-System ist das Tripel $G =< V, \omega, P >$ mit

- $\mathtt{V} = \{\mathtt{F}, \mathtt{f}, +, -\}$
- $\omega = \{\mathtt{F}\}$
- $\mathtt{P} = \{\mathtt{F} \to \mathtt{Ff}, \mathtt{f} \to +\mathtt{F} - \mathtt{F}\}$.

Die Ableitung bis zum 4. Schritt erfolgt dann folgendermaßen:

0. Schritt:	`F`
1. Schritt:	`Ff`
2. Schritt:	`Ff+F-F`
3. Schritt:	`Ff+F-F+Ff-Ff`
4. Schritt:	`Ff+F-F+Ff-Ff+Ff+F-F-Ff+F-F`.

2.2.2 Turtle-Graphik

Zur geometrischen Interpretation einer so erzeugten Zeichenkette dient die *Turtle-Graphik*. Eine imaginäre Schildkröte (siehe Abbildung 2.8) faßt die einzelnen Zeichen als Befehle auf, ihre Position oder ihre Laufrichtung zu verändern.

Durch ihre Bewegungen erzeugt sie die durch den String vorgegebene Graphik. Der Zustand der Turtle wird durch ein Tripel (x, y, α) beschrieben, wobei (x, y) die kartesischen Koordinaten in der Ebene und α die Laufrichtung (Winkel) darstellt. Außerdem sind die *Schrittweite* d und der *Drehwinkel* δ festgelegt. Alle Werte werden zu Beginn initialisiert, d.h. der Turtle wird eine Ausgangsposition in der Ebene und eine initiale Laufrichtung zugewiesen. Die oben benutzten Zeichen `F, f, +, -` entsprechen folgenden Graphikbefehlen:

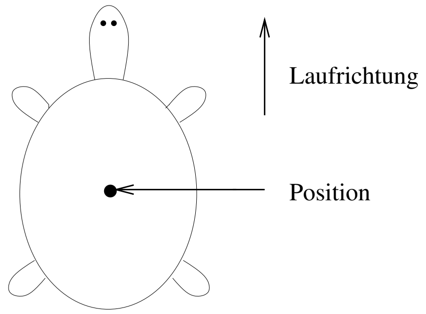

Abbildung 2.8: Imaginäre Turtle zur graphischen Interpretation

`F`	gehe um eine Längeneinheit d in Laufrichtung und zeichne eine Linie
`f`	gehe um eine Längeneinheit d in Laufrichtung, ohne zu zeichnen
`+`	drehe nach links um den Drehwinkel δ
`-`	drehe nach rechts um den Drehwinkel δ.

Wendet man diese Graphikbefehle auf den 4. Schritt des obigen Beispiels mit $d = 1$ und $\delta = 90°$ an, so ergibt sich das Bild in Abbildung 2.9.

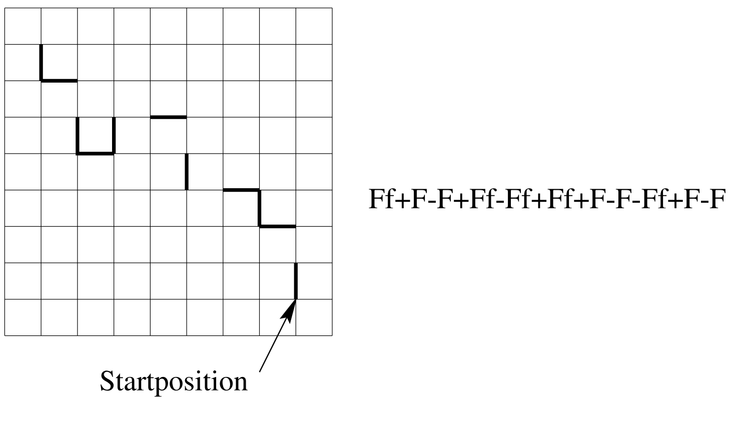

Abbildung 2.9: Graphische Darstellung eines L-Systems

Zur Simulation von Pflanzen reicht allerdings eine zweidimensionale Graphik nicht aus. Daher werden Position und Laufrichtung der Turtle sowie die Graphikbefehle auf das Dreidimensionale erweitert (siehe Abbildung 2.10). Die Turtle wird beschrieben durch

(x, y, z)	Position im Raum
(h_x, h_y, h_z)	Vektor geradeaus (Laufrichtung)
(l_x, l_y, l_z)	Vektor links
(u_x, u_y, u_z)	Vektor oben
d	Schrittweite
δ	Drehwinkel
n	Ableitungslänge.

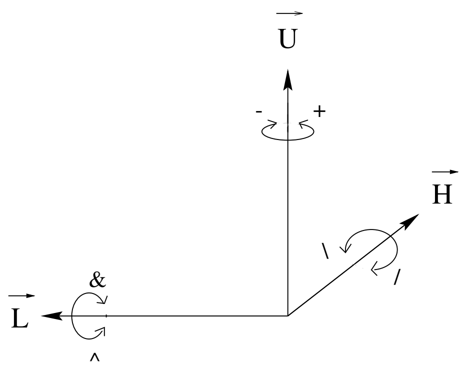

Abbildung 2.10: Bewegungsvektoren für eine Turtle im Dreidimensionalen

Durch die Erweiterung der Turtle kommen auch neue Befehle zur Bewegung im Dreidimensionalen hinzu. Eine Auflistung aller Graphikbefehle befindet sich in Anhang A.

2.2.3 Geklammerte L-Systeme

Mit den oben beschriebenen Befehlen ist es möglich, die Turtle im Dreidimensionalen zu bewegen. Allerdings ist sie ausschließlich in der Lage, Linienteile zu zeichnen. Damit kann keine für einen Betrachter interessante Pflanze erstellt werden. Um real aussehende Bäume und Pflanzen modellieren zu können, bedarf es eines Hilfsmittels, das die Verzweigung der Linien ermöglicht. Auf diese Weise könnten *Äste* und *Zweige* simuliert werden. Diese Möglichkeit bieten *geklammerte L-Systeme*. Sie benutzen die beiden zusätzlichen Zeichen

- `[`, um sich die momentane Position der Turtle zu merken, und

- `]`, um zur letzten gemerkten Position zurückzuspringen.

Dadurch kann die Turtle einen Seitenzweig zeichnen und anschließend zur Ausgangsposition am Stamm zurückkehren. Die Klammern dürfen beliebig ineinander geschachtelt werden. Allerdings ist auf korrekte Klammerung zu achten.

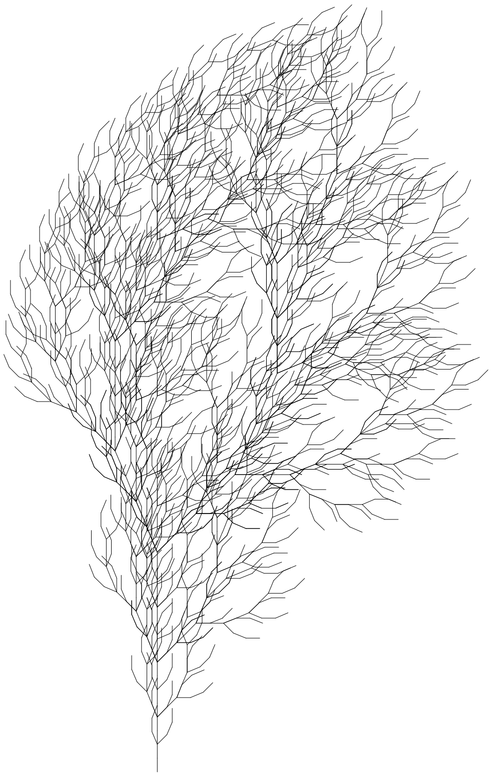

Abbildung 2.11: Ergebnis eines geklammerten L-Systems

Ein Beispiel für ein solches L-System ist in Abbildung 2.11 dargestellt. Das zugehörige Lindenmayer-System lautet:

n	4
δ	22.5°
d	4.0
ω	F
P	F $\rightarrow$ FF-[-F+F+F]+[+F-F-F]

2.2.4 Kontext-sensitive L-Systeme

Eine weitere Verbesserung sind die *kontext-sensitiven L-Systeme*. Durch sie ist es möglich, Produktionen nur in bestimmten Fällen anzuwenden und *Signale*, z.B. zum Blühen, durch die Pflanze zu schicken. Die bisher behandelten L-Systeme werden auch als 0L-Systeme bezeichnet, da die Anwendung einer Produktion auf ein Zeichen unabhängig vom Kontext des Zeichens, d.h. unabhängig von seinen Nachbarn, geschieht. Bei den sogenannten 1L-Systemen bzw. 2L-Systemen muß der Vorgänger in einem bestimmten Kontext stehen, um durch den Nachfolger ersetzt werden zu dürfen. Dieser Kontext besteht bei den 1L-Systemen aus dem linken *oder* dem rechten Nachbarn und bei den 2L-Systemen aus beiden Nachbarn. Die Schreibweise der Produktionen ändert sich folgendermaßen:

1L-Systeme	$a_l < a \to \chi$ oder $a > a_r \to \chi$
2L-Systeme	$a_l < a > a_r \to \chi$.

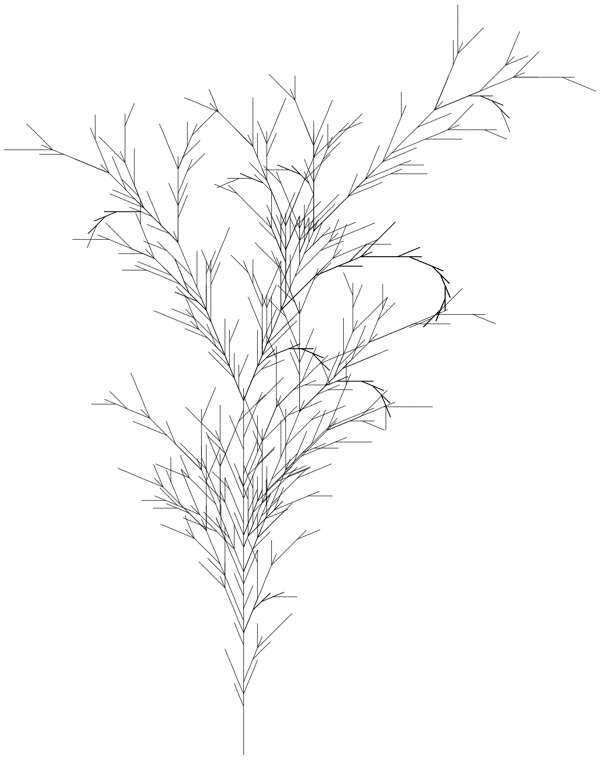

Abbildung 2.12: Ergebnis eines kontext-sensitiven L-Systems

Es kann eine Menge von Zeichen angegeben werden, die bei der Kontextsuche ignoriert werden sollen. Falls kein Kontext vorgegeben werden soll, ist an den entsprechenden Stellen ein $*$ zu benutzen. Abbildung 2.12 zeigt das kontext-sensitive L-System:

n	30
δ	22.5°
d	4.0
ignore	+-F
ω	F1F1F1
P	0 < 0 > 0 → 0
	0 < 0 > 1 → 1[-F1F1]
	0 < 1 > 0 → 1
	0 < 1 > 1 → 1
	1 < 0 > 0 → 0
	1 < 0 > 1 → 1F1
	1 < 1 > 0 → 1
	1 < 1 > 1 → 0
	* < + > * → -
	* < - > * → +

2.2.5 Stochastische L-Systeme

Da L-Systeme nach festen Regeln (Produktionen) arbeiten, ist das Erscheinungsbild eines L-Systems immer dasselbe. In der Natur sieht aber niemals eine Eiche wie die andere aus, da viele Merkmale dem *Zufall* (Wetter, Standort etc.) überlassen bleiben. Unterschiedliche Ausprägungen desselben Baumes erhält man durch *stochastische L-Systeme*, die durch ein Viertupel $G_\pi = < V,\ \omega,\ P,\ \pi >$ dargestellt werden. Jeder Vorgänger kann dabei mehrere Nachfolger haben, wobei jedem Nachfolger durch die Wahrscheinlichkeitsverteilung $\pi : P \rightarrow (0, 1]$ eine Wahrscheinlichkeit zugewiesen wird. Die Summe aller Wahrscheinlichkeiten beträgt 1.

In Abbildung 2.13 werden mehrere Ausprägungen desselben L-Systems gezeigt. Es lautet:

n	5
δ	22.5°
d	2.0
ω	F
P	F → (.39) F[+F]F[-F]F
	F → (.27) F[+F]F
	F → (.34) F[-F]F

2.2.6 Erweiterungen

Um einen realistischen Baum mit Blättern darstellen zu können, wurden vier weitere Graphikbefehle hinzugefügt. Ein Baumstamm besteht nicht nur aus einer Linie. Aus diesem Grunde kann

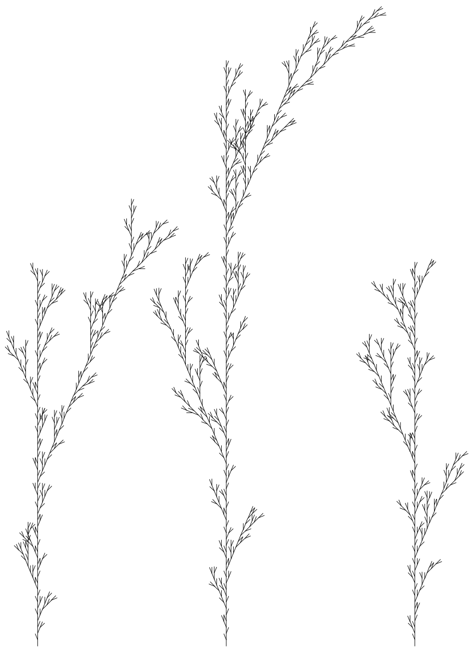

Abbildung 2.13: Ergebnisse eines stochastischen L-Systems

man mit dem Zeichen `G` einen Zylinder erzeugen. Für die Turtle bedeutet dies, daß sie zwar um eine Längeneinheit in Laufrichtung vorläuft, aber dabei keine Linie, sondern ein „Rohr" zeichnet. Der Radius des Zylinders ist abhängig von seiner Länge. D.h., werden mehrere solcher Zeichen aneinandergehängt, so wird der Stamm nicht nur länger, sondern auch dicker. Auf diese Weise ist es möglich, einen Stamm von einem Ast und einem Zweig zu unterscheiden. Graphisch angenähert wird der Zylinder durch acht Rechtecke (siehe Abbildung 2.14).

Zur Generierung eines Polygons wurden die Zeichen

- `{` Beginn eines neuen Polygons,
- `.` Setzen eines Polygonpunktes und
- `}` Schließen des Polygons

eingeführt. Bei Auftreten des `{`-Zeichens merkt sich die Turtle ihre momentane Position, läuft anschließend die übrigen, durch `.` gekennzeichneten Polygonpunkte ab und springt bei Erscheinen des `}`-Zeichens zum Anfangspunkt des Polygons zurück. Die Polygone ermöglichen die Darstellung von Blättern und Blüten.

Zur farblichen Gestaltung der Lindenmayer-Systeme müssen auch den Punkten, die von der Turtle beschrieben werden, RGB-Farbwerte zugewiesen werden. Dazu ist in einem Farbfeld eine Menge von RGB-Werten gespeichert. Die imaginäre Schildkröte wird mit einem Index dieses

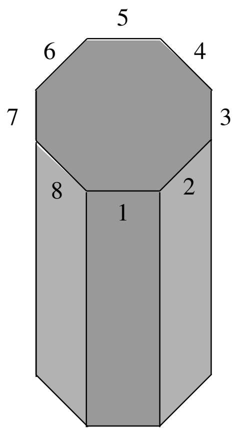

Abbildung 2.14: Beschreibung eines Zylinders

Feldes initialisiert, der einen Grünton enthält. Zur Manipulation des Farbwertes umfaßt das L-System die Befehle

- ' erniedrige Farbindex und
- ‘ erhöhe Farbindex.

Dadurch können aus dem Farbfeld andere RGB-Werte gewählt werden. Jedem Punkt, den die Turtle im Raum anläuft, wird der zum jeweiligen Index gehörende Farbwert zugewiesen. Da mit dem Zylinder ein Stamm oder Ast dargestellt werden soll, erhalten die Punkte der acht Rechtecke generell — unabhängig vom aktuellen Farbindex — einen Braunton.

Durch die oben ausgeführten Erweiterungen der herkömmlichen Lindenmayer-Systeme können recht natürliche Pflanzen modelliert werden. Diese Realitätsnähe besitzen L-Systeme aufgrund der Selbstähnlichkeit, durch die Fraktale charakterisiert werden. Schon Mandelbrot erkannte *die fraktale Geometrie der Natur* [Man87]. Betrachtet man z.B. einen Baum, so bemerkt man, daß ein Ast mitsamt seinen Zweigen und Blättern nur ein verkleinerter Baum ist. So finden sich in einem Ausschnitt alle Elemente des Ganzen wieder. Auch an dem L-System selbst kann man die Selbstähnlichkeit ablesen. Eine Produktion wird immer auf ein bestimmtes Zeichen angewendet, wenn es Vorgänger der Produktion ist. Dabei macht es keinen Unterschied, ob sich das Zeichen in einer Klammer (Verzweigung) befindet oder nicht. D.h. kleine Teile verändern sich auf die gleiche Art und Weise wie die gesamte Pflanze. Eine ausführliche Beschreibung der Lindenmayer-Systeme enthalten [PrLi90] und [PrHa89].

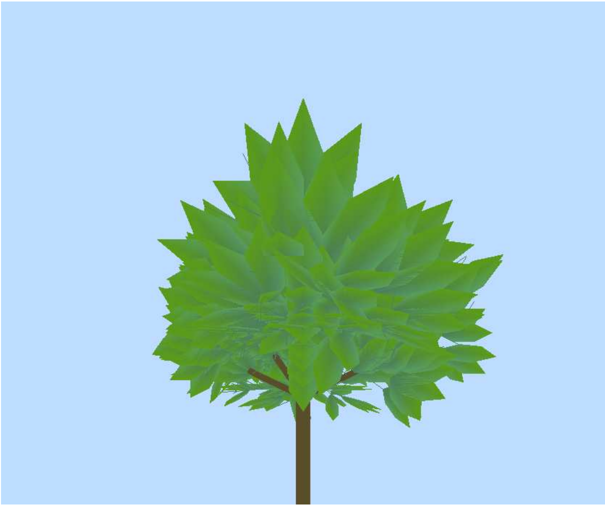

Abbildung 2.15: Fraktaler Baum

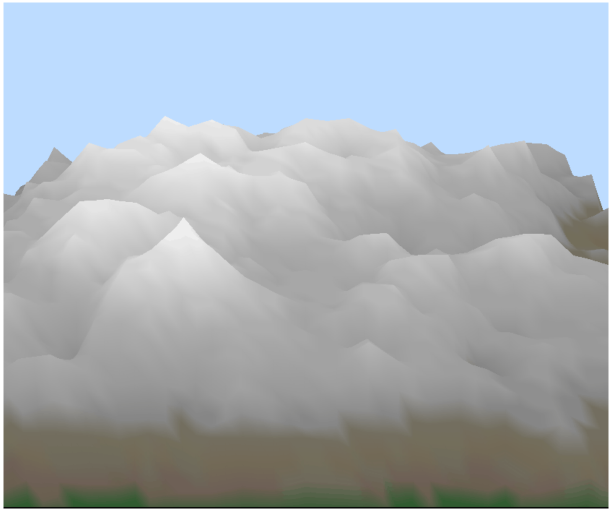

Abbildung 2.16: Fraktaler Berg

Kapitel 3

Darstellung

3.1 Die Viewing-Pipeline

Dreidimensionale Objekte, wie sie z.B. durch die oben beschriebenen Algorithmen erzeugt werden, müssen zur Darstellung auf einen zweidimensionalen Bildschirm projiziert werden. Dabei ist es erforderlich, daß dieselbe Szene von unterschiedlichen Standpunkten betrachtet werden kann. Diese Möglichkeit bietet die *Viewing-Pipeline*. Darunter versteht man eine Folge von Transformationen, die von allen Objekten durchlaufen wird und an deren Ende der sichtbare Teil der Szene in Bildschirmkoordinaten vorliegt.

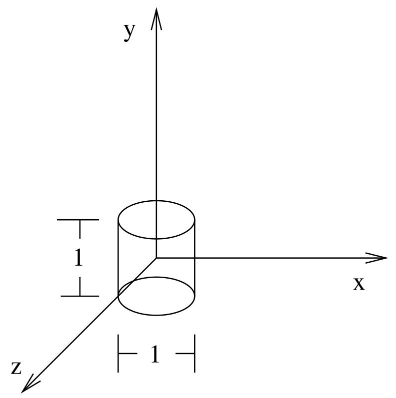

Abbildung 3.1: Zylinder in Modellkoordinaten

In einem ersten Schritt werden alle Objekte in *Modellkoordinaten* definiert. Ein Zylinder hat z.B. in Modellkoordinaten den Mittelpunkt (0, 0, 0), die Höhe 1 und den Durchmesser 1 (siehe

Abbildung 3.1). Das Modellkoordinatensystem (MC) hat nicht nur den Vorteil, daß Objekte recht einfach beschrieben werden können, sondern auch, daß ein Modellobjekt mehrfach für unterschiedliche Szenenobjekte verwendet werden kann. Denn in einem zweiten Schritt werden die Modellobjekte durch Translation, Rotation und Skalierung in ein *Weltkoordinatensystem* (WC) überführt.

Die Objekte, die durch den oben beschriebenen Bergalgorithmus und die Lindenmayer-Systeme entstehen, werden schon zu Beginn in Weltkoordinaten definiert, d.h. die Abbildung des Modellkoordinatensystems in das Weltkoordinatensystem entfällt hier. Im Weltkoordinatensystem wird auch die synthetische Kamera, mit der die Szene betrachtet wird, plaziert.

3.1.1 Das Modell der synthetischen Kamera

Die *synthetische Kamera* ist eine imaginäre Kamera, mit der ein Betrachter, dessen Augenpunkt *Projection Reference Point*, kurz PRP, genannt wird, im Weltkoordinatensystem steht und eine Szene betrachtet. Der fokussierte Punkt in dieser Szene wird als *View Reference Point* oder VRP bezeichnet. Dieser Punkt legt die Position der Bildebene fest, auf die die Szene projiziert wird. Durch die Lage, Position und Aufnahmerichtung der Kamera wird ein neues Koordinatensystem definiert, dessen Ursprung der anvisierte Punkt (VRP) ist (siehe Abbildung 3.2).

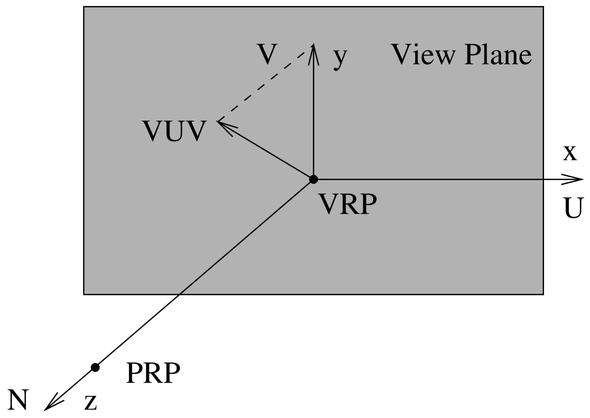

Abbildung 3.2: Definition des UVN-Systems

Als z-Achse (auch N genannt) wird die Normale der Bildebene, die dem Vektor vom VRP zum PRP entspricht, festgelegt. D.h. der Betrachter schaut in Richtung der negativen z-Achse. Zur y-Achse (auch mit V bezeichnet) wird die Projektion des *View Up Vectors* (VUV) in die Bildebene. Der View Up Vector gibt die Orientierung der Kamera im Weltkoordinatensystem an. Er zeigt nach oben. Schließlich erhält man als dritten Vektor in einem rechtshändigen Koordinatensystem die x-Achse (auch U genannt). Das so entstandene Koordinatensystem wird als *View-Reference-Koordinatensystem* (VRC) oder *UVN-System* bezeichnet.

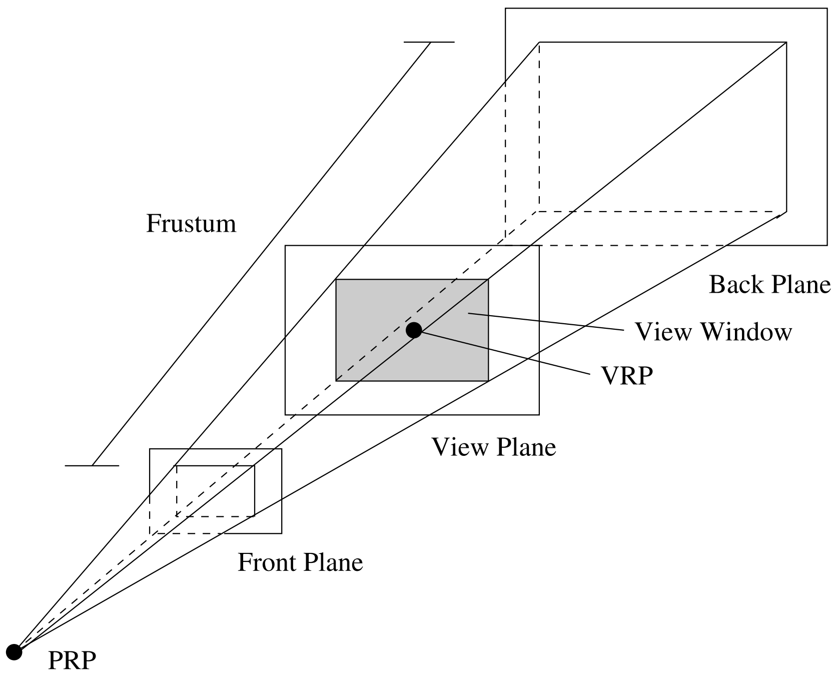

Abbildung 3.3: Frustum gebildet aus Front Plane, Back Plane und View Window

Durch ein Fenster (*View Window*), das in die Bildebene gelegt wird, wird der sichtbare Bereich der Szene zur Seite sowie nach oben und unten abgegrenzt. Damit erreicht das Fenster den Effekt eines Tele- oder Weitwinkelobjektivs für die synthetische Kamera. Sehr nahe und sehr ferne Objekte werden durch zwei Ebenen (*Front Plane* und *Back Plane*) abgeteilt, die vor bzw. hinter der Bildebene liegen (siehe Abbildung 3.3). Nur Objekte bzw. Objektteile, die innerhalb dieses Pyramidenstumpfes (*Frustum*) liegen, werden projiziert, d.h. sind später sichtbar. Daher werden alle außerhalb liegenden Szenenteile „abgeschnitten“. Dies geschieht mittels des sogenannten *Clippings*, welches in Abschnitt 3.1.3 beschrieben wird. Clipping an einem Pyramidenstumpf erweist sich allerdings als recht schwierig. Aus diesem Grund wird das Frustum in einen Würfel (Kanten bei 0 und 1 in allen Dimensionen) transformiert (siehe Abbildung 3.4). Dazu wird zunächst der Pyramidenstumpf in eine (abgeschnittene) regelmäßige Pyramide umgewandelt und diese anschließend durch eine 2-dimensionale Scherung in den Einheitswürfel.

Durch diese Transformationen wird wiederum ein neues Koordinatensystem erzeugt, das *Normalized-Projection-Koordinatensystem* (NPC). Seinen Ursprung bildet der Projection Reference

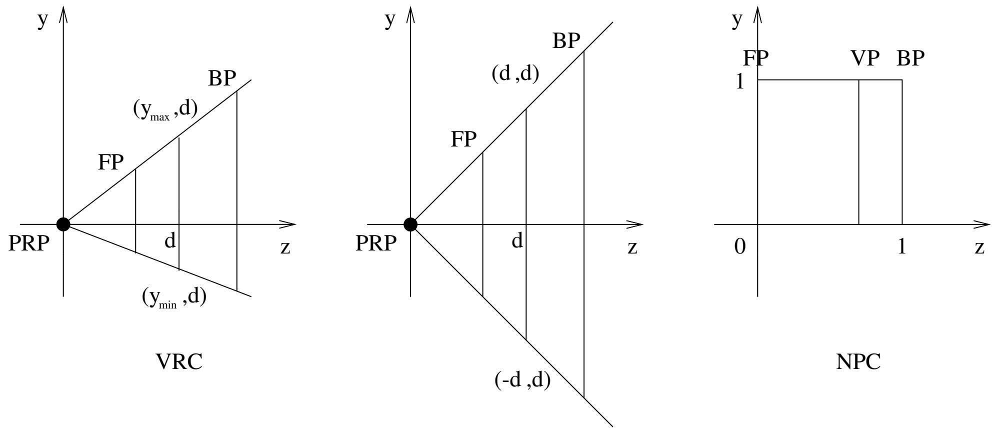

Abbildung 3.4: Transformation des Pyramidenstumpfes in einen Einheitswürfel

Point, und es ist linkshändig. In diesem Koordinatensystem entspricht die Front Plane der Ebene $z = 0$ und die Back Plane der Ebene $z = 1$. Der Betrachterstandpunkt liegt im Unendlichen auf der negativen z-Achse. Um den Inhalt des Würfels auf den Bildschirm zu projizieren, müssen nur noch die x- und y-Werte auf Bildschirmkoordinaten skaliert werden (*Device-Koordinatensystem*, DC). Der z-Wert liefert die Tiefeninformation für das *Z-Buffering*. Eine ausführlichere Darstellung der Viewing-Pipeline einschließlich der für die Berechnungen notwendigen Matrizen enthalten [Fel92] und [Vor94].

3.1.2 Entfernung der Rückflächen

Um den Algorithmus zur Darstellung der Szene zu beschleunigen, ist es notwendig, daß die nicht sichtbaren Flächen der Objekte bei der weiteren Berechnung nicht mehr berücksichtigt werden (siehe Abbildung 3.5). Die Entfernung der Rückseiten (*Back Face Culling* oder *Back Face Removal*) schließt einen Teil dieser Flächen von der Bearbeitung aus.

Bei dieser Methode handelt es sich um einen *Objektraumalgorithmus*, d.h. sie basiert auf der Betrachtung der Objekte (Polyeder) im Weltkoordinatensystem. Jede Fläche dieser Szenenobjekte besitzt eine Normale, die so definiert ist, daß sie nach außen zeigt. Liegt nun der Augenpunkt (Betrachterstandpunkt, PRP) auf der Außenseite der Fläche eines Polyeders, so handelt es sich um eine Vorderfläche, und sie ist vom Betrachter sichtbar (siehe Abbildung 3.6). Ansonsten ist diese Fläche eine Rückfläche und kann ausgeschlossen werden. Sichtbarkeit oder Unsichtbarkeit einer Fläche wird mittels der Ebenengleichung der Ebene, in der die betroffene Fläche liegt, entschieden. Sie setzt sich aus der Normalen der Fläche (A, B, C) und ihrem Abstand

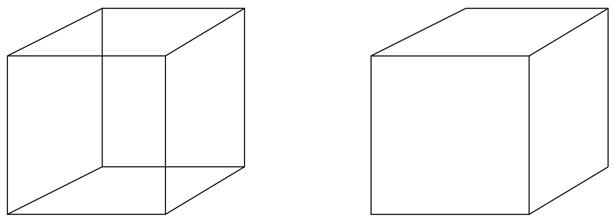

Abbildung 3.5: Würfel mit und ohne Rückflächen

zum Koordinatenursprung (D) zusammen und lautet für alle Punkte (x, y, z), die in der Fläche liegen:

$$Ax + By + Cz + D = 0$$

Setzt man nun den Augenpunkt $P = (x', y', z')$ in die Gleichung ein, so ist die Fläche nicht sichtbar, falls $Ax' + By' + Cz' + D > 0$.

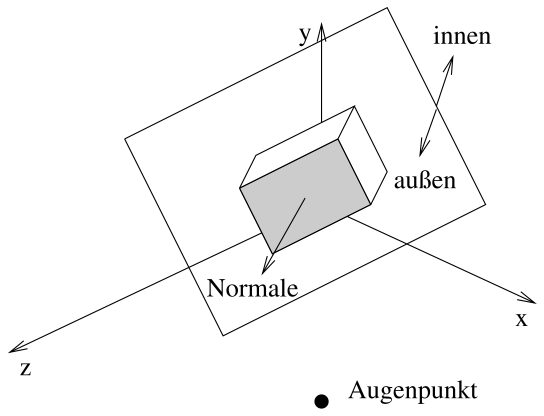

Abbildung 3.6: Die Normale definiert die Außenseite einer Fläche

Dieser Test vereinfacht sich im Normalized-Projection-Koordinatensystem (siehe Abbildung 3.7). Da in diesem Koordinatensystem der Augenpunkt im negativen Unendlichen liegt und die Blickrichtung entlang der positiven z-Achse verläuft, kann anhand der z-Koordinate des Normalenvektors der Fläche (entspricht dem C-Koeffizienten der Ebenengleichung) entschieden werden, ob die Normale vom Betrachter weg zeigt oder zum Betrachter hin. D.h., gilt $C > 0$, so ist

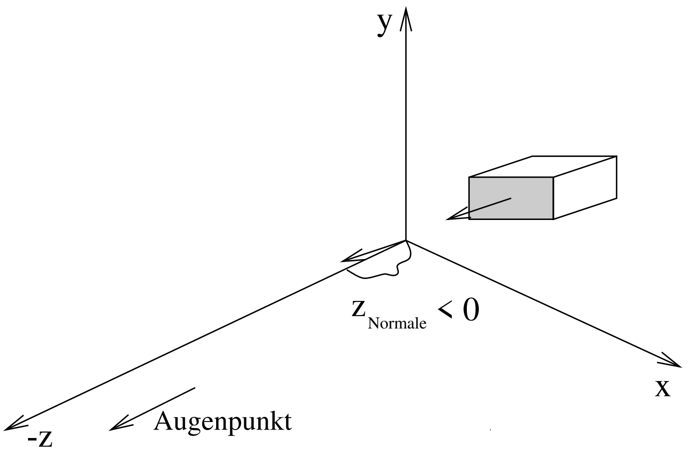

Abbildung 3.7: Für den Sichtbarkeitstest wird die z-Komponente der Normalen im NPC benötigt

die betrachtete Fläche nicht sichtbar. Es ist also ausreichend, nach Berechnung der Objekte in Weltkoordinaten die z-Koordinate der Normalen jeder Fläche ins NPC zu überführen, um zu überprüfen, ob die Fläche eventuell von den weiteren Berechnungen auszuschließen ist.

Durch das Back Face Culling reduziert sich der Zeitaufwand für die Viewing-Pipeline enorm. Bei konvexen Körpern werden alle Rückflächen entfernt. Ansonsten verkleinert sich die Zahl der weiter zu bearbeitenden Flächen auf bis zu 50%.

3.1.3 Clipping

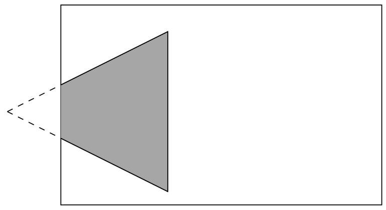

Abbildung 3.8: Clipping eines Dreiecks

Wie in 3.1.1 ausgeführt, werden Objekte bzw. Objektteile, die nicht zum Szenenausschnitt (Frustum) gehören, an den Würfelgrenzen des Einheitswürfels im NPC „abgeschnitten“. Diesen

Vorgang nennt man *Clipping*. Durch das Clippen der Objekte werden unsichtbare Teile von weiteren, unnützen Berechnungen ausgeschlossen (siehe Abbildung 3.8).

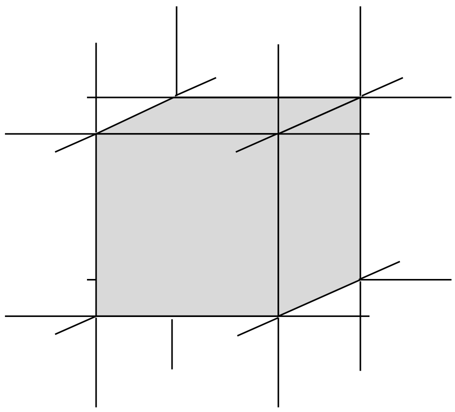

Abbildung 3.9: Einteilung des Raumes in 27 Bereiche anhand des Einheitswürfels im NPC

Das Clipping von Linien wird anhand der dreidimensionalen Erweiterung des Algorithmus von *Cohen & Sutherland* [Fel92] durchgeführt. Dazu teilt man den Raum um den Einheitswürfel im NPC in 27 Bereiche auf (siehe Abbildung 3.9). Jedem Bereich wird ein Code bestehend aus 6 Bits zugewiesen, der sich folgendermaßen zusammensetzt:

1. Bit gesetzt	Bereich links vom Würfel
2. Bit gesetzt	Bereich rechts vom Würfel
3. Bit gesetzt	Bereich unterhalb des Würfels
4. Bit gesetzt	Bereich oberhalb des Würfels
5. Bit gesetzt	Bereich vor dem Würfel
6. Bit gesetzt	Bereich hinter dem Würfel

Für die Endpunkte einer Linie wird nun der jeweilige Bereichscode ermittelt. Gilt

$$Code(P_1) \quad \text{and} \quad Code(P_2) \neq 0,$$

so befindet sich die Linie komplett außerhalb des sichtbaren Ausschnitts und braucht nicht geclippt werden (siehe Abbildung 3.10). Andernfalls wird jeweils für die außerhalb des Würfels liegenden Punkte der Schnittpunkt mit den entsprechenden Bereichsgrenzen des Würfels berechnet, bis beide Endpunkte der Linie innerhalb des Würfels liegen. Der Farbwert des Schnittpunktes wird aus den beiden Farbwerten der Linienendpunkte linear interpoliert. Dabei geht das Verhältnis der Streckenlängen zwischen Schnittpunkt und Endpunkten als Interpolationsgewicht in die Berechnung mit ein.

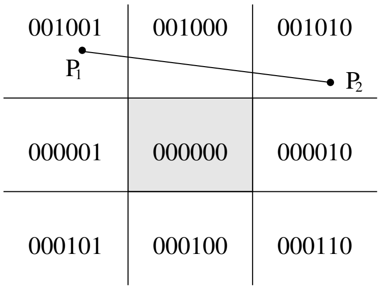

Abbildung 3.10: Außerhalb des sichtbaren Bereichs liegende Linie

Die Methode von Cohen & Sutherland ist für Polygone unbrauchbar, da das Clippen der einzelnen Linien das Polygon in eine Menge von Linienstücken zerfallen läßt, die kein zusammenhängendes Polygon mehr bilden. Statt dessen wird das *Reentrant Polygon Clipping* von *Sutherland & Hodgman* [Fel92] verwendet:

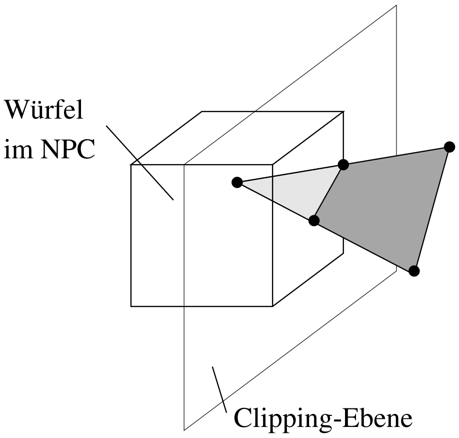

Abbildung 3.11: Polygonclipping am Einheitswürfel im NPC

Für jede Clipping-Ebene (definiert durch die Begrenzungsflächen des Würfels) werden alle Polygonpunkte durchlaufen. Liegt der betrachtete Punkt außerhalb, so wird er vernachlässigt, liegt er dagegen innerhalb, wird er übernommen (siehe Abbildung 3.11). Schneidet nun die Polygonkante dieses Punktes zu seinem Nachfolger in der Punkteliste die Clipping-Ebene, so geht auch der Schnittpunkt als neuer Punkt in das geclippte Polygon ein. Auch hier wird der Farbwert des Schnittpunktes linear aus dem Punkt und seinem Nachfolger interpoliert.

3.1.4 Z-Buffering

Durch das Clipping wurden nicht zum Szenenausschnitt gehörende Objekte bzw. Objektteile entfernt. Die Entfernung der Rückflächen diente dem Aussortieren eines Teils der abgewandten Flächen innerhalb des betrachteten Szenenausschnitts. Um bei der endgültigen Abbildung der Objekte in Normalized-Projection-Koordinaten auf Bildschirmkoordinaten (Projektion) für jedes Pixel des Bildschirms entscheiden zu können, welches der übriggebliebenen Objekte an dieser Stelle letztendlich sichtbar ist, dient das bereits erwähnte *Z-Buffering* (auch *Tiefenpuffer-Algorithmus* genannt) [Fel92]. Es handelt sich hierbei um einen *Bildraumalgorithmus*, da direkt mit den Bildschirmkoordinaten gearbeitet wird.

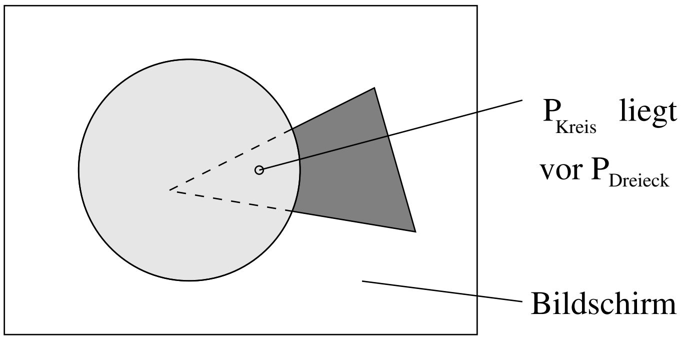

Abbildung 3.12: Z-Buffering entscheidet über die Sichtbarkeit von Punkten

Das Z-Buffering benötigt zwei Felder in der Größe des Bildschirms (Auflösung in Pixeln). Das erste Feld ist der *Bildspeicher*, in dem für jedes Pixel der Farbwert gespeichert ist, der auf dem Bildschirm erscheint. Im zweiten Feld wird für jedes Pixel eine Tiefeninformation gehalten, die den z-Wert des Punktes im NPC enthält, dessen Farbwert im Bildspeicher steht. Zu Beginn wird dieser sogenannte *Z-Buffer* oder *Tiefenpuffer* mit einem möglichst großen Wert initialisiert. Nun wird jede Fläche (Linie) der Szene betrachtet und für jeden Punkt dieser Fläche (Linie) sein Farbwert und seine Tiefe (z-Wert im NPC) berechnet. Ist nun die Tiefe des entsprechenden Pixels kleiner als der im Z-Buffer eingetragene Wert, so heißt dies, daß dieser Punkt vor dem momentan gespeicherten liegt und daher den gespeicherten Punkt verdeckt (siehe Abbildung 3.12). Somit wird der Farbwert des neu berechneten Punktes an die entsprechende Stelle im Bildspeicher geschrieben und die Tiefeninformation an dieselbe Stelle im Z-Buffer.

Wenn alle Objekte der Szene durch diesen Algorithmus abgearbeitet wurden, enthält der Bildspeicher das vom Betrachter sichtbare Bild, welches auf den Bildschirm ausgegeben werden kann.

3.2 Beleuchtung

Zur realistisch wirkenden Darstellung der Szene werden die Objekte auf unterschiedliche Art beleuchtet. Durch den Lichteinfall verändert sich die Intensität des Farbwertes eines Punktes. Der endgültige Farbwert ist also abhängig vom Standpunkt der Lichtquelle, von der Lage der zugehörigen Fläche zur Lichtquelle und von der Materialeigenschaft des gesamten Objektes. Außerdem wird durch Reflexion an den Objekten das Licht im Raum verteilt.

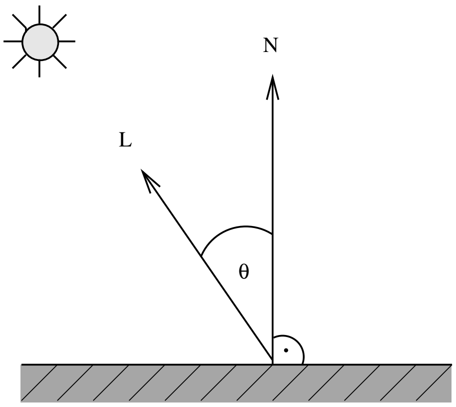

Abbildung 3.13: Das Kosinusgesetz von Lambert

Die Lichtintensität jedes Punktes läßt sich aus der Hintergrundbeleuchtung und der direkten Beleuchtung berechnen. Die *Hintergrundbeleuchtung* geht mit dem konstanten Faktor I_H in das Beleuchtungsmodell ein, da sie im gesamten Raum annähernd gleich ist. Sie wird mit dem *ambienten Reflexionskoeffizienten* k_a $(0 \leq k_a \leq 1)$ multipliziert, durch den das Zurückstrahlen der Hintergrundbeleuchtung festgelegt wird. Für die zu berechnende Intensität gilt also:

$$I = I_H \cdot k_a.$$

Zur Berechnung der Intensität des *direkt einfallenden Lichts* benutzt man das *Kosinusgesetz von Lambert* (siehe Abbildung 3.13). Danach ist die Lichtintensität in einem Punkt einer Fläche proportional zum Kosinus des Winkels θ zwischen der Flächennormalen N und dem Vektor L

zur Lichtquelle. Das Licht trifft den Punkt allerdings nur, falls der Einfallswinkel θ kleiner 90° und größer 0° ist. Daraus ergibt sich für das Beleuchtungsmodell:

$$I = I_{L_q} \cdot \cos(\theta) \cdot k_d + I_H \cdot k_a, \qquad 0 \leq \theta \leq \frac{\pi}{2}.$$

k_d bezeichnet den *diffusen Reflexionskoeffizienten*, der das Zurückstrahlen des Materials angibt, und I_{L_q} die Intensität der Lichtquelle.

Bei diesem Modell wird allerdings die Entfernung des Objektes bzw. der Fläche zur Lichtquelle nicht berücksichtigt. Die Erfahrung zeigt jedoch, daß die Helligkeit eines Körpers mit der Entfernung abnimmt. Daher wird die Intensität des direkt einfallenden Lichtes mit dem Faktor $\frac{1}{d+d_0}$ multipliziert. Darin bezeichnet d die Entfernung zur Lichtquelle und d_0 eine Konstante zur Vermeidung der Division durch Null oder einer Wertebereichsüberschreitung. Die endgültige Formel lautet also:

$$I = \frac{I_{L_q} \cdot \cos(\theta) \cdot k_d}{d + d_0} + I_H \cdot k_a.$$

Mit der daraus berechneten Intensität I wird jeder Anteil des RGB-Farbwertes (Rot, Grün, Blau) multipliziert, um die endgültige Farbe des Punktes zu erhalten.

3.3 Shading

Zur Darstellung der Objektflächen auf dem Bildschirm müssen die Flächen „ausgemalt“, d.h. mit Farbe gefüllt werden. Dazu müssen für jeden Punkt im „Innern“ der Fläche seine Koordinaten im Device-Koordinatensystem und sein Farbwert berechnet werden. Diese Berechnung geschieht anhand der Eckpunkte, durch die die Fläche im Device-Koordinatensystem definiert ist. Eine Methode, Flächen mit Farbwerten zu füllen, ist das in 3.3.1 beschriebene *Gouraud-Shading*. Zur Vereinfachung des Füllalgorithmus werden zuvor alle Flächen *triangulisiert*, d.h. in Dreiecke zerlegt. Diese haben den Vorteil, daß sie eben und konvex sind und daher die Koordinaten ihrer Punkte recht einfach berechnet werden können.

3.3.1 Gouraud-Shading

Beim *Gouraud-Shading* wird der Farbwert jedes Punktes der Fläche durch *doppelte Interpolation* der Farbwerte der Eckpunkte berechnet (siehe Abbildung 3.14). Zu Beginn wird für jeden Eckpunkt seine Intensität mit dem in 3.2 beschriebenen Beleuchtungsmodell ermittelt und anhand dieser die endgültige Farbe des Punktes berechnet. Dazu wird die Flächennormale benötigt. Bei Flächen, die eine gekrümmte Oberfläche annähern sollen (z.B. einen Zylinder), wird für jeden Eckpunkt die Normale auf die gekrümmte Fläche benutzt. Ansonsten wird die Flächennormale des Dreiecks aus den Eckpunkten berechnet (siehe Abbildung 3.15).

Nachdem die tatsächlichen Farbwerte der Eckpunkte festgestellt wurden, werden die Farbwerte der übrigen Punkte des Dreiecks ermittelt, indem eine horizontale *Scanline* vom obersten Punkt des Dreiecks zum untersten geführt wird (siehe Abbildung 3.14).

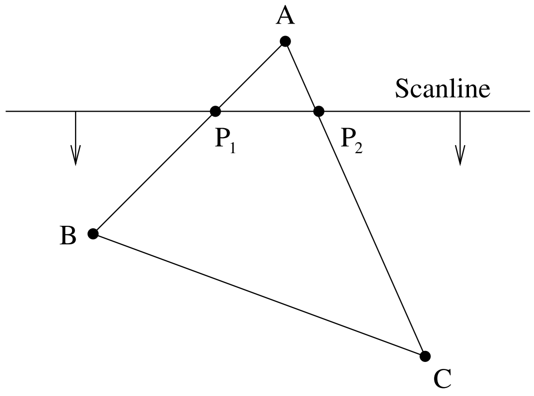

Abbildung 3.14: Doppelte Interpolation jedes Farbwertes mit Hilfe horizontaler Scanline

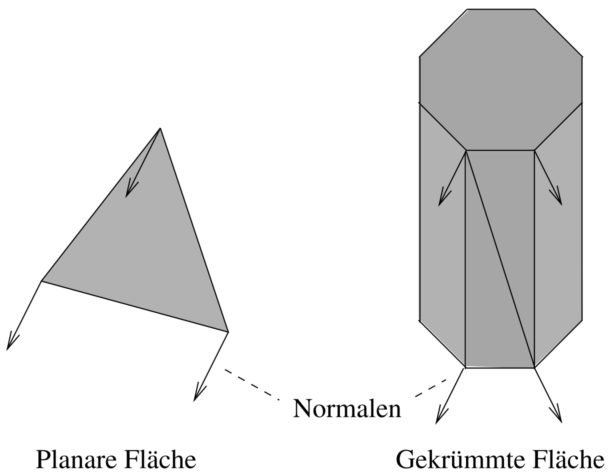

Abbildung 3.15: Festlegung der Normalen einer planaren und einer gekrümmten Fläche

In einem ersten Schritt werden dabei die Schnittpunkte der Scanline mit den Kanten des Dreiecks (Punkte auf den Verbindungslinien der Eckpunkte) berechnet. Ihre Farbwerte werden anhand der Farbwerte der beiden jeweiligen Eckpunkte linear interpoliert. In einem zweiten Schritt werden für alle Punkte auf der Scanline, die im „Innern" des Dreiecks liegen, ihre Koordinaten und Farbwerte durch lineare Interpolation der Kantenpunkte ermittelt. Nach der Berechnung eines Punktes wird mit Hilfe des Z-Buffering festgestellt, ob er auf dem Bildschirm erscheint.

Das Resultat ist eine kontinuierliche Farbabstufung auf dem gesamten Objekt, durch die die Dreiecksübergänge nicht zu erkennen sind. Ein Beispiel wird in Abbildung 3.16 dargestellt.

3.3.2 Wire-Frame-Darstellung

Bei der *Wire-Frame-Darstellung* werden die Flächen der Objekte nicht komplett mit Farbe gefüllt. Statt dessen werden nur die Verbindungslinien zwischen den Dreiecken gezeichnet. Es findet also eine einfache Interpolation zwischen den Eckpunkten des Dreiecks statt. Dadurch entsteht eine Netzstruktur, die den Aufbau der Objekte besser erkennen läßt (siehe Abbildung 3.17). Außerdem wird die Rechenzeit erheblich verkürzt.

3.4 Postprocessing

Bei der Darstellung des Bildes, das nach dem Z-Buffering im Bildspeicher abgespeichert ist, kann es Probleme geben, da die meisten Bildschirme eine Colormap mit nur 256 Farbeinträgen besitzen, das Bild aber häufig mehr als 1000 Farben enthält. Daher muß die Anzahl der Farben vor der Darstellung des Bildes reduziert werden. Dazu wird hier der Bildspeicher im Format eines *Portable Pixmap Files* (PPM-File) in einer Datei abgespeichert und auf diese Datei das Tool *ppmquant* [1] angewendet. *ppmquant* berechnet ein Portable Pixmap File mit höchstens *ncolors* Farben, die den vorher verwendeten am nächsten liegen. Die Berechnung geschieht anhand von Heckberts *median cut* [Hec82].

Als graphische Benutzeroberfläche wird hier das *X Window System* (kurz *X* genannt) verwendet. Um die PPM-Datei auf dem Bildschirm in einem X-Fenster darstellen zu können, wird sie daher zuvor durch die Utility *pnmtoxwd* [1] in ein *XWD*-File[2] (*X Window Dump*) konvertiert. Diese bringt eine „Portable Any Map“ (also auch eine Portable Pixmap) in das Format eines X Window Dumps. Das XWD-File wird mit der Utility *xwud* (X Window Undump) in einem X-Fenster auf dem Bildschirm dargestellt.

[1] ©Jef Poskanzer, 1989, 1991

[2] Ein XWD-File enthält den Inhalt eines X-Fensters in einem speziellen, eng an die Strukturen von X angelehnten Format.

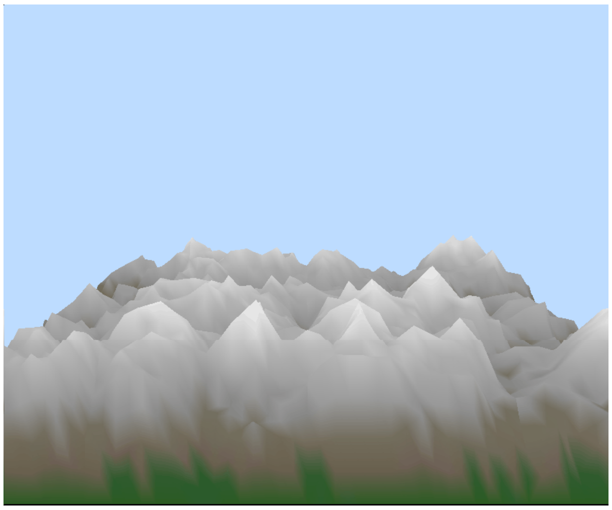

Abbildung 3.16: Gouraud-geshadeter Berg

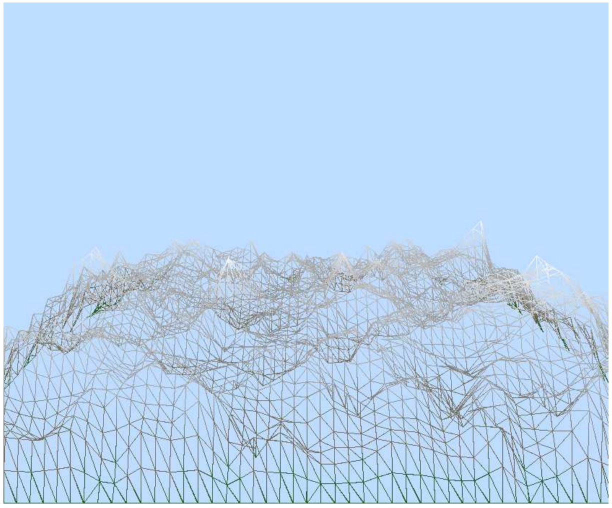

Abbildung 3.17: Berg als Wire-Frame

Kapitel 4

Parallelisierung

Computergraphische Algorithmen benötigen aufgrund ihrer Komplexität lange Laufzeiten. Um die Rechenzeit des hier entwickelten Programms zur Darstellung fraktaler Landschaften zu verkürzen, wird die sequentielle Version *parallelisiert*. Diese *Parallelisierung* soll im folgenden vorgestellt werden. Zuvor werden die Konzepte paralleler Systeme und die hier verwendete Hard- und Software erläutert.

4.1 Massiv parallele Systeme

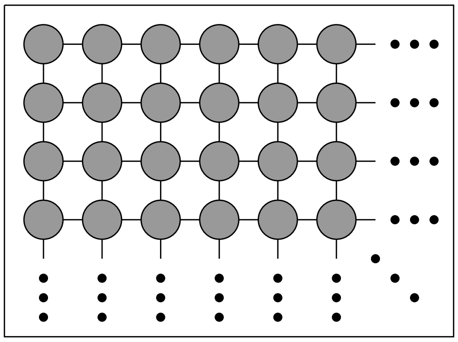

Abbildung 4.1: Massiv paralleles System: 1 Parallelrechner mit vielen Prozessoren

Sowohl im Hardware- als auch im Betriebssystembereich gibt es unterschiedliche Ansätze der Parallelisierung. So besteht eine Möglichkeit darin, vorhandene Unix-Workstations gemeinsam an einem Problem arbeiten zu lassen. Ein solches paralleles System besteht meist nur aus einigen Prozessoren. Oft sollen jedoch Hunderte oder mehr Prozessoren eingesetzt werden. Dann spricht man von einem *massiv parallelen System* (siehe Abbildung 4.1). Die Anzahl der Prozessoren ist bei diesen Systemen beliebig skalierbar. Im vorliegenden Fall wird ein paralleles System mit bis zu 64 Prozessoren verwendet. Bevor jedoch auf die hier verwendete Hard- und Software eingegangen wird, sollen zunächst allgemeine Konzepte paralleler Systeme vorgestellt werden.

4.1.1 Konzepte

Unter einem *massiv parallelen System* versteht man einen Rechner, der aus Hunderten, evtl. sogar mehr, eng geschalteter Prozessoren besteht. Jeder Prozessor stammt aus der Workstation-Klasse und wurde mit spezieller Hardware für Floating-Point-Arithmetik und Interprozessorkommunikation versehen. Einige solcher Systeme arbeiten auf einem verteilten Speicher (*Distributed Memory*). Das bedeutet, jeder Prozessor besitzt seinen eigenen, lokalen Speicher, auf den auch nur er Zugriff hat. Das Gegenteil wäre ein globaler Speicher (*Shared Memory*), bei dem jeder Prozessor auf den gesamten Speicherbereich zugreifen kann. Um Daten austauschen zu können, sind die Prozessoren untereinander verbunden.

Ebenso wie der Speicher ist bei den massiv parallelen Systemen auch die Programmausführung lokal geregelt. D.h., auf jedem Prozessor läuft ein lokales Programm, das die ihm zur Verfügung stehenden Daten verarbeitet. Man bezeichnet dieses Konzept als *MIMD-Architektur* (Multiple Instruction Multiple Data).

4.1.2 Parallelrechner

Die hier verwendeten Architekturen basieren auf dem Konzept der MIMD-Architektur mit verteiltem Speicher. Es handelt sich dabei um einen Multicluster von 64 *Transputern* des Typs T800 und einen *PowerXplorer* mit 4 Motorola MPC601-Prozessoren bzw. um einen GC/PP mit 96 Knoten à 2 MPC601-Prozessoren. Alle Systeme wurden von der Firma Parsytec entwickelt.

Jeder der 64 Transputer besteht aus einem 32-Bit RISC-Prozessor, erweitert um eine 64-Bit Floating Point Unit und 4 KByte On-Chip-RAM. Die Speicherkapazität beträgt 1 oder 2 MByte und die Rechenleistung 1,5 MFLOPS. Außerdem besitzt jeder Prozessor vier bidirektionale Kanäle (sogenannte *Links*), durch die er mit anderen Transputern verbunden werden kann, um Datenaustausch (*Kommunikation*) zu betreiben. Die Kommunikation kann parallel zu den übrigen Berechnungen laufen, da die Links unabhängig von der CPU gesteuert werden.

Die Transputer können zu einem Netzwerk zusammengebaut werden, indem eine beliebige Anzahl von ihnen mit Hilfe ihrer Links verknüpft wird. Man spricht in diesem Zusammenhang von dem Aufbau einer *Topologie*. Dabei können die Prozessoren beliebig verschaltet werden. Allerdings sind pro Transputer nur vier Verbindungen zu anderen Prozessoren möglich.

Der PowerXplorer enthält 4 MPC601-Prozessoren von Motorola. Sie sind mit 80 MHz getaktet und haben eine Speicherkapazität von 32 MByte. Jeder Knoten des Rechners ist außerdem mit einem T800-Transputer (1 MByte Speicher) ausgestattet (siehe Abbildung 4.2). Im GC/PP

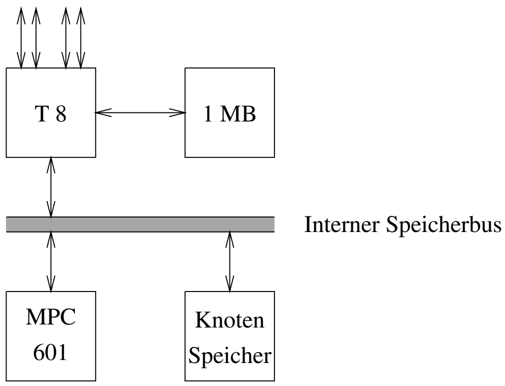

Abbildung 4.2: Aufbau eines Xplorer-Knotens

sind 96 Knoten mit je 2 MPC601-Prozessoren und 4 T800-Transputern vorhanden. Die Transputer besitzen jeweils vier Links und sind für die Kommunikation zuständig. Allerdings sind die Prozessoren des PowerXplorers bzw. des GC/PPs auf der Hardwareseite fest zu einem Gitter verschaltet, d.h. andere Topologien können nur virtuell auf der Softwareseite erzeugt werden.

4.1.3 Parix

Das hier verwendete Betriebssystem für die Parallelrechner ist *Parix* (PARallel extensions to unIX), das ebenfalls von der Firma Parsytec entwickelt wurde. Es handelt sich dabei um ein *Message-Passing-System* für Transputernetzwerke, d.h. es wurde für den Datenaustausch zwischen Transputern mit lokalem Speicher erstellt. Allerdings ist ein unter Parix entwickeltes paralleles Programm nicht nur auf einem Transputernetz, sondern auch auf dem PowerXplorer bzw. dem GC/PP lauffähig. Für diese Architektur existiert ein eigener Compiler.

Die parallelen Programme werden zunächst in einer herkömmlichen Programmiersprache (in diesem Fall ANSI-C) auf einer Workstation mit Unix-Betriebssystem implementiert und crosscompiliert. Anschließend werden die erzeugten Binaries von einem Serverprogramm, welches auf einem Hostrechner läuft, auf die Transputer bzw. den PowerXplorer geladen und gestartet. Als *Host* wird derjenige Rechner bezeichnet, der eine Verbindung zum Prozessornetz besitzt. Nach dem Start des Binaries läuft auf jedem Prozessor dasselbe Programm. Über das Serverprogramm können die Prozessoren Kontakt mit dem Host aufnehmen (z.B. für die Ein- und Ausgabe).

Untereinander sind die Transputer unter Parix zu einem zweidimensionalen Gitter verschaltet. Jeder Prozessor ermittelt seine Aufgabe anhand seiner Position in diesem Netz.

Um die parallele Programmierung zu ermöglichen, stellt Parix außerdem spezielle Bibliotheksfunktionen zur Verfügung, die in die sequentiellen Programme eingebettet werden. So kann mit

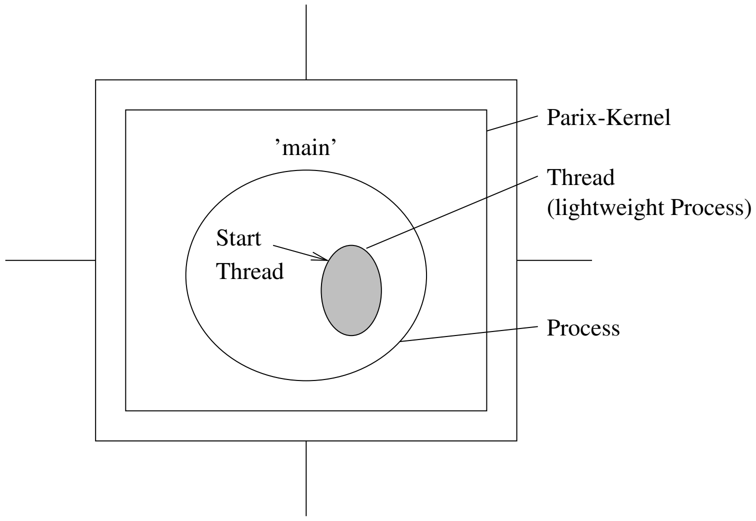

Abbildung 4.3: Ein Thread unter Parix

`Execute` ein anderes Programm nachgeladen und ausgeführt oder nachträglich ein sogenannter *Thread* oder *lightweight process* (arbeitet im selben Kontext wie sein Vaterprozeß, siehe Abbildung 4.3) gestartet werden. Um die Kommunikation zwischen beliebigen Prozessoren zu ermöglichen, können *virtuelle Links* erzeugt werden. Das bedeutet, daß zwei Prozessoren, die nicht unbedingt physikalische Nachbarn sein müssen, softwaremäßig von Parix verbunden werden. Über diese Verbindungen können Byte-Vektoren sowohl asynchron als auch synchron verschickt werden. Zum Verbindungsaufbau stellt Parix Kommunikationsroutinen zur Verfügung. Durch die Threads wird sogar eine parallele Kommunikation ermöglicht. Aus mehreren virtuellen Links kann eine *virtuelle Topologie* erzeugt werden. Verschiedene Topologien stellt Parix bereits zur Verfügung. Weitere Informationen zu Parix enthalten [PaRe92] und [PaSo92].

4.2 Prozessoraufteilung

Die Parallelisierungsidee der Algorithmen zur Bilderzeugung besteht darin, daß jeweils eine Gruppe von Prozessoren die Bergdaten und eine Gruppe die Pflanzendaten berechnet und beide ihre Daten an eine dritte Gruppe zur Weiterverarbeitung durch den Darstellungsalgorithmus schicken. Auf diese Weise können gleichzeitig neue Objekte berechnet und bereits berechnete Objekte weiterverarbeitet werden.

Zu Beginn wird vom Host auf allen Prozessoren des Parallelrechners dasselbe Hauptprogramm gestartet. Dabei wird zum Datenaustausch eine Pipe zwischen den Prozessoren und dem Host

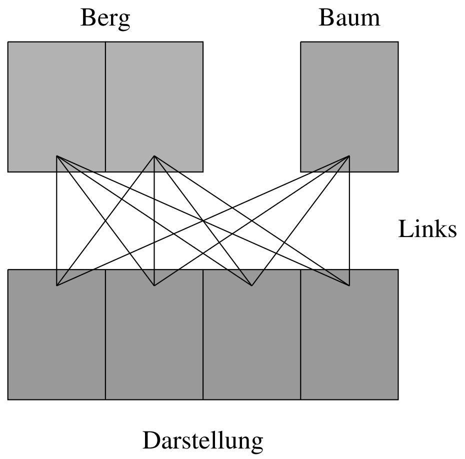

Abbildung 4.4: Globale Topologie: Jeder Berg- und jeder Baumprozessor wird mit jedem Darstellungsprozessor verbunden

aufgebaut. Über diese Pipe schickt der Host einem ausgezeichneten Prozessor die Problemdaten. Nachdem die Prozessoren zu einer Gittertopologie verbunden wurden, verschickt dieser Prozessor die Problemdaten im Gitter an die übrigen Prozessoren. Anhand ihrer Prozessor-IDs bestimmen die Prozessoren nun, ob sie im folgenden den Berg-, den Pflanzen- oder den Darstellungsalgorithmus bearbeiten müssen, und bauen zur späteren Kommunikation eine neue virtuelle Topologie auf. Darin wird jeder Berg- und jeder Baumprozessor mit jedem Darstellungsprozessor verbunden (siehe Abbildung 4.4). Mit dem Parix-Kommando `Execute` werden die entsprechenden Binaries auf den jeweiligen Prozessoren nachgeladen und gestartet. Im Argumentvektor des Programmaufrufs werden die oben beschriebene globale Topologie (Topologie-ID und Control-Blocks der Links) und die für den Algorithmus notwendigen Problemdaten übergeben.

Das parallele Programm läuft auf Transputernetzen der Größe 8×8, 8×4 und 4×4, wobei die beiden kleineren Netze aus Speicherplatzgründen nur aus 2MB-Prozessoren bestehen dürfen. Außerdem ist es auf dem PowerXplorer bzw. dem GC/PP mit 2×2 und mehr Prozessoren lauffähig. Bei der Aufteilung der Prozessoren in drei Gruppen (Berggruppe, Pflanzengruppe, Darstellungsgruppe) müssen $\frac{2}{3}$ der Prozessoren für die Darstellung bereitgestellt werden, da diese einen hohen Speicherbedarf (Z-Buffer, Frame-Buffer) und eine hohe Rechenzeit (Shading) besitzt. Für die Berggruppe wird eine quadratische Anzahl von Prozessoren benötigt. Alle weiteren Prozessoren werden der Pflanzengruppe zugeordnet (siehe Abbildung 4.5).

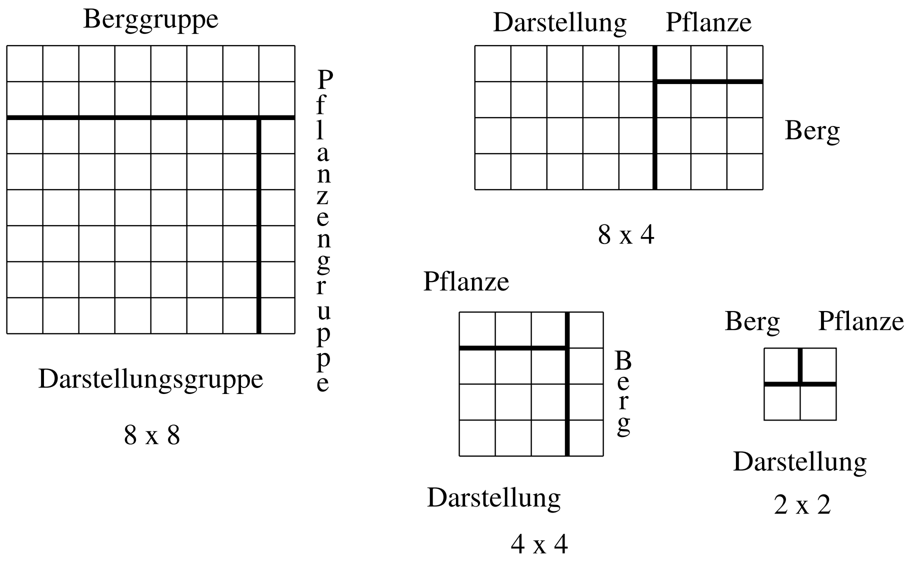

Abbildung 4.5: Aufteilung der drei Programmgruppen auf die Topologien

4.3 Parallelisierung des Bergalgorithmus

Bei der Parallelisierung des Bergalgorithmus wird die Berechnung des Berges anhand eines quadratischen Gitters auf mehrere Prozessoren verteilt (siehe Abbildung 4.6).

Jeder Prozessor berechnet mit Hilfe des sequentiellen Algorithmus einen Teilberg. Aus diesen Teilbergen werden hinterher die Daten eines Gesamtberges ermittelt. Man spricht in diesem Zusammenhang von einer *Objektraumaufteilung*. Zum Datenaustausch werden die Prozessoren untereinander durch den Parix-Aufruf `Make2DGrid` virtuell zu einem 2-dimensionalen Gitter verbunden. Über die Links dieser Gittertopologie werden die Grenzdaten, die von zwei benachbarten Prozessoren gehalten werden, nach jeder Iteration in Vektorform in eine Richtung verschickt, linear interpoliert, und zurückgeschickt. Nur die Randdaten des Gesamtberges werden — wie im sequentiellen Fall — auf Null gesetzt.

Während der Berechnung des Höhenfeldes merkt sich jeder Prozessor seine lokale Bergspitze. Um die Farbverteilung durchführen zu können, muß nach der Berechnung die Bergspitze des gesamten Berges bestimmt werden. Dazu werden in einem ersten Schritt die lokalen Bergspitzen im Gitter verschickt und ihr Maximum bestimmt. In einem zweiten Schritt wird anschließend das berechnete Maximum im Gitter verteilt (siehe Abbildung 4.7). Anhand der Position im Prozessorgitter, der Position im Höhenfeld und der Höhe selbst können nun für jeden Punkt seine 3D-Koordinaten und seine Farbe ermittelt werden. Aus jeweils drei dieser Punkte wird

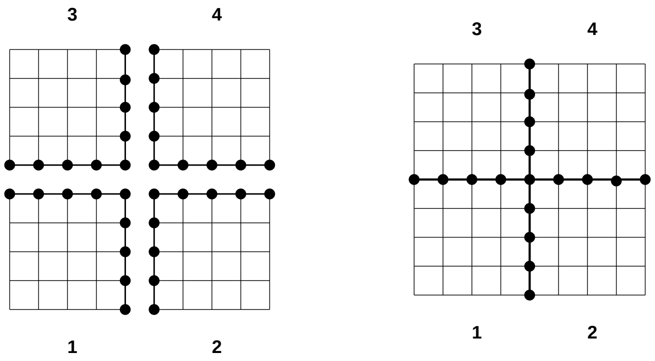

Abbildung 4.6: Aufteilung der Bergberechnung auf die Prozessoren

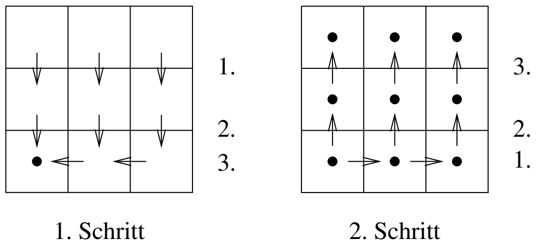

Abbildung 4.7: Ermitteln der globalen Bergspitze bei 9 Bergprozessoren

wiederum ein Dreieck gebildet.

Um über die Weiterverarbeitung der Dreiecke entscheiden zu können, erhalten die Bergprozessoren zu Beginn von den Darstellungsprozessoren die Matrizen zur Umrechnung des Weltkoordinatensystems in das Normalized-Projection-Koordinatensystem und zur Transformation des Normalized-Projection-Koordinatensystems in das Device-Koordinatensystem. Für jedes Dreieck wird nach seiner Berechnung mit Hilfe des Back Face Cullings entschieden, ob es sichtbar ist. Falls es sich um eine Rückfläche handelt, wird es von weiteren Berechnungen ausgeschlossen. Anschließend werden die Dreieckspunkte ins NPC überführt und alle nicht sichtbaren Szenenteile durch das Clipping entfernt. Die geclippten Dreiecke werden ins DC transformiert und zur Weiterverarbeitung (Shading etc.) an die Darstellungsprozessoren verschickt.

4.4 Parallelisierung des Pflanzenalgorithmus

Bei der parallelen Erstellung der Pflanzendaten wird nicht der Algorithmus an sich parallelisiert, sondern der sequentielle Algorithmus läuft parallel auf mehreren Prozessoren ab. Dazu werden die verschiedenen L-Systeme gleichmäßig auf die Prozessoren verteilt und nacheinander abgearbeitet. Da bei der gleichzeitigen Durchführung des sequentiellen Algorithmus keine Kommunikation zwischen den Pflanzenprozessoren notwendig ist, bestehen zwischen ihnen auch keine Links.

Ebenso wie die Bergprozessoren (siehe Abschnitt 4.3) erhalten die Pflanzenprozessoren zu Beginn zwei Transformationsmatrizen (WC_NPC und NPC_DC) über die globale Topologie. Da die L-Systeme als graphische Objekte Dreiecke und Linien liefern, wird das Back Face Culling und das Clipping für beide Objektarten durchgeführt. Nach der Berechnung der Device-Koordinaten werden die Objekte an die Darstellungsprozessoren verschickt.

4.5 Parallelisierung des Darstellungsalgorithmus

Zur Parallelisierung des Darstellungsalgorithmus wird eine *Bildraumaufteilung* durchgeführt. D.h., jeder Prozessor berechnet einen rechteckigen Ausschnitt des Gesamtbildes, für den er auch den Z-Buffer und den Frame-Buffer hält (siehe Abbildung 4.8).

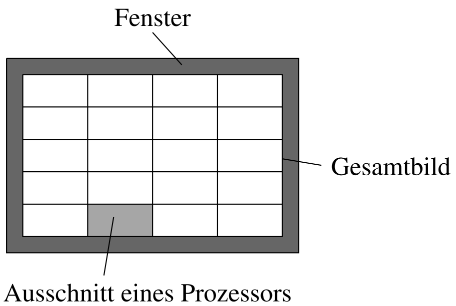

Abbildung 4.8: Aufteilung des Bildraumes auf die Darstellungsprozessoren

Die Daten der zu verarbeitenden Objekte erhalten die Darstellungsprozessoren von den beiden anderen Prozessorgruppen über die globale Topologie. Untereinander müssen die Darstellungsprozessoren nicht kommunizieren und bauen daher auch keine lokale Topologie auf. Bevor die Prozessoren allerdings Daten von den anderen Prozessorgruppen zur weiteren Verarbeitung empfangen, berechnen sie die Transformationsmatrizen WC_NPC und NPC_DC und schicken diese an die Prozessoren der Berg- und der Pflanzengruppe. Anschließend „horcht" jeder Prozessor an seinen Links zu diesen Prozessorgruppen, um festzustellen, ob ihm ein Prozessor Daten schicken möchte. Ist dies der Fall, so empfängt er die entsprechenden Linien- oder Dreiecksdaten und verarbeitet diese.

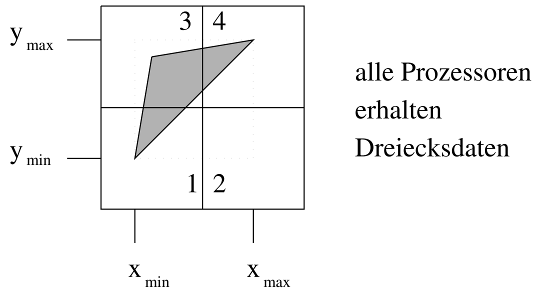

Abbildung 4.9: Ermittlung der Zielprozessoren vor dem Verschicken der Daten

Ein Prozessor erhält die Daten eines Objektes genau dann, wenn das Quadrat, das aus minimalen und maximalen x- und y-Werten des Objektes gebildet wird, den Bildausschnitt des Prozessors schneidet (siehe Abbildung 4.9). Dadurch bekommen die Prozessoren zwar zu viele Daten geschickt, aber genauere Berechnungen wären im Verhältnis zum Kommunikationsaufwand ineffizient. Zur Beendigung der Kommunikation wird ein Stop-Zeichen von jedem Berg- bzw. Pflanzenprozessor an jeden Darstellungsprozessor geschickt. Anschließend wird der Frame-Buffer jedes Prozessors im Portable-Pixmap-Format in eine Datei geschrieben. Vor dem Postprocessing (siehe Abschnitt 3.4) werden diese Dateien zu einer Portable Pixmap zusammengebaut.

4.6 Laufzeiten

Um beurteilen zu können, ob sich die Parallelisierung des Algorithmus und damit der Einsatz zusätzlicher Prozessoren lohnt, wurden auf den beiden Parallelrechnerarchitekturen die Laufzeiten des Programms für unterschiedliche Szenen gemessen. Die erste Szene enthält sechs Bäume und wird geshadet. Die zweite besteht ebenfalls aus sechs Bäumen, allerdings wird für sie eine Wire-Frame-Darstellung berechnet. Für die dritte und vierte Szene wird jeweils nur ein Baum berechnet, wobei bei der dritten Szene wieder das Shading durchgeführt wird und bei der vierten lediglich eine Wire-Frame-Darstellung. Bei allen vier Szenen wird dieselbe Bergdatei mit 3 Iterationen verwendet.

In Abbildung 4.10 sind die Speed-Ups (sequentielle Laufzeit / parallele Laufzeit) für den PowerXplorer bei Berechnung der ersten Szene dargestellt. Nachdem zu Beginn ein Anstieg zu erkennen ist, fällt die Speed-Up-Kurve bei Einsatz weiterer Prozessoren leider ab. Zum einen erklärt sich

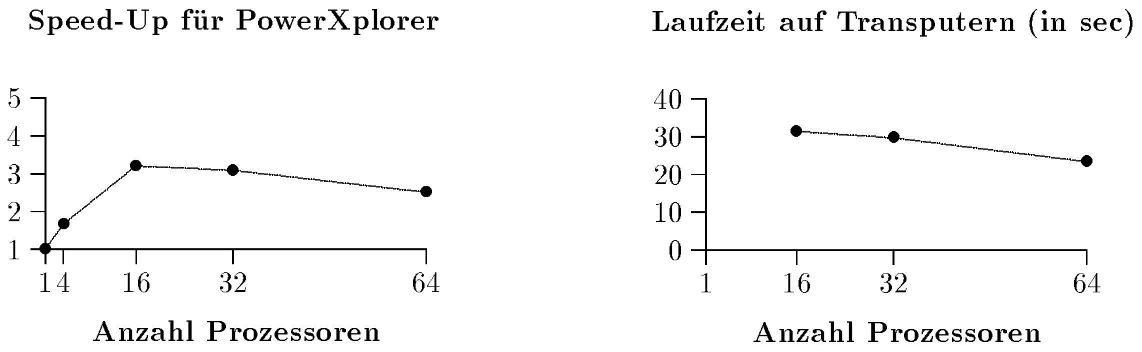

Abbildung 4.10: Speed-Ups und Laufzeiten der Parallelrechner

dies durch den Mehraufwand für die Kommunikation bei größerer Prozessorzahl. Zum anderen muß man bedenken, daß sich durch die Art der Parallelisierung des Bergalgorithmus die Datenmenge bei Erhöhung der Prozessorzahl ebenfalls erhöht. So fallen bei 16 Prozessoren viermal soviele Bergdaten an, wie bei 4 Prozessoren, da auf jedem Bergprozessor ein Teilberg von der Größe des sequentiell berechneten Berges erstellt wird. Durch diese Vergrößerung der Datenmenge erhöht sich nicht nur die Rechenzeit für Berg- und Darstellungsprozessoren, sondern auch der Overhead für die Kommunikation wird größer. Bei gleicher Datenmenge werden daher bessere Speed-Ups erzielt.

Szene Prozessoren	1	4	16	32	64
6 Bäume, Gouraud	12.41	7.42	3.86	4.01	4.95
6 Bäume, Wire-Frame	10.46	6.92	2.74	4.68	4.86
1 Baum, Gouraud	4.40	1.19	1.37	3.93	4.43
1 Baum, Wire-Frame	3.57	1.21	1.46	2.28	4.07

Abbildung 4.11: Laufzeiten (in sec.) des PowerXplorers bei unterschiedlichen Szenen

Für die Laufzeiten des Programms auf den Transputern können keine echten Speed-Ups berechnet werden, da es aufgrund mangelnder Speicherkapazität nicht sequentiell auf einem einzelnen Transputer lauffähig ist. Allerdings zeigt Abbildung 4.10, daß die absolute Laufzeit (in Sekunden) für die erste Szene bei einem Mehreinsatz von Prozessoren abnimmt. Dies gilt im Fall des PowerXplorers ebenfalls für die rechenintensive erste Szene (siehe Tabelle 4.11).

Beim Vergleich der absoluten Laufzeiten des Transputersystems mit den Laufzeiten des PowerXplorers bzw. GC/PPs ist zu bedenken, daß die PowerPC-Prozessoren zwar eine hohe Rechenleistung haben, ihre Kommunikationsleistung aber in keinem ausgewogenen Verhältnis dazu steht. Das Verhältnis Kommunikationsleistung zu Rechenleistung ist bei dem Transputersystem wesentlich günstiger. Daher wird die Laufzeit bei einem größeren Einsatz von Transputern immer besser, während sie sich beim PowerXplorer, aufgrund der vergleichsweise langsamen Kommunikation, eher verschlechtert.

Szene	Laufzeit
6 Bäume, Gouraud	9.27
6 Bäume, Wire-Frame	7.03
1 Baum, Gouraud	4.93
1 Baum, Wire-Frame	3.63

Abbildung 4.12: Laufzeiten (in sec.) des sequentiellen Programms auf einer Sparc 10

Bei einem Vergleich der absoluten Laufzeiten des parallelen Programms auf 4 Prozessoren des PowerXplorers mit den Laufzeiten des sequentiellen Programms auf einer Sparc 10 (siehe Tabelle 4.12, Durchschnitt von 10 Durchläufen) erkennt man, daß es sich auf alle Fälle lohnt, das parallele Programm auf 4 Prozessoren rechnen zu lassen, denn es ist schneller als das sequentielle und hat einen akzeptablen Speed-Up. Ebenso werden sehr gute Laufzeiten erzielt, wenn rechenintensive Szenen auf größeren Topologien berechnet werden (16, 32 Prozessoren), da nur noch ein Drittel bzw. die Hälfte der sequentiellen Rechenzeit benötigt wird. Diese Zahlen werden sich noch verbessern, wenn man die Datenmenge (durch Herabsetzen der Iterationszahl des Bergalgorithmus) ungefähr einheitlich hält.

Kapitel 5

Oberfläche

Zu einem komfortablen Programm gehört eine einfache und übersichtliche Bedienung. Dies kann am besten durch eine interaktive Oberfläche erreicht werden. Sie bietet dem Benutzer eine strukturierte Handhabung und ist daher auch für Laien verständlich. Außerdem können viele Fehlerquellen ausgeschaltet werden, und die Oberfläche gibt dem Programm ein schöneres Aussehen, was gerade in der Computergraphik ein nicht unwesentlicher Aspekt ist.

Aus diesen Gründen wurde für das zuvor beschriebene Programm zur Erzeugung fraktaler Landschaften eine interaktive Oberfläche erstellt. Sie wurde unter dem X Window System mit OSF/Motif und Wcl (Widget Creation Library) entwickelt. Daher soll zu Beginn eine kurze Erläuterung dieser Tools erfolgen, um anschließend auf die Funktionsweise und Bedienung der hier entwickelten Oberfläche einzugehen.

5.1 Das X Window System

Das *X Window System*, kurz auch *X* genannt, ist der Standard zur Erzeugung von graphischen Benutzeroberflächen auf Unix-Workstations. X ist hardwareunabhängig, kann also auf unterschiedlichen Architekturen installiert werden, und netzwerkfähig, d.h. die Ausgabe kann auch auf einem anderen Gerät erscheinen als auf dem Bildschirm des Rechners, auf dem das Programm läuft. Desweiteren ist es multitaskingfähig, so daß mehrere Prozesse gleichzeitig aktiv sein können. Jeder Prozeß bzw. jedes Programm läuft in einem Fenster ab, das als *Top-Level-Fenster* bezeichnet wird und andere Fenster als Kinder besitzen kann. Auf diese Weise sind die Fenster hierarchisch in einer Baumstruktur angeordnet, wobei das oberste Fenster, das *Root-Window*, vom gesamten Bildschirm gebildet wird. Die Top-Level-Fenster erhalten ihre Bezeichnung, da sie direkte Kinder des Root-Windows sind. Über ihre Position und ihr Erscheinungbild entscheidet der *Window-Manager*. Er ist unabhängig von X und kann daher frei gewählt werden (z.B. mwm, fvwm). Über die Plazierung der Kind- und Enkel-Fenster der Top-Level-Fenster entscheidet allerdings nicht der Window-Manager, sondern der Anwendungsprogrammierer selbst.

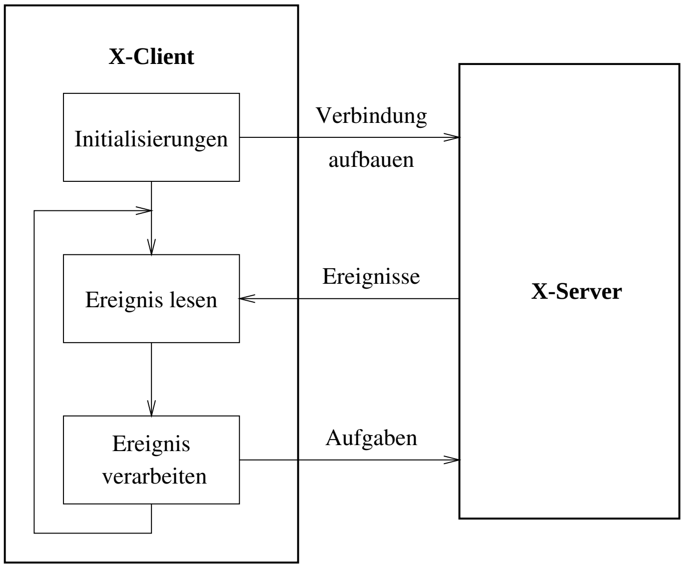

Abbildung 5.1: Struktur von X-Programmen

X basiert auf einem Client-Server-Modell. Auf jedem Rechner läuft ein *X-Server*, der hardwareabhängig ist und eine exklusive Kontrolle über alle Ein- und Ausgabegeräte besitzt. Dieser X-Server kommuniziert mit den *X-Clients*, die die Anwendungen unter X bilden und hardwareunabhängig sind. Er empfängt zum einen Aufträge von den X-Clients (z.B. zum Zeichnen eines Punktes) und setzt diese in Hardwarebefehle um und schickt zum anderen selbst Nachrichten an die X-Clients. Diese Nachrichten sind Mitteilungen über Ereignisse (*Events*), die eingetreten sind (z.B. das Drücken einer Maustaste), und die vom X-Client verarbeitet werden. Die gesamte Kommunikation geschieht über das *X-Protokoll*, welches die Nachrichten puffert und asynchron verschickt. Der X-Client arbeitet ereignisorientiert, d.h. nach einer Initialisierungsphase wartet er in einer Endlosschleife auf Events, die ihm vom X-Server geschickt werden, und verarbeitet diese, bis die Schleife von außen unterbrochen wird (siehe Abbildung 5.1).

Unterstützt wird die X-Programmierung durch eine Vielzahl von Bibliotheken, von denen zwei besonders wichtig sind. Die *Xlib* enthält grundlegende Lowlevel-Funktionen, z.B. zum Zeichnen eines Punktes oder Erzeugen eines einfachen Fensters. Weitergehende Funktionen stellen die *X Toolkit Intrinsics*, kurz auch Xt Intrinsics, zur Verfügung. Mit ihrer Hilfe werden die graphischen Grundobjekte, genannt *Widgets*, erzeugt und verwaltet. Nähere Informationen enthalten [XlPr92], [XlRe92], [XtPr92] und [XtRe92].

5.2 Widgets und das OSF/Motif-Widget-Set

5.2.1 Widgets

Mit dem Begriff *Widget* werden Fenster mit festgelegter Ein- und Ausgabefunktionalität bezeichnet. Diese Funktionalität wird durch ein charakteristisches Aussehen der Widgets unterstützt. Entsprechend der Fensterhierarchie des X-Programms bilden die Widgets einen Widgetbaum. Dabei erben die Kinder die Eigenschaften der Eltern und erweitern diese durch spezielle eigene Eigenschaften.

Die Menge aller Widgets wird in drei verschiedene Gruppen unterteilt:

Display-Widgets bezeichnen die Bedienelemente der Oberfläche, z.B zum Anzeigen oder Abfragen von Daten; sie können keine Kinder haben und bilden daher die Blätter des Widgetbaumes; ein Beispiel wäre ein Button.

Container-Widets sind zur Aufnahme anderer Widgets, ihrer Kind-Widgets, bestimmt; sie legen das geometrische Layout ihrer Kinder fest und besitzen im allgemeinen keine Eingabefunktionalität; ein Beispiel ist ein Menü.

Shell-Widgets sind spezielle Container-Widgets, die mit der Umgebung, vor allem mit dem Window-Manager, kommunizieren; ihre Fenster sind die Top-Level-Fenster der X-Anwendungen, welche jeweils die Wurzel eines Widgetbaumes bilden.

Die Xlib und die Xt Intrinsics stellen keine Widgets selbst, sondern nur Funktionen zu ihrer Erzeugung und Verwaltung zur Verfügung. Zur Vereinfachung der Erstellung einer graphischen Bedieneroberfläche unter X benutzt man ein Widget-Set, z.B. das *OSF/Motif-Widget-Set*[1], das eine Menge vordefinierter Widgets enthält. Ein Widget-Set besteht aus mehreren Widget-Klassen, die das allgemeine Aussehen einer Gruppe von Widgets festlegen. Von diesen werden weitere, speziellere Widgets abgeleitet, die vom Anwendungsprogrammierer bei der Implementation der Oberfläche benutzt werden.

Jedes Widget besitzt *Ressourcen*, durch die sein Aussehen und seine Eigenschaften festgelegt werden können. Diese Ressourcen werden teilweise bei der Erzeugung des Widgets gesetzt, teilweise aber auch nachträglich durch die Xt Intrinsics oder das Widget-Set. Eine weitere Möglichkeit besteht in der Verwendung von Ressource-Dateien.

Einige Widgets können mit Funktionen, sogenannten *Callbacks*, verknüpft werden, die bei Eintreten eines bestimmten Events innerhalb des Widget-Fensters aufgerufen werden. Callbacks können vom Widget-Set vordefinierte oder vom Anwendungsprogrammierer selbst geschriebene Funktionen sein. So kann z.B. nach dem Drücken eines Buttons mit der Bezeichnung Exit eine Funktion zum Beenden des Programms aufgerufen werden.

[1] OSF = Open Software Foundation

5.2.2 Das OSF/Motif-Widget-Set

Ein sehr umfangreiches und komfortables Widget-Set ist das *OSF/Motif-Widget-Set*. Es unterscheidet sich von den meisten anderen Widget-Sets dadurch, daß seine Widgets dreidimensional erscheinen. Auf diese Weise kann z.B. bei Betätigung eines Knopfes der Eindruck erweckt werden, daß dieser tatsächlich gedrückt wurde.

Eine Auswahl von hier verwendeten Widgets soll beispielhaft vorgestellt werden. Dabei werden Display-Widgets, Container-Widgets und Dialog-Widgets unterschieden. Bei den Dialog-Widgets handelt es sich um eine Unterklasse der Shell-Widgets, die speziell zur Kommunikation zwischen Anwendung und Benutzer entwickelt wurden. Die Vorsilbe „Xm“ charakterisiert ein Widget des Motif-Widget-Sets.

Display-Widgets

XmLabel-Widget Mit Hilfe eines Label-Widgets kann Text ausgegeben werden, ohne eine Eingabe des Benutzers zu erwarten. So kann z.B. ein Parameter bezeichnet oder ein Dateiname angegeben werden. Anstelle des Textes kann auch eine Pixmap ausgegeben werden. Als Ressourcen der Label-Widgets können z.B. der Font oder der Abstand des Textes zum Rand gesetzt werden. Die Klasse XmLabel ist eine Oberklasse der Button-Widgets.

XmPushButton-Widget Diese Widgets stellen einen Knopf dar, der aktiviert werden kann, indem der Benutzer einen Mausbutton drückt, während sich der Cursor in dem Widget befindet. Die Betätigung des Knopfes zieht die Ausführung eines Callbacks nach sich. Insgesamt kann auf drei unterschiedliche Arten ein Callback ausgelöst werden. Als Unterklasse der Klasse XmLabel wird der Button mit Text beschriftet oder gibt eventuell eine Pixmap aus.

XmArrowButton-Widget Diese Klasse ist als einzige Button-Klasse nicht der Klasse XmLabel untergeordnet, da ihr Aussehen fest vorgegeben ist. Sie stellt einen Pfeil dar, dessen Richtung (links, rechts, oben, unten) durch eine Ressource gesetzt werden kann. Ein XmArrowButton-Widget besitzt dieselben Callbacks wie ein XmPushButton.

XmSeparator-Widget Mit einem XmSeparator kann eine horizontale oder vertikale Trennlinie gezogen werden. Auf diese Weise können logische oder funktionale Gruppen gebildet werden. Die Form der Linie (einfach, doppelt, gestrichelt, dreidimensional) wird mit Hilfe einer Ressource gesetzt.

Container-Widgets

XmRowColumn-Widget Durch RowColumn-Widgets können aufgrund einer großen Anzahl von Ressourcen eine Vielzahl von Layout-Strategien realisiert werden. Eine verbreitete Anwendung ist die Anordnung von Kind-Widgets gleicher Größe in Zeilen und Spalten. Auf diese Art und Weise werden z.B. viele Menüs erstellt. Man kann die Kind-Widgets aber auch vertikal oder horizontal hintereinander anordnen.

XmScale-Widget Ein XmScale-Widget dient dazu, Werte zwischen zwei Extremwerten anzuzeigen oder einzustellen. Dies kann z.B. eine Länge oder eine Iterationsanzahl sein. Das Widget besteht aus einem Slider, der durch Ziehen der Maus bei gleichzeitig gedrücktem Mausbutton bewegt werden kann. Durch Ressourcen kann festgelegt werden, ob der Slider horizontal oder vertikal ausgerichtet sein soll, mit welchem Text er beschriftet wird und ob er ganze oder Dezimalzahlen anzeigen soll. Callbacks werden sowohl bei Bewegung des Sliders als auch bei Änderung des Wertes aktiviert.

Dialog-Widgets

XmSelectionBox-Widget Eine XmSelectionBox besteht aus einer Liste von Einträgen, aus der der Benutzer mit Hilfe der Maus auswählen kann. Der selektierte Eintrag erscheint in einem editierbaren Textfeld. In dieses Textfeld kann auch von Hand der Name des ausgewählten Eintrags eingegeben werden. Außerdem enthält die XmSelectionBox vier PushButtons, die der Anwendungsprogrammierer mit Callbacks verknüpfen kann. Sie sind standardmäßig mit OK, Apply, Cancel und Help beschriftet. Es besteht die Möglichkeit, den Text zu ändern. Die vorhandenen Ressourcen des XmSelectionBox-Widgets beziehen sich hauptsächlich auf seine Kinder, wie z.B. die Listeneinträge.

XmMessageBox-Widget Bei dieser Klasse handelt es sich um eine Box, die zur Anzeige eines kurzen Informationstextes dient. Zur Verdeutlichung der Informationsart kann die Box zusätzlich mit einer Pixmap, z.B. einem vergrößerten Fragezeichen bei einer Abfrage, versehen werden. Desweiteren enthält das XmMessageBox-Widget drei PushButtons (OK, Cancel, Help), für die Callbacks definiert werden können, z.B. zur Bestätigung oder zur Anzeige eines Hilfetextes. Auch hier kann die Beschriftung geändert werden.

Einen Überblick über weitere wichtige Widgets des OSF/Motif-Widget-Sets und kurze Erläuterungen zu den Themen X Window System und Widgets gibt [BrJo91]. Vertiefende Informationen enthalten [Hel91] und [Fer93].

5.3 Wcl

Ressourcen können nicht nur bei der Erzeugung der Widgets durch die Xt Intrinsics bzw. durch OSF/Motif gesetzt werden, sondern auch in Form von Ressource-Dateien. Dabei handelt es sich um Dateien, die den kompletten Widgetbaum enthalten und zur Laufzeit des Programms interpretiert werden. Um ein Widget zu definieren oder ihm einen Callback zuzuordnen, muß sein kompletter Pfad (anhand des Widgetbaums) angegeben werden. Bei eindeutiger Bezeichnung des Widgets können die Namen der Eltern, Großeltern etc. allerdings auch durch Wildcards („*“) ersetzt werden. Ein gesamter Widgetbaum kann auf mehrere Ressource-Dateien verteilt werden. Es können sogenannte *Template-Files* eingebunden werden, in denen sehr allgemeine Widgets festgelegt sind, die mehrmals benutzt werden (z.B. eine Box mit drei Push-Buttons — OK, Cancel, Help).

Außerdem bietet Wcl eine zusätzliche Widget-Klasse mit Namen „XmpTable". Durch sie werden Container-Widgets definiert, die eine Ressource „layout" besitzen, anhand derer auch unterschiedlich große Kind-Widgets auf einfache Art und Weise durch Spalten- und Zeilenbezeichnungen positioniert werden können.

Um die Verknüpfung zum Anwendungsprogramm der Oberfläche herzustellen, muß in einem Hauptprogramm die Ressource-Datei des Top-Level-Fensters angegeben werden. In dieser Datei sind dann alle weiteren Ressource-Dateien aufzuführen.

Die Entwicklung einer Oberfläche wird durch Verwendung von Ressource-Dateien erheblich beschleunigt und sehr viel übersichtlicher. Desweiteren ist man in der Lage, Ressourcen recht einfach zu ändern, falls das Erscheinungsbild nicht gefällt oder die Oberfläche für ein anderes Programm umgeschrieben werden soll. In vielen Fällen muß das Programm nach einer Änderung nicht noch einmal neu übersetzt werden.

Zur vertiefenden Literatur siehe die beiden Artikel [Smy90] und [Smy91].

5.4 Programmstart

Das Programm und die zugehörige Oberfläche zur interaktiven Erzeugung fraktaler Landschaften wird durch die Eingabe des Befehls `fractals` gestartet. Daraufhin erscheint die Oberfläche, deren Screendump in Abbildung 5.2 dargestellt ist. Sie kann in drei Teile aufgeteilt werden: die Menüleiste oben mit fünf Menüpunkten, das Menüfeld rechts mit drei Push-Buttons und der Bildbereich. Die Menüpunkte der Leiste und die Knöpfe des Menüfeldes können aktiviert werden, der Bildbereich dient nur zur Ausgabe fertig berechneter Bilder.

Nach der Eingabe von `fractals` werden vom Programm die Dimensionen des Bildschirms bestimmt und die Oberfläche darauf ausgelegt. Anschließend werden eine Pflanzen-, eine Berg- und eine Darstellungsdatei mit Werten initialisiert und das entsprechende Bild mit `xwud` in das Fenster geladen.

Zum besseren Verständnis des Programms und der Oberfläche kann der Menüpunkt Hilfe angewählt werden. Daraufhin erscheint ein Fenster, in dem ein Hilfetext allgemeine Erläuterungen zu Programm und Oberfläche gibt. Für speziellere Fragen existiert bei jedem Menü und bei jedem Popup (aufspringendes Widget) ein Hilfe-Knopf.

Die gesamte Oberfläche kann sowohl mit der Maus als auch mit der Tastatur bedient werden. Dazu wurden für die Punkte der Menüleiste und ihre Untermenüpunkte Acceleratoren und Mnemoniks definiert. Dabei handelt es sich um Tastenkombinationen oder einzelne Zeichen, durch die Menüpunkte aktiviert werden können. Bei Eingabe eines Accelerators, z.B. `Ctrl-C` für Datei Ende, spielt es keine Rolle, ob der Menüpunkt sichtbar ist oder nicht. Im Gegensatz dazu muß der Menüpunkt aufgeklappt sein, falls er mit einem Mnemonik, wie e für Ende, angewählt werden

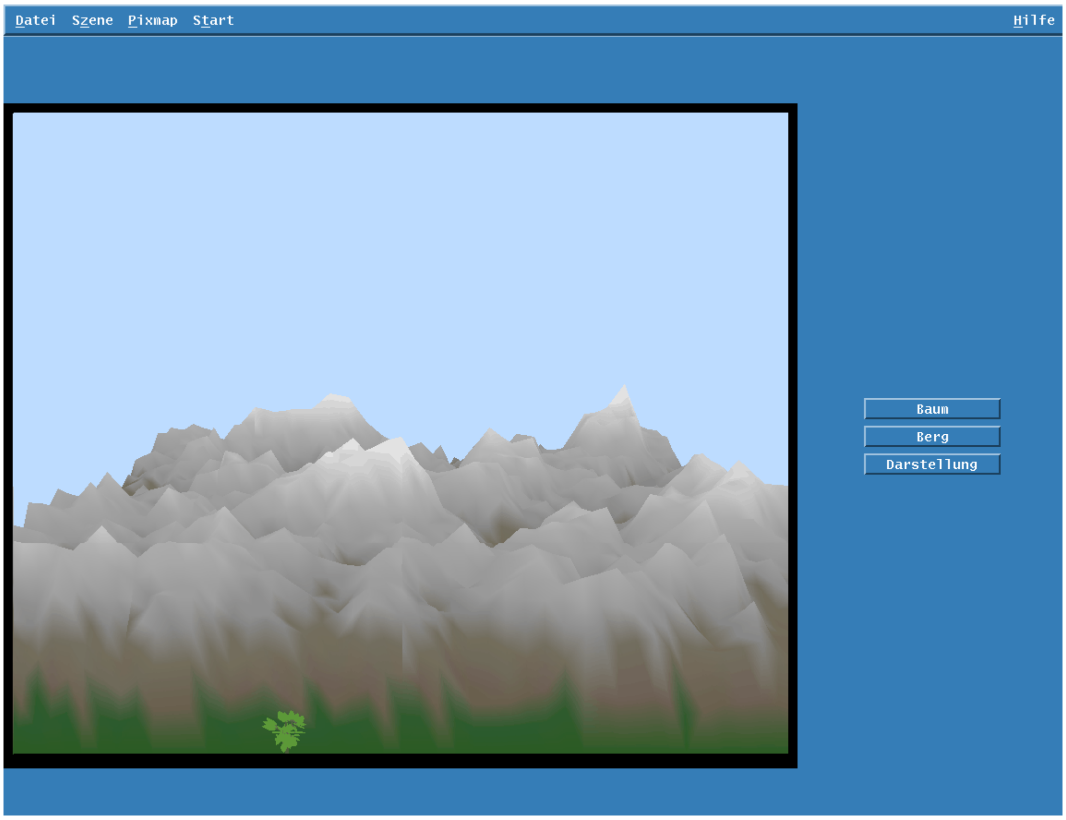

Abbildung 5.2: Oberfläche nach Programmstart

soll. Bei einem Menüpunkt der Menüleiste, die immer sichtbar ist, muß zusätzlich zum Mnemonik die Meta-Taste gedrückt werden. Die Mnemoniks der Menüpunkte sind durch Unterstreichen des entsprechenden Buchstabens gekennzeichnet, und der Text des Accelerators ist hinter dem Menüpunkt angegeben. Innerhalb von Menüs kann man sich mit Hilfe der Cursor-Tasten und der Tabulator-Taste bewegen und Buttons mit der Space- oder Return-Taste betätigen.

5.5 Bearbeiten einer Szene

Um ein einfaches und strukturiertes Arbeiten mit der Oberfläche zu ermöglichen, werden je eine Darstellungs- und Bergdatei sowie beliebig viele Pflanzendateien bzw. Baumdateien zu einer Szene zusammengefaßt. Jedes Daten-File (Darstellung, Berg, Baum) darf in mehreren Szene-Dateien benutzt werden. Allerdings werden bei Manipulation eines Daten-Files in einer Szene die Daten auch für alle anderen, dieses File enthaltenden Szenen geändert.

Es kann immer nur mit der momentan aktuellen Szene gearbeitet werden. Unter der aktuellen Szene versteht man die zur Zeit geöffnete Szene, die immer unter dem Namen `akt_szene.dat` abgespeichert ist. Beim Starten des Programms wird die aktuelle Szene mit Standard-Dateien initialisiert. Damit beim Aufruf des Programms immer dieselbe Ausgangssituation wiederhergestellt wird, dürfen diese Standard-Dateien nicht verändert werden. Um solche und andere mögliche Fehler bei der Bedienung der Oberfläche zu vermeiden, werden, soweit möglich, Fehlerquellen abgefangen und Warnungen oder Fehlermeldungen ausgegeben.

Die Szenendateien können auf unterschiedliche Art und Weise bearbeitet werden. Dazu muß aus dem Untermenü des Menüpunktes Szene der Menüleiste einer der im folgenden beschriebenen Punkte ausgewählt werden.

5.5.1 Öffnen einer bestehenden Szenendatei

Durch die Wahl des Menüpunktes Öffnen kann aus allen existierenden Szenendateien eine neue aktuelle Szene ausgewählt werden. Dazu erscheint die Selection-Box in Abbildung 5.3, in der alle Szenendateien aufgezählt sind. Durch Anklicken eines Namens oder durch Eingabe dieses Namens in das Textfeld und anschließende Betätigung des OK-Buttons (oder durch Doppelklick auf den Namen) kann eine Datei selektiert werden. Wird der OK-Button betätigt, ohne daß zuvor eine Szene ausgewählt wurde, erscheint eine Fehlermeldung.

Abbildung 5.3: Menü zur Auswahl einer Szenendatei

Nach der Auswahl einer Szene erscheint ein Popup, in dem die Daten-Files der Szene angegeben sind. Durch Betätigung des OK-Knopfes wird die ausgewählte Szenendatei zur aktuellen Datei, d.h. die Namen der Daten-Files werden in die Datei `akt_szene.dat` geschrieben. Soll die dargestellte Szene jedoch nicht ausgewählt werden, so kann der Abbruch-Button gedrückt werden, um das Popup verschwinden zu lassen und die Selection-Box erneut aufzuklappen. Der gesamte Auswahlvorgang kann durch Betätigung des Abbruch-Knopfes der Selection-Box beendet werden.

5.5.2 Ändern der aktuellen Szenendatei

Ändern kann man immer nur die aktuell geöffnete Szenendatei. Dazu muß der Menüpunkt Ändern angewählt werden. Daraufhin erscheint das Popup in Abbildung 5.4, in dem die Namen der in der momentan aktuellen Szene zusammengefaßten Daten-Files aufgeführt sind.

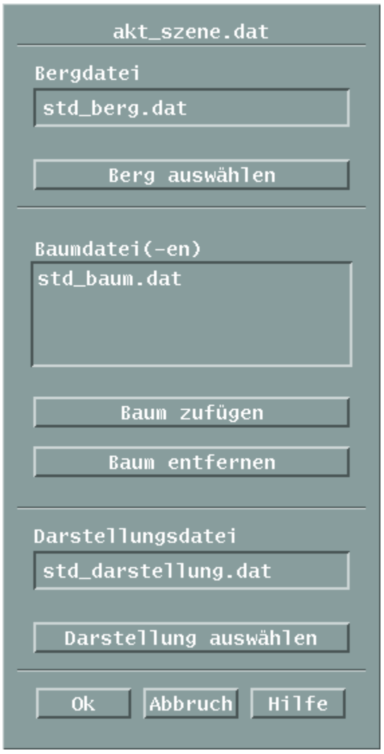

Abbildung 5.4: Menü zum Ändern der aktuellen Szenendatei

Außer dem Kontrollblock (OK, Abbruch, Hilfe) sind vier weitere Buttons vorhanden, mit deren Hilfe die Szenendatei geändert werden kann:

- Berg auswählen: Mit diesem Knopf kann eine andere Bergdatei für die Szene ausgewählt werden. Nach Drücken des Buttons erscheint eine Selectionbox, in der alle Bergdateien aufgeführt sind. Durch Anklicken eines Namens und anschließendem OK wird eine Datei selektiert, und es erscheint ein Popup, in dem die Werte der Bergdatei angezeigt werden. Wird in diesem Popup der OK-Knopf aktiviert, so wird die Bergdatei als neue Bergdatei gewählt und das Szene-Popup erscheint erneut, wobei die Bergdatei bereits eingefügt wurde. Eine Betätigung des Abbruch-Knopfes im Berg-Popup zieht ein erneutes Erscheinen der Selectionbox nach sich. Wird in dieser Box der Abbruch-Button gedrückt, so wird die Bergdatei nicht ausgetauscht, und es wird erneut das Szene-Popup aufgeklappt.

- Baum zufügen: Nach Aktivierung dieses Buttons erscheint eine Selectionbox mit allen Baum- bzw. Pflanzendateien, aus der auf dieselbe Weise wie bei der Änderung der Bergdatei eine neue Baumdatei ausgewählt werden kann. Die selektierte Datei wird der bereits bestehenden Liste von Pflanzendateien zugefügt. Allerdings gibt es eine Maximalzahl von Bäumen, bei deren Überschreitung die in Abbildung 5.5 dargestellte Fehlermeldung erscheint.
- Baum entfernen: Aus der Liste der Pflanzen- bzw. Baumdateien können Dateien gestrichen werden. Dazu muß die zu entfernende Datei angeklickt und anschließend der Knopf Baum entfernen aktiviert werden. Daraufhin erscheint ein Dialog zur Abfrage, ob die ausgewählte Datei tatsächlich entfernt werden soll. Bei Betätigung des OK-buttons wird die Datei aus der Liste gestrichen, bei Drücken des Abbruch-Knopfes bleibt sie erhalten. In beiden Fällen erscheint erneut das Popup zur Darstellung der Szenendatei. Ist nur noch eine Pflanze in der Liste vorhanden, so darf diese nicht gelöscht werden und daher erscheint eine Fehlermeldung.
- Darstellung auswählen: Dieser Button dient zum Einfügen einer anderen Darstellungsdatei. Die Verfahrensweise ist dabei dieselbe wie bei der Auswahl einer anderen Bergdatei, d.h. es wird eine Darstellungsdatei ausgewählt und in einem entsprechenden Popup dargestellt, bevor eine endgültige Auswahl getroffen wird.

Abbildung 5.5: Fehlermeldung bei Überschreitung der maximalen Anzahl von Bäumen

Ist auf die oben beschriebene Art und Weise die aktuelle Szenendatei geändert worden, so wird durch Aktivierung des OK-Knopfes im Szene-Popup die Änderung bestätigt und die neuen Daten werden in der Datei `akt_szene.dat` abgespeichert. Durch Betätigung des Abbruch-Knopfes werden die Änderungen ignoriert und die aktuelle Szene bleibt in ihrem Ausgangszustand erhalten.

5.5.3 Abspeichern der aktuellen Szenendatei

Soll die Änderung der aktuellen Szenendatei erhalten bleiben, so muß sie unter einem anderen Namen abgespeichert werden. Dazu wird der Menüpunkt Speichern angewählt. Es erscheint ein Popup mit den Szenedaten. Wird nun der OK-Button aktiviert, klappt der Dialog in Abbildung 5.6 zur Eingabe eines Dateinamens auf.

Abbildung 5.6: Dialoge zur Eingabe und zur Bestätigung eines Dateinamens

Nach der Angabe eines Namens und anschließendem OK wird mit Hilfe eines zweiten Dialogs (siehe Abbildung 5.6) abgefragt, ob die aktuelle Szene unter dem eingegebenen Namen abgespeichert werden soll, eventuell mit einer Mitteilung, daß bereits eine Datei mit diesem Namen existiert. Durch Betätigung des OK-Knopfes dieses Dialogs wird die Speicherung vollzogen. Soll allerdings ein anderer Dateiname gewählt werden, kann man durch Drücken des Abbruch-Knopfes erneut zum ersten Dialog gelangen und einen neuen Dateinamen eingeben. Wird jedoch auch hier der Abbruch-Button aktiviert, so wird der gesamte Speichervorgang abgebrochen.

5.5.4 Erzeugen einer neuen Szenendatei

Da nicht nur die Möglichkeit existieren soll, bereits vorhandene Szenen zu ändern, sondern auch komplett neue Szenen zu erstellen, ist ein weiterer Menüpunkt Neu eingerichtet worden. Wird dieser Punkt ausgewählt, so erscheint dasselbe Popup wie beim Ändern einer Szene (siehe Abbildung 5.4). Die Berg-, Darstellungs- und Baumdateien sind in diesem Fall mit Standarddateien initialisiert. Sie können jedoch durch die oben (siehe „Ändern") beschriebenen Aktionen mit Hilfe der vier Buttons (Berg auswählen, Baum zufügen, Baum entfernen, Darstellung auswählen) ausgetauscht werden. Auf diese Art und Weise kann eine neue Szenendatei gestaltet werden.

Um der Datei einen Namen geben zu können, muß der OK-Button aktiviert werden, woraufhin ein Dialog zur Eingabe eines Dateinamens für die neue Szene erscheint.

5.6 Arbeiten mit einer Pixmap

Nach der Berechnung des Bildes einer Szene wird das entstandene PPM-File durch das Postprocessing in ein XWD-File umgewandelt. Dieses XWD-File wird durch die Utility `xwud` am Bildschirm angezeigt. Um ein solches File auch später noch einmal laden zu können, sind die Menüpunkte Speichern und Laden für eine Pixmap (entspricht dem XWD-File) eingefügt worden.

5.6.1 Speichern der aktuellen Pixmap

Nach der Berechnung des Bildes zur aktuellen Szene wird der X Window Dump unter dem Namen `akt_pixmap.dat` abgelegt. Um einen späteren Zugriff auf diese Pixmap zu ermöglichen, kann sie mit Hilfe des Menüpunktes Speichern unter einem neuen Namen abgespeichert werden. Dazu erscheint nach der Wahl dieses Punktes ein Dialog, in dem der Dateiname eingegeben werden muß. Betätigt man anschließend den OK-Button, so klappt ein zweiter Dialog auf, in dem man den zuvor eingetippten Namen bestätigen soll. Eventuell weist eine zusätzliche Mitteilung darauf hin, daß bereits eine Datei mit diesem Namen existiert. Durch Drücken des OK-Knopfes dieses Dialogs wird der Speichervorgang aktiviert und die Pixmap bzw. das XWD-File unter dem angegebenen Namen abgelegt. Soll der Name jedoch nochmal geändert werden, so gelangt man mit Hilfe des Abbruch-Knopfes zum ersten Dialog zurück.

5.6.2 Laden einer Pixmap

Soll eine zuvor abgespeicherte Pixmap erneut geladen werden, so ist der Menüpunkt Laden anzuwählen. Daraufhin kommt eine Selectionbox hoch, in der die Namen aller Pixmaps angegeben sind. Durch Anklicken mit der Maus und anschließender Betätigung des OK-Knopfes oder durch ein doppeltes Anklicken des Namens wird eine Datei ausgewählt und im Bildfenster der Oberfläche dargestellt. Drückt man den OK-Button, ohne vorher eine Pixmap ausgewählt zu haben, so erscheint eine Fehlermeldung.

5.7 Starten der Programme

Die Programme zur Berechnung eines neuen Bildes, ob sequentiell oder parallel, können aus der Oberfläche heraus gestartet werden. Dazu ist der Menüpunkt Start anzuwählen. Es erscheint ein Popup, in dem die Abarbeitungsart (sequentiell oder parallel) festgelegt werden muß. Nach Betätigung des OK-Knopfes klappt ein weiteres Popup auf, von dem aus das entsprechende Programm gestartet werden kann.

Direkt nach seiner Berechnung enthält das Bild mehr Farben als auf dem Bildschirm gleichzeitig dargestellt werden können. Daher wird festgestellt, wieviele freie Farbeinträge die Colormap des Bildschirms momentan besitzt, und im Postprocessing die Anzahl der Farben des Bildes auf die Anzahl freier Einträge der Colormap heruntergerechnet. Das Ergebnis, d.h. der XWD, wird mit Hilfe eines etwas veränderten `xwud` im Bildfenster der Oberfläche dargestellt.

Während der Berechnung eines neuen Bildes erhält der Cursor die Form einer Uhr, um anzuzeigen, daß die Oberfläche nicht aktiv ist. Allerdings kann in allen Fenstern außerhalb der Oberfläche weitergearbeitet werden. Ebenso ist es möglich, das Top-Level-Fenster der Applikation zu verschieben oder zu verkleinern.

5.7.1 Start des sequentiellen Programms

Wurde eine sequentielle Abarbeitung für die Berechnung des Bildes gewählt, so erscheint das in Abbildung 5.7 dargestellte Popup zum Starten des sequentiellen Programms.

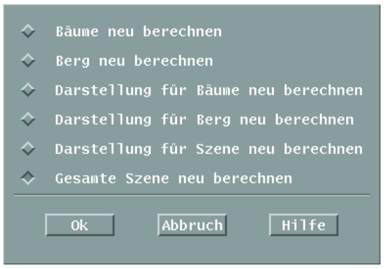

Abbildung 5.7: Menü zum Start des sequentiellen Programms

Es enthält sechs Knöpfe zur Auswahl, deren Funktion kurz beschrieben werden soll:

- Bäume neu berechnen: Bei Wahl dieses Punktes werden ausschließlich Daten für die Pflanzen der Szene anhand der L-System-Dateien berechnet; es findet keine weitere Bearbeitung der Daten durch den Darstellungsalgorithmus statt und daher erfolgt auch keine Ausgabe eines neuen Bildes im Bildfenster.
- Berg neu berechnen: Es werden nur die Daten für den Berg der Szene neu ermittelt; auch hier findet keine Weiterverarbeitung dieser Daten und keine Anzeige eines neuen Bildes statt.

Beide Knöpfe sind für den Fall eingerichtet worden, daß nur ein Teil der Szene (Berg oder Pflanzen) verändert wurde und neu berechnet werden soll. Um aus den entstandenen Daten ein Bild zu erzeugen, müssen sie erst den Darstellungsalgorithmus durchlaufen.

- Darstellung für Bäume neu berechnen: Wurde dieser Knopf angewählt, so werden nur die Daten der Pflanzen mit Hilfe der zugehörigen Darstellungsdatei der Szene weiterverarbeitet; es erfolgt zuvor keine neue Berechnung der Pflanzendaten; das entstandene Bild, welches natürlich nur aus Pflanzen besteht, wird (nach dem Postprocessing) in das Bildfenster geladen.
- Darstellung für Berg neu berechnen: Bei Wahl dieses Knopfes werden ausschließlich die Bergdaten von der Darstellung verarbeitet; das anschließend angezeigte Bild besteht daher nur aus dem Berg.

Diese beiden Knöpfe dienen dazu, die Darstellung von Berg und Pflanzen trennen zu können und für dieselben Daten mehrere Darstellungen berechnen zu können.

- Darstellung für Szene neu berechnen: Mit der Wahl dieses Knopfes werden sowohl Pflanzen- als auch Bergdaten bei der Darstellung verarbeitet; das Ergebnisbild wird in das Bildfenster geladen; auf diese Weise kann dieselbe Szene von unterschiedlichen Positionen betrachtet werden.
- Gesamte Szene neu berechnen: Dieser Punkt ist initial eingestellt, da er zur problemlosen, schnellen Berechnung eines ersten Bildes dient; es werden Daten für die Pflanzen und den Berg erzeugt und durch die Darstellung weiterverarbeitet; das Bild wird im Bildfenster angezeigt.

Durch die Betätigung des OK-Knopfes wird das ausgewählte Programm gestartet. Tritt während des Programmdurchlaufes ein Fehler auf, d.h. endet das Programm nicht erwartungsgemäß, so wird dies dem Benutzer in einer Fehlermeldung mitgeteilt.

5.7.2 Start des parallelen Programms

Hat man sich für das parallele Programm entschieden, so klappt das Popup in Abbildung 5.8 auf, in dem vier Knöpfe zur Auswahl der Topologiegröße vorgesehen sind. Wählt man eine Topologie mit 64, 32 oder 16 Prozessoren, so wird das parallele Programm auf einem Transputergitter dieser Größe gestartet. Bei Wahl von vier Prozessoren läuft es auf dem PowerXplorer.

Beim ersten Aufruf mit einer Topologiegröße wird das Programm auf ein Prozessorgitter entsprechender Größe geladen und gestartet. Nach seiner Berechnung wird das enstandene Bild (nach dem Postprocessing) im Bildfenster der Oberfläche angezeigt. Das Programm läuft weiterhin auf der gewählten Topologie und wartet auf die erneute Eingabe einer Szene zur Berechnung eines weiteren Bildes. Dadurch fällt die *Startup-Zeit*, die zum Starten der Prozessoren etc. benötigt wird, nur beim ersten Lauf des Programms auf einer Topologiegröße an. Anschließend ist diese Topologie sofort zur erneuten Berechnung bereit. Soll allerdings die Topologiegröße für den Transputer geändert werden, so muß man zuvor das laufende Programm beenden, um das belegte Transputergitter freizugeben. Dazu wurden zwei weitere Knöpfe eingerichtet:

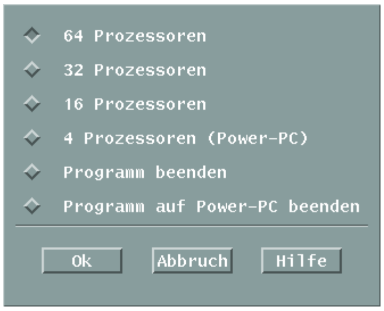

Abbildung 5.8: Menü zum Start des parallelen Programms

- Programm beenden: Beendet das eingabebereite Programm auf den Transputern.
- Programm auf Power-PC beenden: Beendet das eingabebereite Programm auf dem Power-Xplorer.

Bei Wahl eines dieser Knöpfe erhält das entsprechende Programm als Eingabe eine Mitteilung zur Beendigung. Daraufhin werden die Prozessoren freigegeben und ein anderes Prozessorgitter kann belegt werden. Wird versucht, das Programm auf einer anderen Gittergröße zu starten, ohne vorher das laufende Programm beendet zu haben, so erscheint eine Fehlermeldung.

5.8 Ändern der aktuellen Szenedaten

Zur Berechnung des Bildes einer Szene werden Daten für den Berg, die Pflanzen (bzw. Bäume) und die Darstellung benötigt. Diese sind abgespeichert in Dateien, welche wiederum zu Szenen zusammengefaßt sind. Um Berg-, Pflanzen- und Darstellungsdaten manipulieren zu können, wurde das rechte Menüfeld in der Oberfläche eingerichtet.

5.8.1 Ändern der Pflanzendaten

Bei Betätigung des Knopfes mit der Aufschrift Baum wird das Menüfeld durch eine Selectionbox ersetzt, in der, wie gewohnt, aus den Pflanzendateien der aktuellen Szene eine Datei zur Bearbeitung ausgewählt werden kann. Ist dies geschehen und wurde der OK-Button gedrückt, erscheint an derselben Stelle das Menü aus Abbildung 5.9, das die momentanen Werte der ausgewählten Baumdatei bzw. des L-Systems anzeigt. Diese Werte können vom Benutzer verändert werden. Ihre Bedeutung soll kurz erklärt werden. Ausführlichere Informationen enthält das Kapitel 2 „Fraktale“.

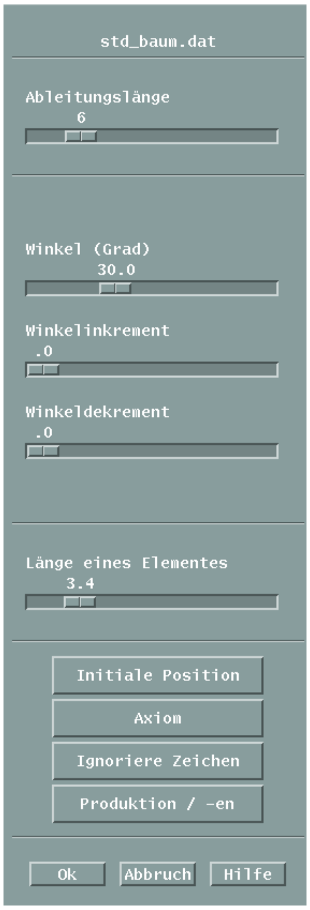

Abbildung 5.9: Menü zum Ändern der Baumdaten

- **Ableitungslänge**: Die Ableitungslänge gibt an, wie oft der String des L-Systems ausgehend vom Axiom abgeleitet werden soll, d.h. in wievielen Iterationen die Produktionen auf den String angewendet werden.
- **Winkel**: Es wird der Winkel eingestellt, um den die Turtle bei einer Drehung rotieren soll, wenn sie bei der graphischen Interpretation des Strings auf einen Befehl zum Drehen trifft.
- **Winkelinkrement**: Mit diesem Slider kann der Betrag eingestellt werden, um den sich der Winkel vergrößern soll, falls ein entsprechendes Zeichen im String auftritt.
- **Winkeldekrement**: Das Winkeldekrement ist das Gegenstück zum Winkelinkrement, da hier der Betrag, um den sich der Winkel verkleinert, manipuliert werden kann.

- Länge eines Elementes: Dieser Wert gibt an, wie lang in Einheiten des Weltkoordinatensystems ein Schritt der Turtle sein soll.
- Initiale Position: Bei Betätigung dieses Knopfes klappt ein Feld zur Eingabe der Ausgangsposition der Turtle im Weltkoordinatensystem auf. x-, y- und z-Koordinaten können anhand von Arrow-Buttons erhöht und erniedrigt werden. Wird der OK-Button betätigt, so werden die neuen Werte vorgemerkt. Drückt man dagegen den Abbruch-Knopf, so bleiben die alten Werte erhalten. In beiden Fällen verschwindet das Feld, und das Menü erscheint wieder in seinem ursprünglichen Zustand.
- Axiom: Wird dieser Knopf gedrückt, erscheint ein editierbares Textfeld zur manuellen Eingabe des Axioms. Eventuelle Manipulationen werden nur bei Aktivierung des OK-Knopfes gemerkt.
- Ignoriere Zeichen: Bei Wahl dieses Knopfes klappt ein Textfeld zur Veränderung der Zeichenmenge auf, die bei der Suche des rechten und linken Kontextes einer Produktion ignoriert werden soll. Wiederum kann das Feld durch OK und Abbruch verlassen werden, wobei Änderungen nur bei Drücken des OK-Knopfes vorgemerkt werden.
- Produktionen: Mit Hilfe dieses Punktes können in einem Textfeld die Produktionen für das L-System manipuliert werden. Sie müssen in der Form

  $$l.\ Kontext < Vorgänger > r.\ Kontext -- > (.x)Nachfolger, (.y)Nachfolger, \ldots$$

  eingegeben werden. x und y stehen für die Wahrscheinlichkeiten (in Prozent), mit denen die Nachfolger gewählt werden. Durch Drücken des OK-Knopfes werden Änderungen gemerkt, und das Textfeld verschwindet. Mit Abbruch kann das Feld verlassen werden, ohne die neuen Änderungen vorzumerken.

Sind alle beabsichtigten Änderungen durchgeführt, so werden sie erst durch Drücken des OK-Knopfes des Menüs unter dem Namen der bearbeiteten Datei abgespeichert. Der Name wird zur Erinnerung in einem Label oben im Menü angezeigt. Um die Datei auch unter einem anderen Namen ablegen zu können — wenn z.B. die gleiche Pflanze an anderer Stelle nochmals auftreten soll — erscheint nach Aktivierung des OK-Knopfes ein Dialog zur Abfrage, ob die Daten in einer weiteren Datei abgespeichert werden sollen. Wird dies mit OK bejaht, kann in dem neu aufklappenden Textfeld ein Dateiname eingegeben werden.

Die unterschiedlichen Dialoge und Eingabefelder können jeweils mit dem Abbruch-Button verlassen werden, ohne daß Aktionen ausgeführt werden. Insbesondere verschwindet bei Drücken des Abbruch-Knopfes das Menü zur Manipulation des L-Systems, ohne daß Veränderungen abgespeichert werden, und es erscheint erneut die Selectionbox zur Auswahl einer anderen Pflanzendatei.

Wurden im Menü falsche Werte eingegeben, z.B. falsche Zeichen bei der Angabe der Produktionen, so erscheint nach der Betätigung des OK-Knopfes die Fehlermeldung in Abbildung 5.10 und die Daten werden nicht abgespeichert.

Abbildung 5.10: Fehlermeldung bei Eingabe falscher Werte

5.8.2 Ändern der Bergdaten

Sollen die aktuellen Daten zur Generierung des Szenen-Berges verändert werden, so ist der Knopf Berg im Menüfeld zu aktivieren. Daraufhin erscheint das Menü aus Abbildung 5.11, in dem die aktuelle Bergdatei und ihre Daten angezeigt sind. Mit der Hilfe dieses Menüs können die Werte zur Bergerzeugung manipuliert werden. Sie sollen im folgenden kurz erläutert werden. Weitergehende Informationen enthält das Kapitel 2 „Fraktale".

- Iterationen: Dieser Wert gibt die Anzahl der Durchläufe des Algorithmus zur Erzeugung des Berges an. In jeder Iteration wird die quadratische Gitterfläche, auf der der Berg entsteht, verfeinert und die Punkte des Gitters werden um einen zufälligen Betrag erhöht. Je mehr Iterationen, desto zerklüfteter und höher wird der Berg.
- Bergfaktor: Mit dem Bergfaktor kann man einstellen, welcher Betrag (multipliziert mit einem Zufallswert) auf die Höhe jedes Gitterpunktes in einer Iteration addiert wird.
- Schrumpffaktor: Der Bergfaktor wird nach jeder Iteration mit dem Schrumpffaktor multipliziert, um ihn zu verkleinern.
- Gewicht Außenseite: Die neuen Punkte eines verfeinerten Gitters erhalten ihren Höhenwert durch lineare Interpolation ihrer Nachbarwerte. Dabei wird der jeweilige Außenpunkt mit dem hier eingestellten Gewicht multipliziert, der jeweilige Innenpunkt mit 1 minus dem Außengewicht.
- Breite und Tiefe: Dieser Wert bestimmt die Breite und Tiefe des Berges in Einheiten des Weltkoordinatensystems.
- Niedrigster Farbwert: Mit Hilfe dieses Sliders kann angegeben werden, welcher der minimale Index in einem Feld von 32 Farben bei der Berechnung der Farbwerte für die einzelnen Höhenwerte sein soll. Der niedrigste Farbwert entspricht in etwa der dunkelsten Farbe.
- Höchster Farbwert: Der höchste Farbwert bezeichnet den maximalen Index in dem Farbfeld. Er entspricht quasi der hellsten Farbe.

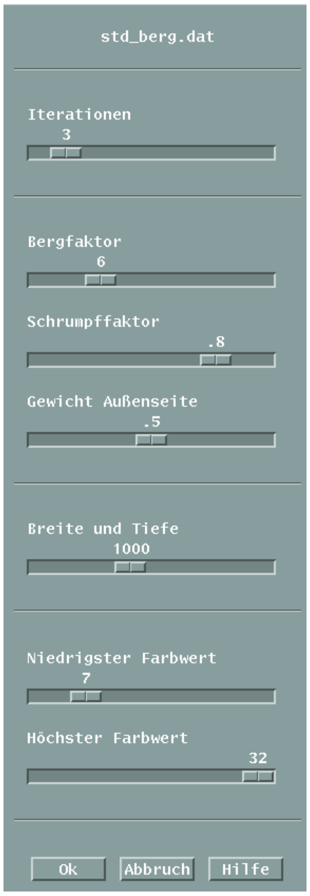

Abbildung 5.11: Menü zum Ändern der Bergdaten

Nachdem die gewünschten Werte verändert wurden, können sie durch Drücken des OK-Knopfes in der Bergdatei der aktuellen Szene abgespeichert werden. Falls sie auch noch unter einem anderen Namen abgelegt werden sollen, ist in dem daraufhin aufklappenden Dialog der OK-Knopf zu aktivieren. Nun kann auf dieselbe Art und Weise wie bei der Änderung der Pflanzendaten ein neuer Dateiname eingegeben werden.

5.8.3 Ändern der Darstellungsdaten

Drückt man den Knopf Darstellung im Menüfeld, so erhält man die Möglichkeit, die Daten für die aktuelle Darstellung, also für die Viewing-Pipeline, die Beleuchtung und das Shading, einzustellen. Ein Screendump des Menüs zur Einstellung der Darstellungsdaten ist in Abbildung

5.12 dargestellt. Es folgt eine kurze Erklärung der einzelnen Menüpunkte. Für genauere Informationen wird auf das Kapitel 2 „Fraktale“ verwiesen.

Abbildung 5.12: Menü zum Ändern der Darstellungsdaten

- **Betrachteter Punkt**: Wird dieser Button aktiviert, so erscheint anstelle der Menüknöpfe ein Feld zur Eingabe des Punktes, der in der Szene vom Betrachter anvisiert wird (VRP). x-, y- und z-Wert sind in Einheiten des Weltkoordinatensystems angegeben und können mit Hilfe der Arrow-Buttons manipuliert werden. Mit dem OK-Knopf werden eventuelle Änderungen vorgemerkt, und das Feld wird wieder durch die Menüknöpfe ersetzt.
- **Betrachterstandpunkt**: Die Position des Betrachters in der Szene (PRP) kann nach Betätigung dieses Knopfes manipuliert werden. Sie wird ebenfalls in Einheiten des Weltkoordi-

natensystems angegeben und mit den Arrow-Buttons erhöht oder herabgesetzt. Sollen die neuen Werte gemerkt werden, so ist der OK-Button zu drücken.

- Kameraorientierung: Nach der Betätigung dieses Knopfes erscheint ein Feld zur Eingabe des Winkels, um den die Kamera nach links (positiver Wert) oder rechts (negativer Wert) gedreht wird (VUV). Der Winkel wird in Grad gemessen und kann durch Drücken der Arrow-Buttons erhöht oder herabgesetzt werden. Die Änderungen werden wiederum nur dann gemerkt, wenn zum Verlassen des Feldes der OK-Knopf gedrückt wird.

- Ausmaße Sichtfenster: Mit dem Menü, das nach der Aktivierung dieses Knopfes erscheint, kann die Größe des Fensters, mit dem der betrachtete Szenenausschnitt begrenzt wird, (View Window) festgelegt werden. Die Werte werden durch die Arrow-Buttons manipuliert.

- Standort Lichtquelle: In dem aufklappenden Menü kann die Position der Lichtquelle in der Szene angegeben werden. Sie wird in Einheiten des Weltkoordinatensystems festgelegt und durch Drücken der Arrow-Buttons verändert.

- Intensität Lichtquelle: Mit diesem Slider wird die Intensität der Lichtquelle, mit der die Szene beleuchtet wird, eingestellt.

- Diffuser Reflexionskoeffizient: Der diffuse Reflexionskoeffizient legt das Ausmaß der Fähigkeit des Objektmaterials, Licht zu reflektieren, fest.

- Intensität Hintergrund: Dieser Wert bezeichnet die Intensität der Hintergrundbeleuchtung.

- Ambienter Reflexionskoeffizient: Durch diesen Slider wird das Zurückstrahlen der Hintergrundbeleuchtung eingestellt.

- Wire-Frame: Bei der Wire-Frame-Darstellung werden lediglich die Kanten der Dreiecke, in die alle Polygone zerlegt werden, geshadet. D.h., die Szene wird nur in einer Netzstruktur dargestellt.

- Gouraud-Shading: Wird als Shading-Modus das Gouraud-Shading gewählt, so werden alle Polygonflächen komplett geshadet.

Ebenso wie bei dem Menü für die Bergdaten und dem Menü für die Pflanzendaten werden auch die Daten dieses Menüs nur in dem Fall gespeichert, daß der OK-Button aktiviert wird. Daraufhin gibt es wiederum die Möglichkeit, die Werte in einer weiteren Datei abzuspeichern. Sollen die ursprünglichen Werte der Datei nicht verändert werden, so muß das Darstellungsmenü mit dem Abbruch-Knopf verlassen werden.

5.9 Programmende

Zur Beendigung des Programms `fractals` ist im Menüpunkt Datei der Unterpunkt Ende anzuwählen. Daraufhin erscheint ein Dialog zur Abfrage, ob man das Programm beenden, d.h. die Oberfläche verlassen, möchte. Ist dies der Fall, so drückt man den OK-Knopf, ansonsten kann mit Hilfe des Abbruch-Buttons der Vorgang unterbrochen werden.

Vor der Beendigung des Programms und dem Abräumen der Oberfläche wird getestet, ob eventuell noch ein paralleles Programm läuft und auf Eingabe wartet. Trifft dies zu, so wird es korrekt beendet.

Kapitel 6

Zusammenfassung und Ausblick

Mit dem Programm `fractals` wurde eine komfortable Applikation zur interaktiven Erstellung von fraktalen 3D-Szenen auf Parallelrechnern geschaffen. Zur Beschleunigung der Laufzeit wurden sowohl die Implementation des sequentiellen Algorithmus möglichst effektiv gestaltet als auch eine effiziente Parallelisierung des Programms durchgeführt, die auf einem Transputernetz oder einem PowerXplorer bzw. einem GC/PP mit bis zu 64 Prozessoren lauffähig ist. Die Implementierung des parallelen Programms geschah unter der von Parsytec entwickelten Laufzeitumgebung Parix.

Als elementare Objekte der dreidimensionalen Landschaftsszenen werden fraktale Berge und Pflanzen benutzt. Die Erzeugung der Objektdaten geschieht mit Hilfe von Beschreibungsdateien, in denen Parameter zur Manipulation der Objekte abgelegt sind. Die fraktalen Pflanzen werden dabei anhand von Lindenmayer-Systemen, die durch eine Turtle-Graphik interpretiert werden, generiert.

Zur Projektion der dreidimensionalen Szene auf den zweidimensionalen Bildschirm wird das Modell einer synthetischen Kamera benutzt. Der Anwender kann den betrachteten Punkt, den Betrachterstandpunkt, die Lage der Kamera und den darzustellenden Ausschnitt der Szene festlegen. Anhand dieser Werte wird eine Viewing-Pipeline erstellt, welche von allen Objekten vor ihrer Darstellung durchlaufen wird und die Projektion dieser Objekte auf Bildschirmkoordinaten durchführt. Als besonders effizient für die Minimierung des Rechenaufwandes erweist sich hierbei das Clipping und das Back-Face-Culling. Bei der Darstellung der Szene kann der Anwender zwischen einem Drahtmodell und gouraud-geshadeten Objekten wählen. Außerdem ist es möglich, die Position der Lichtquelle und die Oberflächeneigenschaften der Szenenobjekte festzulegen.

Durch die Parallelisierung des sequentiellen Algorithmus wird die Laufzeit des Programms erheblich beschleunigt. Insbesondere extrem rechenaufwendige Szenen werden erst durch die parallele Berechnung in einer annehmbar kurzen Zeit fertiggestellt. Durch die Implementation des Programms für verschiedene Topologiegrößen kann der Anwender die jeweils passende Alternative für ein Problem auswählen. Als vorteilhaft erweist sich außerdem die Aufteilung der sequentiellen Programme — Erzeugung der Bergdaten, Generieren der Pflanzendaten und Rendern dieser

Daten — auf drei Prozessorgruppen. Dabei wird sowohl eine Objektraumaufteilung durch die Bergprozessoren als auch eine Bildraumaufteilung durch die Renderingprozessoren vorgenommen.

Für die Applikation `fractals` wird eine Oberfläche zur interaktiven Bedienung des Programms zur Verfügung gestellt. Sie enthält ein Fenster, in dem die berechneten Bilder dargestellt werden, und verschiedene Menüs zur Manipulation der Parameter für die Erzeugung der Berg-, Pflanzen- und Darstellungsdaten. Außerdem wird das Programm zur Berechnung eines Bildes, ob parallel oder sequentiell, aus der Oberfläche heraus gestartet. Jedes erzeugte Bild kann abgespeichert und später erneut geladen werden.

Durch die Implementation des Programms in ANSI-C ist es auf verschiedene Architekturen portierbar. Ebenso ist durch die Verwendung des X Window Systems eine hardwareunabhängige und damit portable Oberfläche erstellt worden. Die Entwicklungszeit dieser Oberfläche wurde durch den Einsatz des OSF/Motif-Widget-Sets zusammen mit Wcl erheblich verkürzt.

Mit dem vorgestellten Programm `fractals` wurde eine interaktive Applikation zur Erzeugung fraktaler Landschaften geschaffen. Doch es bietet in den diversen Bereichen noch vielfältige Möglichkeiten zur Erweiterung.

So werden die vom Programm erzeugten Pixmaps bislang unter dem XWD-Format abgespeichert. Es wäre jedoch wünschenswert, sie auch in anderen Formaten sichern zu können.

Ebenso bietet die Oberfläche zur Bedienung des Programms Erweiterungsmöglichkeiten. Zur Zeit werden die Parameter der Fraktale und der synthetischen Kamera direkt vom Anwender manipuliert. Für Laien ist der Sinn vieler Parameter aber nicht unbedingt von Beginn an verständlich. Es wäre daher eventuell von Vorteil, die Parameter nicht namentlich aufzuführen, sondern sie bildlich zu umschreiben, und auch ihre Bewertung statt mit Zahlen mit Begriffen oder sogar Graphiken durchzuführen.

Das Hinzufügen neuer Algorithmen zur Erzeugung von Fraktalen stellt eine weitere Alternative dar, das vorgestellte Programm noch interessanter und vielfältiger zu gestalten. In [La1/92], [La2/92] und [Pei88] werden zum Beispiel Algorithmen zur Erzeugung von Sternfraktalen, fraktalen Wolken und fraktalen Farnen vorgestellt. Diese Objekte wären eine passende Ergänzung zu einer Landschaft aus Bergen und Pflanzen.

Außerdem besteht die Möglichkeit, weitere Algorithmen zur Erzeugung eines fraktalen Berges zu verwenden. So werden in [Pei88] faszinierende 3D-Bilder einer Mandelbrotmenge und einer Cantormenge gezeigt, welche die Struktur fraktaler Berge widerspiegeln.

Durch die Darstellung fraktaler Objekte können mit Hilfe der Computergraphik schöne und sehr natürliche Landschaftsbilder erzeugt werden. Die Berechnung dieser Graphiken wird durch den Einsatz von Parallelrechnern spürbar beschleunigt. Mit dieser Arbeit wurde gezeigt, wie vielfältig und interessant der Einsatz von Fraktalen in der Computergraphik ist und wie effizient parallele Computergraphikprogramme arbeiten.

Anhang A

Grammatik der Lindenmayer-Systeme

Turtlebefehl	Graphische Bedeutung
F	gehe um eine Längeneinheit vor und zeichne eine Linie
f	gehe um eine Längeneinheit vor, ohne zu zeichnen
G	gehe um eine Längeneinheit vor und zeichne einen Zylinder
+	drehe um den Drehwinkel nach links
-	drehe um den Drehwinkel nach rechts
&	drehe um den Drehwinkel nach unten
^	drehe um den Drehwinkel nach oben
\	rolle um den Drehwinkel nach links
/	rolle um den Drehwinkel nach rechts
|	drehe die Laufrichtung um 180°
[	merke die aktuelle Position
]	springe zur letzten gemerkten Position
!	verkleinere den Drehwinkel um das Drehwinkeldekrement
#	vergrößere den Drehwinkel um das Drehwinkelinkrement
{	beginne ein Polygon
.	merke Punkt als Polygonpunkt
}	schließe das Polygon
'	erniedrige Farbindex
`	erhöhe Farbindex

Anhang B

Installationsanweisung

Um das Programm `fractals` zur interaktiven Erzeugung fraktaler Landschaften zu installieren, sind nach dem Auspacken des Tar-Files folgende Schritte durchzuführen:

1. Im Verzeichnis `$HOME/bin` einen symbolischen Link setzen:
   `ln -s /usr/local/transp/parix-1.3/bin/px ppx`.
2. In den Verzeichnissen `Parallel` und `Sequen` das Programm `gmake` aufrufen.
3. Im Verzeichnis `Fraktale` mit `xmkmf` ein Makefile generieren und mit `gmake` ausführen.
4. Unter dem Heimatverzeichnis ein Verzeichnis `Bin` anlegen und aus dem Verzeichnis `Fraktale/Bin` die Dateien `ypx`, `yppx`, `startup.rsh` und `linkfree` dorthin kopieren.
5. Die Datei `startup_profile` aus dem Verzeichnis `Fraktale/Bin` in die Datei `.startup_profile` im Heimatverzeichnis kopieren.
6. In der Datei `.startup_profile` im Heimatverzeichnis die Umgebunsvariablen setzen für

   `FRAKTALE_DATA` gibt an, wo das Verzeichnis `Fraktale` liegt,
   z.B. `$HOME/Fraktale`

   `FRAKTALE_SEQ` gibt an, wo das Verzeichnis `Fraktale/Sequen` liegt,
   z.B. `$HOME/Fraktale/Sequen`

   `FRAKTALE_PAR` gibt an, wo das Verzeichnis `Fraktale/Parallel` liegt,
   z.B. `$HOME/Fraktale/Parallel`
7. Aufruf des Programms durch Eingabe von `fractals` im Verzeichnis `Fraktale` . Eine Bedienungsanleitung erhält man, wenn man in der Menüleiste der Applikation den Menüpunkt `Hilfe` anwählt.

Anhang C

Beispiel für die Szenenbeschreibung

In diesem Kapitel werden zur Veranschaulichung Beispieldateien für die Beschreibung einer Szenen-, einer Berg-, einer Baum- und einer Darstellungsdatei dargestellt.

Beispiel für eine Szenendatei: `std_szene.dat`

```
Berg: std_berg.dat
Darstellung: std_darstellung.dat
Baum: std_baum.dat
```

Beispiel für eine Bergdatei: `std_berg.dat`

```
Anzahl Iterationen: 3
Bergfaktor: 6
Schrumpffaktor fuer Bergfaktor: 0.800000
Interpolationsgewicht Aussenseite: 0.500000
Ausdehnung in x- bzw. z-Richtung: 1000
Niedrigster Farbwert: 7
Hoechster Farbwert: 32
```

Beispiel für eine Baumdatei: `std_baum.dat`

```
derivation length: 6
angle factor: 30.000000
angle increment: 0.000000
angle decrement: 0.000000
scale factor: 3.400000
initial position: 400.000000 0.000000 1000.000000
axiom: T
ignore: *
*  <  T  >  * --> GGGGGGG[^^GGGBFB][^^GGBFB][++GGGB][+GGGBFB][B][GGGB][-GGGBFB]
*  <  B  >  * --> (.30)[+L][L][-L],(.30)[^L][L][&L],(.40)[+fB][-fB][^fB],
*  <  L  >  * --> {-f'.+f.+f'.-|-f.+f`.`}
```

Beispiel für eine Darstellungsdatei: `std_darstellung.dat`

```
ViewReferencePoint.x: 500.000000
ViewReferencePoint.y: 0.000000
ViewReferencePoint.z: 600.000000
ProjectionReferencePoint.x: 500.000000
ProjectionReferencePoint.y: 400.000000
ProjectionReferencePoint.z: 1500.000000
ViewAngle: 0.000000
XMin ViewWindow: -500.000000
XMax ViewWindow: 500.000000
YMin ViewWindow: -250.000000
YMax ViewWindow: 650.000000
Light.x: 680.000000
Light.y: 70.000000
Light.z: 450.000000
Intensitaet Lichtquelle: 1.000000
Diffuser Reflexionskoeffizient: 0.950000
Intensitaet Hintergrund: 1.000000
Ambienter Reflexionskoeffizient: 0.950000
Shading-Modus: G
```

Anhang D

Die C-Quellen, Ressource-Dateien und Shell-Skripte

In diesem Kapitel werden die C-Quellen, Ressource-Dateien und Shell-Skripte für das Programm `fractals` aufgelistet und kurz beschrieben.

D.1 C-Quellen

D.1.1 Die C-Quellen des sequentiellen Programms

Dieser Unterabschnitt enthält eine Aufzählung der C-Quellen, die für die sequentielle Version des Programms zur Erzeugung eines fraktalen Bildes benötigt werden.

`berg.c`	Erzeugt mit Hilfe einer Bergdatei Daten für einen Berg.
`generate.c`	Erzeugt den Graphik-String aus einem L-System.
`interpret.c`	Generiert Daten für die Pflanzen durch graphische Interpretation des Strings mit Hilfe einer Turtle-Graphik.
`render.c`	Liest die zu verarbeitenden Daten ein und ruft Funktionen für ihre Weiterverarbeitung auf.
`proj_math.c`	Enthält verschiedene Funktionen zum Umgang mit Matrizen und Vektoren.
`proj_trans.c`	Enthält Funktionen zur Erstellung von Transformationsmatrizen.
`proj_view.c`	Berechnet die Matrizen für die Viewing-Pipeline; enthält Funktion für das Back Face Culling und für die Weiterverarbeitung der Objektdaten.
`proj_clip.c`	Enthält Funktionen für das Clipping von Linien und Polygonen.
`proj_shade.c`	Shadet Dreiecke und Linien; gibt Frame-Buffer als PPM-File in Datei aus.

D.1.2 Die C-Quellen des parallelen Programms

In diesem Unterabschnitt werden die C-Quellen für das parallele Programm und das Postprocessing vorgestellt, und ihre Funktion wird beschrieben.

`start_transp.c`	Hauptprogramm, das auf alle Transputer geladen wird; baut die globale Topologie auf; startet Teilprogramme anhand von Prozessor-IDs nach.
`start_expl.c`	Hauptprogramm, das auf die Prozessoren des PowerXplorers geladen wird; baut die globale Topologie auf; startet Teilprogramme anhand von Prozessor-IDs nach.
`a_berg.c`	Erzeugt Daten für einen Berg; wendet auf die Daten Viewing-Pipeline mit Back Face Culling und Clipping an; verschickt die Ergebnisdaten asynchron an entsprechende Darstellungsprozessoren.
`a_generate.c`	Erzeugt einen Graphikstring aus einem L-System; asynchrone Kommunikation, um Stop-Zeichen zu verschicken.
`a_interpret.c`	Interpretiert String mit Turtle-Graphik; wendet auf entstehende Daten Viewing-Pipeline mit Back Face Culling und Clipping an; verschickt Ergebnisdaten asynchron an Darstellungsprozessoren.
`berg.c`	Erzeugt Daten für einen Berg; wendet auf Daten Viewing-Pipeline mit Back Face Culling und Clipping an; verschickt die Ergebnisdaten synchron an entsprechende Darstellungsprozessoren.
`generate.c`	Erzeugt einen Graphikstring aus einem L-System; synchrone Kommunikation, um Stop-Zeichen zu verschicken.
`interpret.c`	Interpretiert String mit Turtle-Graphik; wendet auf entstehende Daten Viewing-Pipeline mit Clipping und Back Face Culling an; verschickt Ergebnisdaten synchron an Darstellungsprozessoren.
`render.c`	Empfängt Daten von Pflanzen- und Bergprozessoren und ruft Funktionen für ihre Weiterverarbeitung auf.
`proj_math.c`	Enthält verschiedene Funktionen zum Umgang mit Matrizen und Vektoren.
`proj_trans.c`	Enthält Funktionen zur Erstellung von Transformationsmatrizen.
`proj_view.c`	Berechnet die Matrizen für die Viewing-Pipeline; ruft Funktionen zur Weiterverarbeitung der Objektdaten auf.
`proj_clip.c`	Enthält Funktionen für das Clipping von Linien und Polygonen.
`proj_shade.c`	Shadet Dreiecke und Linien; gibt Frame Buffer in Datei aus.
`convert.c`	Fügt die von den Transputern generierten Dateien zu einem PPM-File zusammen.

D.1.3 Die C-Quellen für die Oberfläche

Dieser Abschnitt beschreibt alle C-Quellen der Oberfläche.

`fraktale.c`	Hauptprogramm der Oberfläche; initialisiert Graphikvariablen und startet Eventschleife.
`initialize.c`	Initialisiert globale Variablen und Dateien der Standardszene.
`register.c`	Enthält Funktion zum Registrieren der Callbacks.
`generalCB.c`	Enthält allgemeine Callbacks.
`mountCB.c`	Enthält Callbacks für die Bearbeitung der Bergdateien.
`treeCB.c`	Enthält Callbacks für die Bearbeitung der Pflanzendateien (L-Systeme).
`renderCB.c`	Enthält Callbacks für die Bearbeitung der Darstellungsdateien.
`sceneCB.c`	Enthält Callbacks für das Arbeiten mit Szenedateien.
`general_util.c`	Funktionen, die von allgemeinen Callbacks aufgerufen werden.
`mount_util.c`	Funktionen, die von Callbacks zur Bergbearbeitung aufgerufen werden.
`tree_util.c`	Funktionen, die von Callbacks zur Pflanzenbearbeitung aufgerufen werden.
`render_util.c`	Funktionen, die von Callbacks zur Darstellungsbearbeitung aufgerufen werden.
`scene_util.c`	Funktionen, die von Callbacks zur Szenenbearbeitung aufgerufen werden.
`help.c`	Funktion, die Hilfetexte für das Hilfefenster richtig setzt.
`start.c`	Funktion zum Starten des parallelen Programms auf den Transputern und zur Übergabe der Szenedaten an die Prozessoren.
`start_pow.c`	Funktion zum Starten des parallelen Programms auf dem PowerXplorer und zur Übergabe der Szenedaten an die Prozessoren.
`xwud.c`	Umgeschriebenes `xwud.c`, das X Window Dump nach Berechnung im Bildfenster der Oberfläche darstellt.

D.2 Shell-Skripte

Zum einfacheren Arbeiten mit der Oberfläche existieren drei Shell-Skripte, die kurz erläutert werden sollen.

`fractals`	Shell-Skript zum Starten der Oberfläche; setzt Environment-Variablen für die Pfade zu einigen Verzeichnissen.
`renseq`	ruft sequentielles Programm zur Verarbeitung der Berg- und Pflanzendaten durch den Darstellungsalgorithmus auf; führt Postprocessing mit Ergebnisdatei durch.
`renpar`	ruft Programm `convert` zum Aufbau eines PPM-Files aus den Ergebnisdateien des parallelen Programms auf; führt Postprocessing für das entstandene PPM-File durch.

D.3 Ressource-Dateien

Im folgenden werden alle Wcl-Ressource-Dateien, die zur Erzeugung der Oberfläche entwickelt wurden, aufgelistet und kurz erläutert.

`fraktale.res`	Enthält Ressourcen für das Hauptfenster mit Menüleiste, Bildfenster und rechtem Menüfeld, für das Hilfefenster und den Dialog für Fehlermeldungen.
`bergmenu.res`	Beschreibt das Menü zur Manipulation der Werte einer Bergdatei.
`baummenu.res`	Beschreibt das Menü zur Manipulation der Werte einer Pflanzendatei, d.h. eines L-Systems.
`darstellmenu.res`	Beschreibt das Menü zur Manipulation der Werte einer Darstellungsdatei.
`waehlebaum.res`	Beschreibt Selectionbox zur Auswahl einer Pflanzendatei aus allen Pflanzendateien der aktuellen Szene.
`szenemenu.res`	Beschreibt das Popup, in dem die Dateien einer Szene aufgelistet werden und ausgetauscht werden können.
`schowberg.res`	Beschreibt das Popup, das zur Darstellung der Werte einer Bergdatei dient.
`showbaum.res`	Beschreibt das Popup, das zur Darstellung der Werte einer Pflanzendatei dient.
`showdarstell.res`	Beschreibt das Popup, das zur Auflistung der Werte einer Darstellungsdatei dient.
`startmenu.res`	Beschreibt die Menüs zur Auswahl der Abarbeitungsart und zum Starten des parallelen bzw. des sequentiellen Programms.
`speicherMenu.res`	Enthält Ressourcen für den Dialog zur Abfrage, ob gespeichert werden soll, und zur Eingabe des Dateinamens.
`speicherpopup.res`	Enthält Ressourcen für ein Popup zur Abfrage eines Dateinamens.
`koordinaten.tpl`	Template-Datei; enthält Ressourcen für ein Widget zur Manipulation von 3D-Koordinaten mit Arrow-Buttons; wird mehrmals verwendet.
`ctrl.tpl`	Template-Datei; enthält Ressourcen für ein Widget mit OK-, `Abbruch`- und `Hilfe`-Knopf; wird mehrmals verwendet.

Literaturverzeichnis

[BrJo91] Hans Joachim Brede, Nicolai Josuttis et al. *Programmieren mit OSF/Motif*. Addison-Wesley Publishing Company, Bonn, München, 1991

[Fel92] Wolf Dietrich Fellner. *Computergrafik*. 2., vollständig überarbeitete und erweiterte Auflage. BI-Wissenschaftsverlag Band 58, Mannheim, 1992

[Fer93] Paula M. Ferguson. *Motif Reference Manual*. O'Reilly & Associates, Inc., USA, 1993

[Hec82] Paul Heckbert. *Color Image Quantization for Frame Buffer Display*. SIGGRAPH '82 Proceedings, 1982

[Hel91] Dan Heller. *Motif Programming Manual*. O'Reilly & Associates, Inc., USA, 1991

[La1/92] Hans Lauwerier. *Fraktale verstehen und selbst programmieren, Band 1, Einführung*. 2., verbesserte Auflage. R. Wittig Fachbuchverlag, Hückelhoven, 1992

[La2/92] Hans Lauwerier. *Fraktale verstehen und selbst programmieren, Band 2, Vertiefung*. R. Wittig Fachbuchverlag, Hückelhoven, 1992

[Man87] Benoît B. Mandelbrot. *Die fraktale Geometrie der Natur*. Birkhäuser Verlag, Basel, 1987

[PaRe92] Parsytec Computer GmbH. *Parix Reference Manual*. Release 1.2. Parsytec GmbH, Aachen, 1992 - 1993

[PaSo92] Parsytec Computer GmbH. *Parix Software Documentation*. Release 1.2. Parsytec GmbH, Aachen, 1992 - 1993

[Pei88] Heinz-Otto Peitgen, Dietmar Saupe (Hrg.). *The Science of Fractal Images*. Springer-Verlag, New York, 1988

[PrHa89] Przemyslaw Prusinkiewicz, James Hanan. *Lindenmayer Systems, Fractals, and Plants*. Lecture Notes in Biomathematics 79. Springer-Verlag, New York, 1989

[PrLi90] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990

[Smy90] David E. Smyth. *Wcl – An Easier Way to Develop Widget Based Applications*. Pasadena, 1990

[Smy91] David E. Smyth. *Wcl – A Thin Veneer Over Xrm*. Pasadena, 1991

[Vor94] Oliver Vornberger. *Computergrafik*. 2., überarbeitete Auflage. Osnabrücker Schriften zur Mathematik, Heft 118, Osnabrück, 1994

[XlPr92] Adrian Nye. *Xlib Programming Manual*. Third Edition. O'Reilly & Associates, Inc., USA, 1992

[XlRe92] Adrian Nye. *Xlib Reference Manual*. Third Edition. O'Reilly & Associates, Inc., USA, 1992

[XtPr92] Adrian Nye, Tim O'Reilly. *X Toolkit Intrinsics Programming Manual*. O'Reilly & Associates, Inc., USA, 1992

[XtRe92] David Flanagan. *X Toolkit Intrinsics Reference Manual*. Third Edition. O'Reilly & Associates, Inc., USA, 1992