Inhaltsverzeichnis

1 Einleitung

1.1 Was ist es?

selftree.inc ist ein in PoV geschriebener Codegenerator für PoV. Der erzeugte Code wiederum kann in Scenefiles eingebunden werden und erzeugt dort baumähnliche Objekte. Kurz gesagt, kann selftree.inc also dazu benutzt werden Bäume zu erzeugen. Dabei ist das Aussehen der Bäume durch eine Vielzahl von Parametern beeinflußbar, von selbstdefinierten Blättern bis hin zu verschiedenen Texturen für verschiedene Äste, deren Zahl und Neigungswinkel ebenso bestimmbar sind, ist fast alles möglich. Insgesammt beläuft sich die Anzahl der einstellbaren Werte auf mindestens 35, wobei aber für jeden Wert eine Defaulteinstellung vorhanden ist, so daß nur die gewünschten Parameter verändert werden müssen. Die einzelnen Parameter, ihre Bedeutung und ihre Wertebereiche werden weiter unten einzeln beschrieben.

Die folgende Anleitung muß nicht komplett gelesen werden, um mit dem File umzugehen. Zum Gebrauch wichtig sind nur die Abschnitte 2 (einschließlich Unterabschnitte), 3 und vielleicht 5. Der Rest ist optional und nur interessant für ein genaueres Verständnis der Funktionsweise.

1.2 Idee, Motivation und wozu das ganze?

Die Idee ist ursprünglich geklaut von einem Mathematikstudenten (Hallo, Holger!), der mit einigen ineinander verschachtelten Include-Files ein paar recht beeindruckende Bäume erzeugt hat. Seine Methode hatte jedoch den Nachteil, daß er die Include-Files von Hand erzeugen mußte, und daß er mangels Rekursion in PoV mehrfach verschachtelt includen mußte und somit wegen einer Einschränkung in PoV auf höchstens Rekursionstiefe 10 beschränkt war (nicht daß man für vernünftige Bäume tiefer rekursieren müßte, aber dennoch...).

Zuerst löste ich das Problem, indem ich ein tcl-Script schrieb, das Bäume in voll ausgeschriebenem PoV-Code erzeugte. Das funktionierte zwar einwandfrei, hatte aber den Nachteil, daß die erzeugten Include-Files extrem lang waren (includes von 10 MB waren nicht ungewöhnlich...) und außerdem nicht von den Sprachfähigkeiten von PoV Gebrauch machten. Das in PoV geschriebene Include-File löst beide Probleme. Die erzeugten Includes sind recht kurz (meist zwischen 10 und 30 kb), **weil** sie die Sprachfähigkeiten von PoV nutzen und durch ineinander verschachtelte While-Schleifen endrekursiven Code erzeugen, der dann erst beim Parsen durch PoV aufgefaltet wird. Das ganze löst zudem ein weiteres Problem. Denn obwohl tcl eine weit verbreitete Interpretersprache ist, ist das Include-File portabler als die erste Lösung.

Na ja, und wozu das ganze nun? Um Bäume zu erzeugen, die sich in manchen Bildern sicher sehr gut machen, per Hand aber einfach zu aufwendig zu erzeugen sind.

2 Gebrauchsanleitung

selftree.inc wird in ein normales PoV-Sourcefile eingebunden. Die Parameter, die verändert werden sollen, müssen vor der Include-Anweisung mit

```
#declare VARNAME = VALUE
```

gesetzt werden, wobei VARNAME der Name des zu modifizierenden Parameters ist (Groß- und Kleinschreibung werden unterschieden!), und VALUE der Wert.

Nach der Include-Anweisung parsiert PoV selftree.inc wie einen Teil des normalen Sourcecodes, und gibt dabei den von selftree.inc erzeugten Sourcecode als Debugging-Meldungen aus. Diese Debugging-Meldungen müssen mit der Kommandozeilen-Option **+GD** in eine weitere Source-Datei umgeleitet werden, z.B. tree.inc.

```
Aufruf: povray +ISOURCEFILE +GDtree.inc
```

Das damit erzeugte File muß nun wiederum in den eigentlichen Source included sein.

Es ist natürlich auch möglich, erst mehrere Baumfiles zu erzeugen, und diese dann später in einem weiteren Source erst einzubinden. Man kann aber auch mehrfach das File selftree.inc einbinden und die erzeugten Bäume direkt einbinden, also in etwa eine Folge aus

```
// Variablen setzen
#declare VARNAME = VALUE
...
#include ''selftree.inc''
#include ''tree.inc''
```

```
// Variablen setzen
#declare VARNAME = VALUE
...
#include ``selftree.inc''
#include ``tree.inc''
// Variablen setzen
#declare VARNAME = VALUE
...
#include ``selftree.inc''
#include ``tree.inc''
.
.
.
```

2.1 Parameter

Im folgenden werden alle einstellbaren Parameter, ihre Bedeutung und ihr möglicher Wertebereich aufgelistet. Für alle Werte gibt es einen in selftree.inc eingestellten Default-Wert, so daß nur die Variablen verändert werden müssen, die nicht den Default aufweisen sollen. Wenn selftree.inc mehrfach in einem Source eingebunden wird, bleiben natürlich alle früheren declare-Anweisungen weiterhin gültig, so daß die Default-Werte dauerhaft überschrieben sind.

2.1.1 TRUNKLEN

Beschreibung: Die Länge des Stammes. Der Stamm jeder Baumunion beginnt am Punkt $< 0, 0, 0 >$, und endet am Punkt $< 0, TRUNKLEN, 0 >$.
Wertebereich: FLOAT und $TRUNKLEN > 0$
Defaultwert: 5

2.1.2 TRUNKBASER

Beschreibung: Der Radius des Stammes am Punkt $< 0, 0, 0 >$.
Wertebereich: FLOAT und $TRUNKBASER > 0$
Defaultwert: 1

2.1.3 TRUNKCAPR

Beschreibung: Der Radius des Stammes am Punkt $< 0, TRUNKLEN, 0 >$.
Wertebereich: FLOAT und $TRUNKCAPR > 0$
Defaultwert: 0.5

2.1.4 DEPTH

Beschreibung: Die Rekursionsstiefe des Baumes. Dieser Parameter beeinflußt im wesentlichen die Größe des Baumes. Es werden für die Äste des Baumes DEPTH geschachtelte Unions erzeugt. Die meisten der folgenden Werte werden in irgendeiner Weise von dem beim Parsen des erzeugten Codes aktuellen Wert von DEPTH beeinflußt.
Wertebereich: INT und $DEPTH > 0$
Defaultwert: 4

2.1.5 BLEN

Beschreibung: Die Länge der Äste in der ersten Rekursionsstufe (diejenigen, die direkt am Stamm ansetzen). Die jeweilige Länge der Äste in tieferen Rekursionsstufen wird durch Berechnung der Gleichung $LEN = (BLEN + (rand(RANDLDIFF) * LDIFF)) * RECSCALE^{REKURSIONSSTUFE}$ ermittelt. Dabei ist $rand(RANDLDIFF) * LDIFF$ ein Zufallswert, der zwischen 0 und LDIFF liegt. $RECSCALE^{REKURSIONSSTUFE}$ ist der eigentliche Skalierungswert, mit dem die Länge der Äste für die unteren Rekursionsstufen berechnet wird.
Wertebereich: FLOAT und $BLEN > 0$
Defaultwert: 5

2.1.6 LDIFF

Beschreibung: Wie oben beschrieben ein Wert, der die maximale Abweichung in der Länge von Ästen beschreibt. Der Wert wird in den Rekursionsstufen mitskaliert und wird als Zufallswert zwischen 0 und LDIFF zur LÄnge des Astes addiert.
Wertebereich: FLOAT, $|LDIFF| < BLEN$
Defaultwert: 0

2.1.7 RECSCALE

Beschreibung: Der Skalierungswert für die Astlänge zwischen den Rekursionsstufen.
Wertebereich: FLOAT und $RECSCALE > 0$
Defaultwert: 0.5

2.1.8 RADIUS

Beschreibung: Der Ausgangsradius der Äste. Der wirklich Radius eines Astes wird mit Hilfe dieses Wertes nach folgender Formel berechnet: $(RADIUS - (rand(RANDRDIFF)) * RDIFF) * RADSCALE^{REKURSIONSSTUFE}$. Dabei

ist wie zuvor $RADSCALE^{REKURSIONSSTUFE}$ der Skalierungswert in Abhängigkeit von der aktuellen Rekursionsstufe, und $rand(RANDRDIFF)) * RDIFF$ ein Zufallswert zwischen 0 und RDIFF.
Wertebereich: FLOAT und $RADIUS > 0$
Defaultwert: 0.5

2.1.9 RDIFF

Beschreibung: Wie LDIFF, nur für den Wert RADIUS und mit dem Unterschied, daß der Zufallswert von RADIUS abgezogen wird (LDIFF wird addiert).
Wertebereich: FLOAT und $RDIFF > -RADIUS$
Defaultwert: 0

2.1.10 RADSCALE

Beschreibung: Der Skalierungswert für den RADIUS, wird in Abhängigkeit von der aktuellen Rekursionsstufe berechnet.
Wertebereich: FLOAT und $RADSCALE > 0$
Defaultwert: 0.5

2.1.11 SLOPEANGLE

Beschreibung: Der Neigungswert der Äste gegenüber ihren Vaterästen. Die eigentliche Neigung eines Astes wird folgendermaßen berechnet: $(SLOPEANGLE + rand(RANDSDIFF)*SANGLEDIFF)*ANGLESCALE^{REKURSIONSSTUFE}$, wobei $rand(RANDSDIFF) * SANGLEDIFF$ wie üblich ein Zufallswert zwischen 0 und SANGLEDIFF ist und das ganze im letztem Teil mit einem Skalierungswert multipliziert wird.
Wertebereich: FLOAT, keine Einschränkung
Defaultwert: 45

2.1.12 SANGLEDIFF

Beschreibung: Der Wert für die maximale Abweichung von SLOPEANGLE, wird als Zufallswert $0 <= x <= SANGLEDIFFzuSLOPEANGLE$ addiert.
Wertebereich: FLOAT, keine Einschränkung
Defaultwert: 0

2.1.13 ANGLESCALE

Beschreibung: Der Skalierungswert für SLOPEANGLE.
Wertebereich: FLOAT, keine Einschränkung
Defaultwert: 1

2.1.14 NRBRANCHES

Beschreibung: Die Anzahl der Äste, die von einem darunterliegendem Ast, bzw. vom Stamm abzweigen. Der genaue Wert für jeden Schleifendurchgang wird wie folgt berechnet: $ANZAHL = (NRBRANCHES + (rand(RANDNRBDIFF)*NRBDIFF))*NRBSCALEMUL^{REKURSIONSSTUFE}+(NRBSCALEADD * REKURSIONSSTUFE)$, wobei $rand(RANDNRBDIFF) * NRBDIFF$ wie üblich die Abweichung bestimmt.

$NRBSCALEMUL^{REKURSIONSSTUFE}$ ist ein Skalierungswert, mit dem NRBRANCHES multipliziert wird, $(NRBSCALEADD * REKURSIONSSTUFE)$ ist ein Skalierungswert, der zu NRBRANCHES addiert wird. Dadurch stehen zwei Methoden zur Verringerung bzw. Vermehrung der Anzahl Äste in Abhängigkeit von der Rekursionsstufe zur Verfügung.
Wertebereich: FLOAT und $NRBRANCHES > 0$
Defaultwert: 4

2.1.15 NRBDIFF

Beschreibung: Der Wert für die maximale Abweichung von NRBRANCHES. Wird wie aus der Gleichung oben ersichtlich zu NRBRANCHES addiert.
Wertebereich: FLOAT und $NRBDIFF > -NRBRANCHES$
Defaultwert: 0

2.1.16 NRBSCALEADD

Beschreibung: Der addierte Skalierungswert für NRBRANCHES (siehe NRBRANCHES)
Wertebereich: FLOAT und $NRBSCALEADD > -NRBRANCHES$
Defaultwert: 0

2.1.17 NRBSCALEMUL

Beschreibung: Der multiplizierte Skalierungswert für NRBRANCHES (siehe NRBRANCHES)
Wertebereich: FLOAT und $|NRBSCALEMUL| > 0$
Defaultwert: 1

2.1.18 BRANCHTOP

Beschreibung: Ein Wahrscheinlichkeitswert, der angibt, mit welcher Wahrscheinlichkeit ein Ast von der Spitze seines Vorgängers abzweigt. Es wird beim Parsieren ein Zufallswert zwischen null und eins erzeugt, ist BRANCHTOP größer als der erzeugte Wert, zweigt der Ast von der Spitze seines Vorgängers ab, ansonsten irgendwo zwischen dessen Ursprung und der Spitze einschließlich.

Wertebereich: FLOAT und $0 <= BRANCHTOP <= 1$
Defaultwert: 1

2.1.19 LEAFINC

Beschreibung: Ein Bool, der bestimmt, ob der Baum Blätter haben soll, oder nicht.[1]
Wertebereich: YES|NO
Defaultwert: NO

2.1.20 NRLEAFINCS

Beschreibung: Gibt die Zahl der verschiedenen Blätterdefines an.
Wertebereich: INT und $NRLEAFINCS >= 0$
Defaultwert: 0

2.1.21 NRLEAFS

Beschreibung: Gibt die Anzahl der Blätter pro Ast an.
Wertebereich: INT und $NRLEAFS >= 0$
Defaultwert: 0

2.1.22 NRLDIFF

Beschreibung: Wie alle Werte mit "DIFF" im Namen ein Wert, der beim Parsen zu einem Wert zwischen 0 und NRLDIFF wird und dann durch Addition zu NRLEAFS den wirklichen Wert für die Anzahl Blätter erzeugt.
Wertebereich: INT und $NRLDIFF > -NRLEAFS$
Defaultwert: 0

2.1.23 ROTRANGE

Beschreibung: Ein Bereichswert, der angibt, um welchen Winkelbereich ein Ast um seinen Vaterast rotiert werden kann. Normalerweise ist eine volle Rotation erlaubt, so daß Äste gleichmäßig um ihren Vaterast verteilt sind. Eine Einschränkung des Wertes ergibt eine wenn auch recht magere Methode Windausrichtung zu erzeugen.
Wertebereich: FLOAT und $0 <= ROTRANGE <= 360$
Defaultwert: 360

2.1.24 RANDINIT_ROT

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für ROTRANGE.

[1] siehe Kapitel "Blätter" zur genaueren Beschreibung.

Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.25 RANDINIT_LDIFF

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für LDIFF.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.26 RANDINIT_RDIFF

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für RDIFF.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.27 RANDINIT_SDIFF

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für SANGLEDIFF.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.28 RANDINIT_NRBDIFF

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für NRBDIFF.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.29 RANDINIT_NRLDIFF

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für NRLDIFF.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.30 RANDINIT_LEAFSELECT

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für ROTRANGE.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.31 RANDINIT_LEAFTRANS

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für ROTRANGE.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.32 RANDINIT_LEAFROT

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für den Zufallsgenerator, der Blätter aus der Liste der Blätterincludes wählt.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.33 RANDINIT_BT

Beschreibung: Initialisierungswert für den Zufallsgenerator für die Zufallswerte für BRANCHTOP.
Wertebereich: FLOAT
Defaultwert: 0.1234

2.1.34 MOD_TR

Beschreibung: Texture-Modifier für den Stamm. Es muß eine vollständige Textur-Definition angegeben werden. Beispiel:

```
#declare MOD_TR = texture { pigment { color Green } }
```

Wertebereich: texture-Definition
Defaultwert: pigment { color rgb $< 0.647059, 0.164706, 0.164706 >$ }

2.1.35 MOD_x

Beschreibung: Texture-Modifier für Äste. Die Namenskonvention für Asttexturen lautet: MOD_x, wobei x ein Integerwert ist, der die Rekursionsstufe angibt, für deren Äste die Textur benutzt werden soll. Die Zählung beginnt bei 0 und läuft bis DEPTH-1. Es können beliebig viele Definitionen angegeben werden. Für Rekursionsstufen, für die keine Textur angegeben ist, wird die Default-Textur verwendet. Beispiel:

```
#declare MOD_0 = texture { pigment { color Green } }
#declare MOD_1 = texture { pigment { color Blue } }
#declare MOD_2 = texture { pigment { color Yellow } }
#declare MOD_12 = texture { pigment { color Red } }
```

Wird nun ein Baum mit DEPTH=13 erzeugt, so haben alle Äste in der Rekursionsstufe 0 bis 3 und 12 die angegebene Textur, alle anderen die Defaulttextur.
Wertebereich: texture-Definition
Defaultwert: pigment { color rgb $< 0.647059, 0.164706, 0.164706 >$ }

2.2 Anmerkungen zu einigen Werten

Die Variablen sind, wie man merkt zum Teil in "Hauptvariablen" und "Nebenvariablen" eingeteilt, wobei letztere, die genaue Wirkung der ersteren beeinflussen. Vermutlich wird man am ehesten durch Herumprobieren gute Werte für die Variablen finden. Wenn man Bäume erzeugen möchte, die eher natürlich wirken, sind vielleicht folgende Tips nützlich:

- TRUNKxxx: Egal, welche Werte man angibt, ein Stamm ist in dieser Version immer ein einfacher gerader Konus. Um bessere Ergebnisse zu erzielen, ist es manchmal am besten, das erzeugte Include-File selber zu edieren, und einen eigenen Stamm einzusetzen. Die Includes werden so erzeugt, daß der Stamm leicht zu entfernen und durch einen eigenen zu ersetzen ist.
- RADIUS: Wenn die untersten Äste möglichst nahtlos am Stamm ansetzen sollen, sollte dieser Wert am besten gleich TRUNKCAPR sein.
- RANDINIT_xxx: Warum so viele RANDOM-Streams? Die große Zahl einzelner Random-Streams erlaubt, die Zufallsfolge für eine einzelne Baumkomponente zu verändern (z.B. nur alle Neigungswinkel), ohne die anderen Teile des Baumes zu beeinflussen.

3 Blätter

Soll ein Baum mit Blättern erzeugt werden, so müssen diese vorher definiert werden. Es gibt kein Defaultlaub, es existiert nur die Möglichkeit, vorgegebene Blätter an den Baum zu haften. Blätter werden folgendermaßen definiert:

```
#declare LEAFINC_0 =
sphere
{
  <0, 0, 0>, 0.3
  scale <1, 0.1, 0.5>
  translate <0.3, 0, 0>
  pigment { color Red }
}
```

Namen: Die Blätter müssen immer den Namen LEAFINC_X haben, wobei X für eine Integerzahl steht, die von dem Wert NRLEAFINCS abhängt. Ist NRLEAFINCS z.B. 15, so müssen LEAFINC_0 ... LEAFINC_9, LEAFINC_10 ...

LEAFINC_14 existieren. Die beispielhafte Definition muß natürlich nicht unbedingt eine Sphere enthalten, sondern darf ein beliebig komplexes Object werden, einschließlich Textur und allem drum und dran...

Blätter (oder andere Objekte, die an den Baum sollen, wie z.B. Früchte oder Weihnachtskugeln...), sollten so definiert werden, daß ihr Ursprung mit dem Punkt $< 0, 0, 0 >$ zusammenfällt. Alle Blätterincludes werden mit diesem Punkt an den Baum geheftet und zufällig um den jeweiligen Zweig rotiert. Leider ist es mangels eines Stacks, der sich die Position eines Astes im Raum merkt, nicht möglich, die Blätter sinnvoll auszurichten. Daher stehen alle Blätter senkrecht von ihrem Vaterzweig ab.

4 Funktionsweise

Da es in PoV nicht möglich ist, Felder oder Funktionsaufrufe (dieses Problem wird sich unter Umständen bald erledigen, wenn es Makros gibt) im Code zu haben, muß für einen Baum ein endrekursiver Sourcecode erzeugt werden. Normalerweise hat PoV keine Möglichkeit, selber Dateien zu erzeugen, außer die Bilder, die berechnet weden und verschiedene statische Ausgaben. Um nun doch von PoV selber einen Sourcecode erzeugen zu lassen, muß daher eine für einen anderen Zweck gedachte Möglichkeit gebraucht werden.

Mit dem debug-Statement ist es möglich, Meldungen über den Sourcecode und den Zustand verschiedener Variablen zu Zwecken der Fehlersuche auf der Console ausgeben zu lassen. Diese Meldungen können wiederum mit dem Kommandozeilen-Argument **+GD** in eine beliebige Datei umgeleitet werden (sie erscheinen dann allerdings trotzdem noch auf der Console, was nicht verwirren sollte). selftree.inc nutzt nun die Möglichkeiten der Stringverarbeitung und anderer Funktionen in PoV aus, um mit diesen Debug-Meldungen einen Sourcecode zu erzeugen, der wiederum selber von PoV parsierbar ist. Dieser Code in eine Datei umgeleitet ergibt dann den gewünschten endrekursiven Baumsource, der nur noch mit einer Include-Anweisung eingebunden werden muß.

Dieses Verfahren ist natürlich noch mehrstufig verwendbar. Theoretisch ist es möglich, mehrere Sourcecode-Erzeuger (das ist es, was selftree.inc eigentlich ist) weitere Sourcecode-Erzeuger produzieren zu lassen, die immer komplexere Operationen durchführen, die in der jeweils vorhergehenden Stufe nicht durchgeführt werden können. selftree.inc nutzt nur eine dieser Stufen aus.

Die Bäume, die von dem von selftree.inc erzeugten Include-File erzeugt werden, bestehen aus hierarchisch angeordneten Konussen. Alle Konusse sind, um glattere Kanten zu haben, mit Kugeln gleichen Durchmessers abgeschlossen. Nur die Konusse in der untersten Rekrusionsstufe (die Astspitzen) haben keine abschließenden Kugeln, um die Anzahl der Objekte, und damit dem Speicherbedarf der Bäume, zu reduzieren.

5 Beispiel

Die von selftree.inc erzeugten Files sind zwar kommentiert, um das Verständnis der erzeugten Files zu erleichtern ist im folgenden eine ausführlich kommentierte Version des Files zu sehen, die erzeugt wird, wenn nur die Default-Werte benutzt werden (Zeilen, die mit einem \ enden, sind absichtlich umgebrochen, um LaTeXden Satz zu erleichtern, in PoV dürfen diese Zeichen so nicht vorkommen.):

/* Union öffnen: Der Baum besteht aus einer einzigen Union, um ihn als Objekt einbinden zu können, die üblichen Objektmodifier und Transformationen anwenden zu können. */

```
union
{
```

/* Baumstamm erzeugen: Der Baumstamm ist ein eigenes Objekt, um die Parameter für die Äste zu vereinfachen. Dadurch ist es weiterhin möglich, spezielle Werte für den Stamm anzugeben. Der Wert BRANCHTOP ist aber dennoch auch für den Stamm gültig, so daß die untersten Äste ebenfalls nicht unbedingt an der Stammspitze beginnen müssen. Baumstämme bestehen aus einem Konus und einer abschließenden Kugel am oberen Ende. */

```
    union
    {
        cone , 0,{<0, 0>, TRUNKBASER, <0, TRUNKLEN, 0>, TRUNKCAPR }
        sphere { <0, TRUNKLEN, 0>, TRUNKCAPR }
```

/* Am Ende einer jeden union werden, falls vorhanden, die frei definierbaren Texture-Modifier eingefügt. Diese müssen, wie oben im Kapitel über die Parameter gezeigt, mit durchnummerierten Namen als vollständige Textur-Anweisungen deklariert werden. Falls für eine Baumkomponente keine Textur definiert wurde, wird die in selftree.inc definierte Default-Textur benutzt. */

```
        #ifdef ( MOD_TR )
            // falls definiert, Texture Modifier einfügen
            texture { MOD_TR }
        #end
    }
```

/* Schleifenzähler auf Anzahl (Äste+Differenz) und in Abhängigkeit v.d. aktuellen Rekursionsstufe setzen: Der Schleifenzähler bestimmt die Anzahl der zu erzeugenden Äste in der zugehörigen Rekursionsstufe (hier Stufe 0, die Äste, die direkt am Stamm sitzen). Der Wert wird berechnet, indem zu dem vorgegebenen Wert NRBRANCHES ein Zufallswert zwischen 0 und der erlaubten Abweichung NRBDIFF hinzuaddiert wird. Anschließend wird der Wert noch mit dem Skalierungswert für die Astanzahl hoch der aktuellen Rekursionsstufe (hier also $NRBSCALEMUL^0 = 1$) multipliziert, und mit dem zweiten Skalierungswert für die Astanzahl mal der aktuellen Rekursionsstufe (hier also NRBSCALEADD*0 = 0) addiert. — Aus den letzten beiden Operationen ergibt sich zwar an dieser Stelle kein Effekt, aber an den Stellen, an denen die Rekursionstiefe ungleich 0 ist, wird der Wert der zu erzeugenden Äste dadurch entsprechend verändert. */

```
#declare DEPTH_0 = (NRBRANCHES+(rand(RANDNRBDIFF)*NRBDIFF)) \
                          *pow(NRBSCALEMUL,0)+(NRBSCALEADD*0)
#while ( DEPTH_0 > 0 )

// Rekursionsstufe: 0
union
{
```

/* Aktuellen Astradius berechnen: Der Radius eines Astes wird berechnet, indem der Ausgangsradius (RADIUS) mit der Potenz aus dem Skalierungswert für den Astradius (RADSCALE) hoch der aktuellen Rekursionstiefe (hier auch wieder 0 wegen der R.-tiefe 0, der eigentliche Effekt tritt also auch erst wieder bei tieferen Stufen auf) multipliziert wird. Zuvor wird aber noch ein Differenzwert abgezogen, der zwischen 0 und RDIFF liegt, so daß Äste derselben Rekursionsstufe verschiedene Radien haben können, was das Ausssehen des Baumes natürlicher macht. */

```
#declare ACTRAD = (RADIUS-(rand(RANDRDIFF))*RDIFF)\
                     *pow(RADSCALE,0)
```

/* Aktuelle Astlänge berechnen: Die Länge eines Astes berechnet sich ähnlich wie schon die zwei vorhergehenden Werte. Zuerst wird zu dem Längenwert BLEN ein Wert zwischen 0 und der erlaubten Längenabweichung LDIFF addiert. Der erhaltene Wert wird dann multipliziert mit der Potenz aus dem Skalierungswert für die Astlänge (RECSCALE) hoch der aktuellen Rekursionstiefe. */

```
#declare BLEN_0 = (BLEN+(rand(RANDLDIFF)*LDIFF))\
                    *pow(RECSCALE,0)
```

/* Der Endradius des Konus, der für den Ast erzeugt wird, wird durch Multiplikation mit dem Astskalierungswert so gewählt, daß er mit dem Startradius der Äste der nachfolgenden Rekursionsstufe übereinstimmt. */

```
cone { <0, 0, 0>, ACTRAD, <0, BLEN_0, 0>, ACTRAD*RADSCALE }
sphere { <0, BLEN_0, 0>, ACTRAD*RADSCALE }
```

/* Schleifenzähler auf Anzahl (Äste+Differenz) und in Abhängigkeit v.d. aktuellen Rekursionsstufe setzen: Ab hier werden in eine union die unteren Rekursionsstufen eingeschachtelt, ab hier läuft also erstmal bis zum nächsten ausführlichen Kommentar alles wie oben beschrieben. — Dadurch, daß die hier berechneten Werte allerdings alle bei jedem Schleifendurchlauf der höheren Rekursionsstufe neu ermittelt werden, ist es möglich, daß z.B. die verschiedenen Äste auf der gleichen Rekursionsstufe verschieden viele Sohnzweige haben, was ebenfalls zu einem echteren Aussehen der Bäume beiträgt. */

```
#declare DEPTH_1 = (NRBRANCHES+(rand(RANDNRBDIFF)
           *NRBDIFF))*pow(NRBSCALEMUL,1)+(NRBSCALEADD*1)
#while ( DEPTH_1 > 0 )
// Rekursionsstufe: 1
union
{
   // Aktuellen Astradius berechnen
   #declare ACTRAD = (RADIUS-(rand(RANDRDIFF))*RDIFF)\
                     *pow(RADSCALE,1)
   // Aktuelle Astlänge berechnen
   #declare BLEN_1 = (BLEN+(rand(RANDLDIFF)*LDIFF))\
                     *pow(RECSCALE,1)
   cone { <0, 0, 0>, ACTRAD, <0, BLEN_1, 0>,\
                           ACTRAD*RADSCALE }
   sphere { <0, BLEN_1, 0>, ACTRAD*RADSCALE }

   // Schleifenzähler auf Anzahl (Äste+Differenz) und in
   // Abhängigkeit v.d. aktuellen Rekursionsstufe setzen.
   #declare DEPTH_2 = (NRBRANCHES+(rand(RANDNRBDIFF)\
            *NRBDIFF))*pow(NRBSCALEMUL,2)+(NRBSCALEADD*2)
   #while ( DEPTH_2 > 0 )
   // Rekursionsstufe: 2
   union
   {
      // Aktuellen Astradius berechnen
      #declare ACTRAD = (RADIUS-(rand(RANDRDIFF))\
                     *RDIFF)*pow(RADSCALE,2)
      // Aktuelle Astlänge berechnen
      #declare BLEN_2 = (BLEN+(rand(RANDLDIFF)*LDIFF))\
                        *pow(RECSCALE,2)
      cone { <0, 0, 0>, ACTRAD, <0, BLEN_2, 0>,\
                              ACTRAD*RADSCALE }
      sphere { <0, BLEN_2, 0>, ACTRAD*RADSCALE }
```

/* Schleifenzähler auf Anzahl (Äste+Differenz) und in Abhängigkeit v.d. aktuellen Rekursionsstufe setzen.

Dies ist die unterste Rekursionsstufe in diesem Beispiel und damit die einzige nicht weiter geschachtelte union. In der untersten Stufe erhält der erzeugte Konus keine abschließende Kugel mehr, um die Anzahl der erzeugten Objekte zu reduzieren. */

```
#declare DEPTH_3 = (NRBRANCHES+(rand(RANDNRBDIFF)\
      *NRBDIFF))*pow(NRBSCALEMUL,3)+(NRBSCALEADD*3)
#while ( DEPTH_3 > 0 )
object
{
   // Aktuellen Astradius berechnen
   #declare ACTRAD = (RADIUS-(rand(RANDRDIFF))\
            *RDIFF)*pow(RADSCALE,3)
   // Unterste Astebene ohne abschließende Kugel
   #declare BLEN_3 = (BLEN+(rand(RANDLDIFF)*LDIFF))\
                  *pow(RECSCALE,3)
   cone { <0, 0, 0>, ACTRAD, <0, BLEN_3, 0>,\
                          ACTRAD*RADSCALE }
```

/* Auch bei den Ästen können selbstdefinierte Texturen eingebunden werden. Falls wie oben beschrieben eine Textur für eine bestimmte rekursionsstufe definiert ist, wird sie jeweils an der entsprechenden Stelle eingefügt. Ansonsten wird die Default-Textur genommen. */

```
   #ifdef ( MOD_3 )
      // falls definiert, Texture Modifier einfügen
      texture { MOD_3 }
   #end
```

/* Astneigung: Jeder Ast ist gegenüber seinem Vorgänger um einen Winkel von SLOPEANGLE geneigt, wobei der Winkel durch Addition eines Zufallswertes zwischen 0 und SANGLEDIFF noch variieren kann. Außerdem wird der errechnete Winkel noch in Abhängigkeit von ANGLESCALE und der aktuellen Rekursionsstiefe skaliert. */

```
   rotate <(SLOPEANGLE+rand(RANDSDIFF)*SANGLEDIFF)\
           *pow(ANGLESCALE,2), 0, 0>
```

/* Innerhalb des Rotationsbereiches drehen: Nachdem der Ast geneigt wurde, wird er um einen zufälligen Winkel zwischen 0 und ROTRANGE gedreht. Durch die Bereichsangabe mit ROTRANGE gibt es eine Möglichkeit, so etwas Ähnliches wie eine Ausrichtung des Baumes nach Wind zu simulieren. Leider ist es nicht möglich, diese Ausrichtung durch alle Rekursionsebenen durchzuhalten, da man nicht mehr weiß, welche Transformationen die unterliegenden Stufen durchgemacht haben. Einen Transformationsstack kann das Programm derzeit nicht verwalten. Vielleicht ein Punkt für die Todo-Liste...*/

```
            rotate <0, ROTRANGE*rand(RANDROT), 0>
```

/* Am Astansatz verschieben: Es wird ein Zufallswert zwischen 0 un 1 ausgewürfelt. Ist dieser größer oder gleich als BRANCHTOP, so wird der erzeugte Ast an einen zufällig gewählten Punkt seines Vaterastes verschoben, ist er kleiner als BRANCHTOP, so wird der Ast an die Spitze seines Vaterastes verschoben. */

```
            #if ( rand(RANDBT) < BRANCHTOP )
               // an Astspitze verschieben
               translate <0, BLEN_2, 0>
            #else
            // an beliebigen Punkt verschieben
            translate <0, (BLEN_2)*rand(RANDBT), 0>
            #end
         }
```

/* Zuletzt noch schnell den Zähler für die Anzahl verbleibender Schleifendurchgänge runtersetzen.

Der Rest des Codes ist nicht weiter kommentiert, um da er nur Wiederholungen des bisher kommentierten Abschnitts bietet. */

```
         #declare DEPTH_3 = DEPTH_3 - 1
         #end

         #ifdef ( MOD_2 )
            // falls definiert, Texture Modifier einfügen
            texture { MOD_2 }
         #end
         // Astneigung
         rotate <(SLOPEANGLE+rand(RANDSDIFF)*SANGLEDIFF)\
                  *pow(ANGLESCALE,1), 0, 0>
         // Innerhalb des Rotationsbereiches drehen
         rotate <0, ROTRANGE*rand(RANDROT), 0>
         // am Ansatzast verschieben
         #if ( rand(RANDBT) < BRANCHTOP )
            // an Astspitze verschieben
            translate <0, BLEN_1, 0>
         #else
            // an beliebigen Punkt verschieben
            translate <0, (BLEN_1)*rand(RANDBT), 0>
         #end
      }
      #declare DEPTH_2 = DEPTH_2 - 1
      #end
```

```
        #ifdef ( MOD_1 )
           // falls definiert, Texture Modifier einfügen
           texture { MOD_1 }
        #end
        // Astneigung
        rotate <(SLOPEANGLE+rand(RANDSDIFF)*SANGLEDIFF)\
                 *pow(ANGLESCALE,0), 0, 0>
        // Innerhalb des Rotationsbereiches drehen
        rotate <0, ROTRANGE*rand(RANDROT), 0>
        // am Ansatzast verschieben
        #if ( rand(RANDBT) < BRANCHTOP )
           // an Astspitze verschieben
           translate <0, BLEN_0, 0>
        #else
           // an beliebigen Punkt verschieben
           translate <0, (BLEN_0)*rand(RANDBT), 0>
        #end
     }
     #declare DEPTH_1 = DEPTH_1 - 1
     #end
     #ifdef ( MOD_0 )
        // falls definiert, Texture Modifier einfügen
        texture { MOD_0 }
     #end
     // Astneigung
     rotate <(SLOPEANGLE+rand(RANDSDIFF)*SANGLEDIFF)\
              *pow(ANGLESCALE,-1), 0, 0>
     // Innerhalb des Rotationsbereiches drehen
     rotate <0, ROTRANGE*rand(RANDROT), 0>
     // Ans Stammende des Baumes verschieben
     translate <0, TRUNKLEN, 0>
  }
  #declare DEPTH_0 = DEPTH_0 - 1
#end
}
```

6 Historie

Da dies abgesehen von der tcl-Version die erste Version des Programms ist, ist die Geschichte des Dings recht kurz:

- Oktober 1996: erste tcl-Version

- November 1996: PoV-Include-Version

7 Todo

- Zur Zeit sehen die Bäume noch recht künstlich aus, da ein Ast immer nur aus einem Konus und einer abschließenden Kugel besteht. Besser wäre entweder für jeden Ast eine wählbar unregelmäßige Folge von Objekten (z.B. Konusse, Blobs etc.), so daß ein einzelner Ast nicht mehr einfach eine gerade Linie ist. (Ich arbeite daran...)
- Für die verschiedenen von der aktuellen Rekursionstiefe abhängigen Werte sollte man eine weitere Möglichkeit der Berechnung haben. Zur Zeit stelle ich mir die Angabe einer Funktion vor, die als Parameter die aktuelle Rekursionsstufe bekommt und daraus einen Wert berechnet. (Ich arbeite dran...)
- Zur Zeit sind die erzeugten Sources nicht korrekt eingerückt, eine Erweiterung um ein weiteres Include-File könnte dieses Problem lösen (Denke ich später drüber nach...)
- Frei einbindbarer selbstdefinierter Stamm.

Für weitere Ideen und Vorschläge zur Erweiterung bin ich dankbar. Vorschläge bitte an die gleiche Adresse wie bei den Bugs unten.

8 BUGS

Keine bekannt bisher. Wer welche entdeckt, möge mir diese bitte mailen an:

stk@informatik.uni-bremen.de

9 Dankeschön

Hey, Leute (Holger, Harald, Ralf), ihr könnt hier im Moment noch nicht stehen, weil ihr noch nichts getestet habt... Wenn ihr eine Zeit lang ein paar Bäume gemacht habt, sehen wir weiter...

10 Copyright