Genereren van **realistische bomen** met behulp van **L-Systems**

5 April, 2004

TU Delft
Technische Universiteit Delft

Remco van Arkel,
Thijs Kruithof

1 Inhoudsopgave

2 Het doel

Het door ons gestelde doel van dit project is tweezijdig. Aan de ene kant willen we met behulp van verschillende technieken zo natuurgetrouw mogelijk 3-dimensionale modellen genereren van bomen. Aan de andere kant streven we naar een zo groot mogelijke controle over alle aspecten van de boomstructuur, ook wel “controlled randomness” genoemd.

Voor het genereren van realistische/natuurgetrouwe objecten bestaan reeds diverse technieken. Eén techniek die al geruime tijd bestaat en tevens zeer geschikt is voor het modelleren van planten en bomen is het gebruik van *L-Systems* [1,14]. Verder zijn er ook andere algoritmen bekend (zoals Holton’s model [2]), echter hebben wij ons project specifiek gericht op het gebruik van L-Systems.

3 Theorie

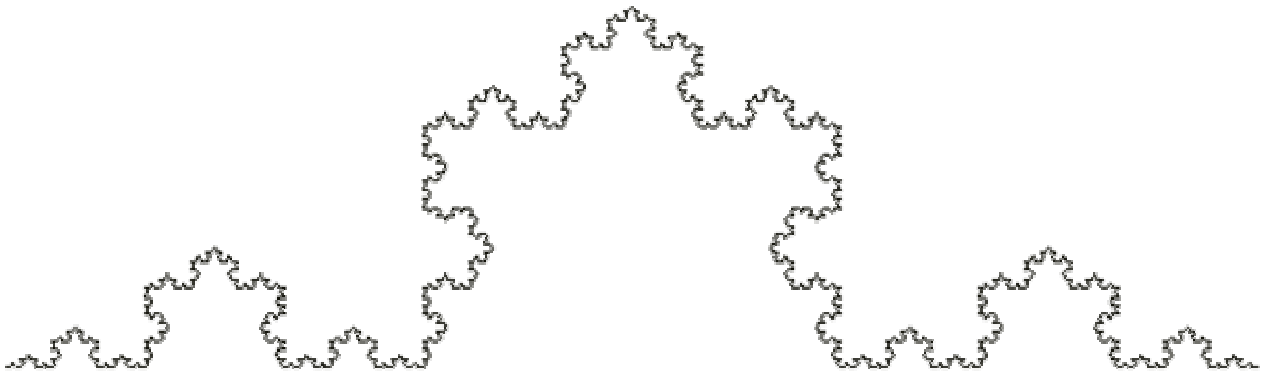
3.1 L-Systems

In het algemeen zijn L-Systems een vorm van procedurele graphics. Bij procedurele graphics worden beelden/modellen gegenereerd aan de hand van bepaalde procedures. Bij L-Systems zijn de procedures kleine programma's die wanneer ze geëvalueerd worden modellen opleveren. Deze programma's bestaan uit diverse regels welke zijn opgebouwd uit reeksen van commando's.

Om direct duidelijk te maken hoe L-Systems zijn opgebouwd en hoe ze verwerkt worden zullen we beginnen met een voorbeeld van een eenvoudig L-System en kort behandelen hoe deze geëvalueerd wordt en wat het resultaat van de evaluatie zal zijn.

3.1.1 Koch curve

De Koch curve [3] is een goed voorbeeld van een eenvoudig L-System. In onderstaand figuur is een evaluatie van de Koch curve te zien (een zogenaamde vijfde orde evaluatie).



Figuur 3.1: Koch curve (5de orde)

De bovenstaande figuur wordt verkregen door een voor de Koch curve specifiek programma te evalueren. Het programma dat hiervoor nodig is ziet er als volgt uit:

```
F = F + F - - F + F
```

Zonder direct de betekenis uit te leggen van bovenstaande reeks karakters kan gezegd worden dat het programma bestaat uit één regel (een regel bestaat uit een reeks commando's toegekend aan één letter) voor **F**, welke bestaat uit een reeks van 8 opeenvolgende commando's.

Door dit programma één keer te evalueren wordt het volgende figuur verkregen:

Figuur 3.2: Koch curve (1ste orde)

3.1.2 Turtle graphics

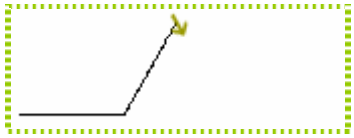
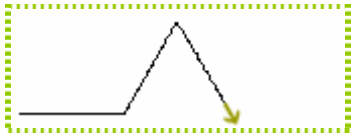
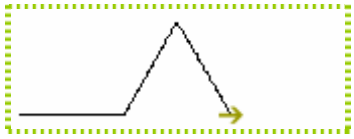
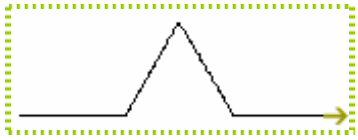
De bij de Koch curve gebruikte commando's zijn zogenaamde *Turtle commands*. Turtle commands komen oorspronkelijk uit de programmeertaal LOGO [4], welke een graphics subsysteem bevatten dat gebaseerd was op schildpad commando's.

Een systeem gebruikmakend van Turtle graphics bevat een *status* en een set commando's die de status kunnen aanpassen. De status wordt gezien als een schildpad, en bevat een positie waar hij zich bevindt en een oriëntatie richting.

L-Systems maken ook gebruik van deze zogenaamde Turtle graphics. Sommige van de commando's waaruit de regels opgebouwd zijn hebben effect op de status. Naast deze "schildpadcommando's" zijn er binnen L-Systems ook andere commando's o.a. voor het sturen van recursie e.d. De volgende commando's zijn in feite de basis commando's van L-Systems en hebben effect op de schildpadstatus:

Commando	*Effect op*	*Actie*
F	Positie	Verplaats schildpad een vaste afstand naar voren (in de oriëntatierichting) en teken een lijn van vorige positie naar nieuwe positie.
f	Positie	Verplaats schildpad een vaste afstand naar voren (in de oriëntatierichting).
+	Oriëntatie	Roteer de oriëntatierichting een vaste hoeveelheid graden tegen de richting van de wijzers van de klok in.
-	Oriëntatie	Roteer de oriëntatierichting een vaste hoeveelheid graden met de richting van de wijzers van de klok mee.

Wanneer we de commando's voor het genereren van een Koch curve achter elkaar uitvoeren en daarbij beginnen met onze schildbad links onderaan het scherm met de schildbad naar rechts wijzend krijgen we de volgende reeks figuren, waarbij het laatste figuur het uiteindelijke resultaat is:

Commando's	*Resulterend figuur*	*Commando's*	*Resulterend figuur*
<geen>		**F+F--**	
F		**F+F--F**	
F+		**F+F--F+**	
F+F		**F+F--F+F**	

3.1.3 L-System evaluatie

Het evalueren van een L-System programma begint bij het *Axioma*. Het Axioma is een speciale regel die slechts bestaat om aan te geven waar begonnen moet worden met evalueren. De Koch curve heeft als eigenlijk programma:

```
Axiom = F
F = F + F - - F + F
```

Het evalueren gaat nu als volgt in zijn werk: Allereerst wordt de reeks commando's voor het axioma ingelezen. Vervolgens wordt het eerste commando van deze reeks bekeken; indien voor dit karakter (elk commando wordt namelijk gerepresenteerd door een karakter) een regel aanwezig is én de maximale recursiediepte nog niet behaald is zal de regel voor dit karakter worden geëvalueerd waarna het volgende karakter uit de huidige reeks commando's wordt verwerkt (op dezelfde wijze als het vorige karakter). Indien er voor een karakter geen regel aanwezig is, of wanneer de maximum recursiediepte bereikt is, zal het karakter als commando opgevat worden en uitgevoerd worden.

In pseudo-code ziet het verwerken van een programma er als volgt uit:

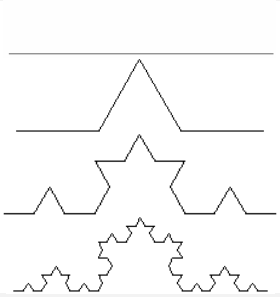
```
1.    Function ParseRule(commands : list; iteration : integer)
2.         For i = 1 to count(commands)
3.
4.              If (iteration < MaxIteration) and
5.                 (RuleExistsForCommand(commands[i]) then
6.
7.                   ParseRule(GetRuleForCommand(commands[i]),
8.                             iteration+1)
9.              Else
10.                  ExecuteCommand(commands[i])
11.             End If
12.
13.        End For
14.   End Function
```

Waarbij begonnen wordt met het volgende stuk pseudo-code:

```
    ParseRule(GetAxiomRule(), 1)
```

Uiteraard is dit een stuk code in een fictieve programmeertaal, waarbij hopelijk de namen van de gebruikte routines voldoende indicatie geven voor de werking ervan. Verder is op regel 7 de (begrensde) recursie te zien.

Om de werking van dit algoritme te verduidelijken zullen we de hierboven vermelde Koch curve een aantal maal evalueren, waarbij steeds de maximum iteratie diepte met 1 opgehoogd zal worden, beginnend met een maximum iteratiediepte van 1.

Maximum Iteratiediepte	*Werkelijk verwerkte reeks commando's*	*Resultaat*
1	F	
2	F+F–F+F	
3	F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F	
4	F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F+F+F-- F+F+F+F--F+F--F+F--F+F+F+F--F+F (etc.)	

Duidelijk is te zien dat een hogere maximum iteratiediepte een complexer resultaat oplevert.

Naast de maximum iteratiediepte moeten er voor aanvang van de evaluatie nog meer parameters bepaald worden. Zo worden ook een *standaard lijnlengte*, een *standaard draaihoek*, een *lijnlengte factor* en een *draaihoek factor* gedefinieerd. De lengte van een te tekenen lijnstuk is nu afhankelijk van de iteratie; bij de eerste iteratie zullen alle lijnstukken de standaard lengte zijn, bij de volgende iteratie zal dit de standaard lengte zijn vermenigvuldigd met de lengte factor. Een zelfde constructie is ook van toepassing op de draaihoek. Hetgeen resulteert in de volgende twee definities:

$$LijnLengte = StandaardLijnLengte * LijnLengteFactor^{iteratie}$$
$$Draaihoek = StandaardDraaihoek * DraaihoekFactor^{iteratie}$$

3.1.4 Stacks

Het door L-Systems gebruikte commandoset bevat naast de zojuist behandelde commando's ook 2 speciale commando's:

Commando	*Actie*
[	Onthoud huidige positie en richting (eq. "zet huidige positie en richting van turtle op de stack")
]	Herstel positie en richting naar laatst onthouden waarden (eq. "stel huidige positie en richting gelijk aan waarden op de stack, en verwijder de waarden van de stack")

Bovenstaande commando's kunnen gebruikt worden om eenvoudiger complexe figuren te genereren. Zo levert de volgende code:

```
Axiom = F [+F] [-F]
```

het volgende figuur op na 1 iteratie:

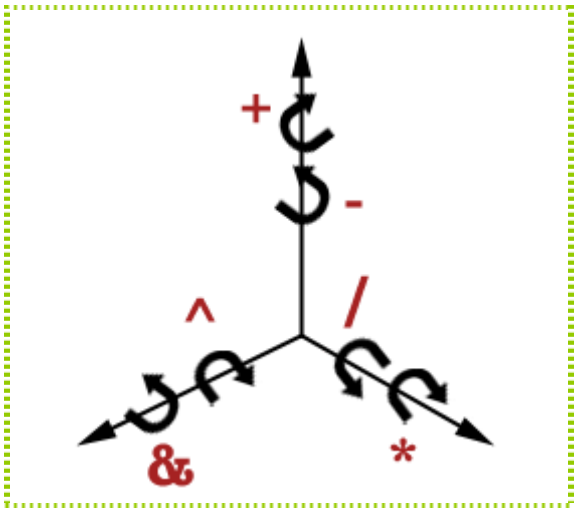
Figuur 3.3: Voorbeeld gebruik van stack-commando's

3.2 Uitbreidingen naar de derde dimensie

De in de vorige paragraaf uitgelegde L-Systems zijn oorspronkelijk bedoeld voor het construeren van 2-dimensionale structuren, zoals afbeeldingen van planten e.d. Echter kunnen zij ook gebruikt worden voor het genereren van 3-dimensionale structuren, door het enigszins uitbreiden van het commandoset.

Het standaard commandoset (zie paragraaf 3.1.2) bevat commando's om de positie aan te passen van de "turtle" en enkele commando's om de richting van de "turtle" aan te passen. De uitbreiding van 2-dimensionaal naar 3-dimensionaal heeft tot gevolg dat de positie van de "turtle" nu een punt in 3-dimensionale ruimte wordt én dat de richting van de "turtle" een 3-dimensionale vector wordt. Verder zijn meer commando's nodig die de richting van de "turtle" kunnen beïnvloeden; er zijn nu immers meer vrijheidsassen beschikbaar.

In het commandoset dat bedoeld was voor 2-dimensionale figuren waren twee commando's aanwezig voor het manipuleren van de richting: **+** en **-**, waarbij de **–** in feite dezelfde rotatie toepaste als de **+**, maar dan juist in tegengestelde richting. Dit zelfde principe wordt gebruikt in de 3-dimensionale situatie: voor elke vrijheidsas zijn twee commando's aanwezig welke allebei de richting laten roteren om de vrijheidsas én waar ze allebei elkaars rotatie kunnen opheffen. In figuur 3.4 zijn de drie vrijheidsassen weergegeven met de door ons toegekende commando's (de gekozen karakters zijn naar creatief inzicht gekozen).

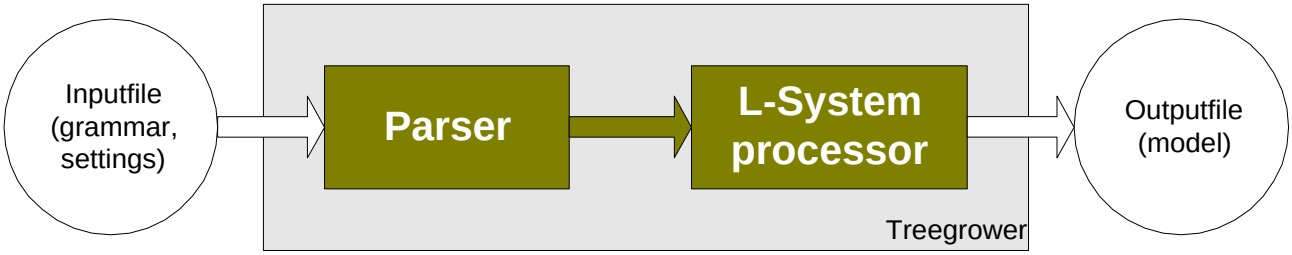


Figuur 3.4: Vrijheidsassen en bijbehorende rotatiecommando's

4 De bomengenerator

4.1 Overzicht

Het door ons ontwikkelde systeem is een losstaand programma geworden, welke als invoer één bestand accepteert welke het L-System programma bevat én eventueel enkele instellingen. De uitvoer van ons systeem is ook één bestand, dat het 3-dimensionale gegenereerde model bevat. Dit gegenereerde model kan vervolgens gevisualiseerd worden m.b.v. bijvoorbeeld een raytracer, zoals POV-ray [5].



Figuur 4.1: Structuuroverzicht

Zoals in figuur 4.1 is te zien is het systeem opgedeeld in twee afzonderlijke gedeelten: de parser, en de L-System processor. In de volgende paragrafen zullen de twee fasen kort uitgelegd worden.

4.2 De Parser

De Parser is verantwoordelijk voor het inlezen van het invoerbestand (de regels, maar ook aanwezige instellingen), het controleren op fouten in de invoer (typefouten, etc…) en het doorgeven van de ingevoerde gegevens aan de L-System processor in een handelbaar formaat. Omdat de invoerbestanden redelijk complex kunnen zijn qua opbouw hebben wij ervoor gekozen om de parser te implementeren als een compiler voor een programmeertaal (immers heeft de grammatica voor L-Systems vele overeenkomsten met een programmeertaal).

Onze parser is derhalve geïmplementeerd met behulp van de compiler constructie tools *bison* [6] en *flex* [7]. Deze opzet heeft redelijke voordelen; de door deze tools automatisch gegenereerde parser heeft ingebouwde ondersteuning voor het afhandelen van syntax-fouten. Verder kunnen de meest complexe grammaticaconstructies ondersteund worden. In appendix 6.1 is de volledige grammatica (bestemd als invoer voor Bison en Flex) na te lezen.

De parser slaat tevens tijdens het parsen van de invoerdata alle regels en instellingen in een intern formaat op. Als er tijdens het parsen geen fouten opgetreden zijn worden de gegevens in het interne formaat doorgegeven aan de L-System processor.

4.3 De L-System processor

De L-System processor is in principe een normale 3-dimensionale L-System processor welke aan de hand van de ingevoerde gegevens een model maakt. Het gegeneerde model wordt weggeschreven in een los bestand welke door POV-ray ingelezen en gerenderd kan worden (er wordt een bestand in een voor POV-ray specifiek bestemd formaat weggeschreven, hier is uit practische overwegingen voor gekozen).

In eerste instantie is functionaliteit van de L-System processor identiek aan de L-Systems beschreven in voorgaande paragrafen, echter zijn enkele uitbreidingen ingevoerd welke de mate van realisme van de gegenereerde modellen ten goede komen.

4.4 Toegevoegde L-System mogelijkheden

Om meer controle te krijgen over onze bomen hebben we een aantal mogelijkheden toegevoegd, die standaard L-Systems niet hebben. In deze paragraaf zullen deze mogelijkheden uiteengezet worden. Eerst zal steeds beschreven worden wat een bepaalde toevoeging is en wat je ermee kan bereiken. Tevens zal aangegeven worden waarom we voor deze toevoegingen hebben gekozen.

4.4.1 Variabelen & Operator parameters

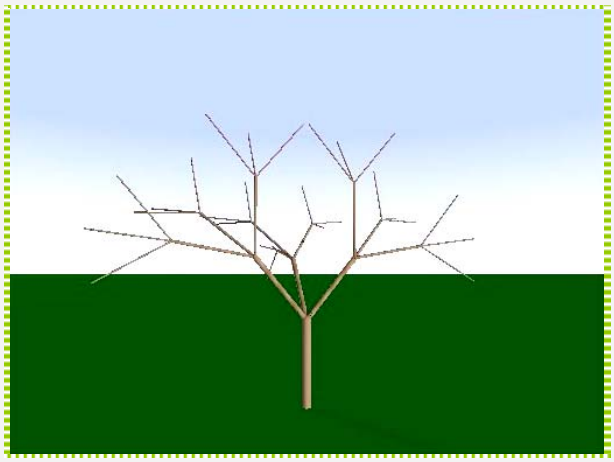
De eerste toevoeging was die van variabelen. Door het gebruik van variabelen kan de vorm van de boom beïnvloed worden. Dit gebeurt met behulp van een aantal standaard variablen, zoals *defaultlength*. Deze leggen de standaard eigenschappen van een boom vast. Ook kan de recursiediepte bepaald worden. Al deze standaard variabelen hebben ook een standaardwaarde en hoeven dus niet opgegeven te worden.

De reden om variabelen toe te voegen is in eerste instantie om meer controle over de boom te krijgen. Door een waarde aan deze variabelen toe te kennen, kan het uiterlijk van de bomen veranderd worden, zonder de generator aan te passen. Daarnaast zijn variabelen ook nodig voor de andere aanpassingen.

Ook kunnen variabelen gebruikt worden om de operatoren te beïnvloeden. Zo kan voor een bepaalde tak explicitiet de hoek worden opgegeven waaronder deze vanaf de stam of vorige tak groeit. Dit gebeurt door achter een operator (bijvoorbeeld F, +, -, etc.) een set haakjes te zetten, met daar tussenin een of meer variabelen. Operatoren kunnen dus als functies gebruikt worden en parameters meekrijgen. Operator parameters zijn altijd optioneel en de hoeveelheid parameters hangt af van de operator. Naast variabelen kunnen ook getalwaarden en simple wiskundige functies (*, /, +, -) gebruikt worden als parameters.

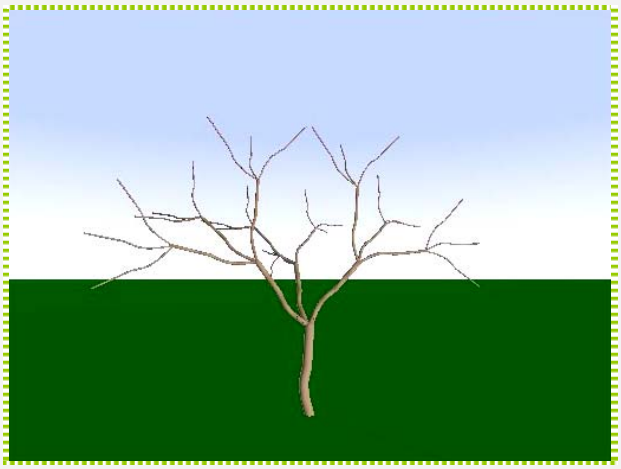
In de grammatica beginnen alle variabelen met een hekje (#). Dit is nodig om ze herkenbaar te maken voor de parser. Zonder het hekje zou het een functie worden (zie paragraaf 4.4.2). Variabelen worden gedeclareerd aan het begin van de grammatica door middel van een *SET* commando.

Hieronder twee verschillende plaatjes met bijbehorende code, die gemaakt kunnen worden door verschillende waardes voor de standaard variabelen. De code bij figuur 4.1 is de uitgangscode voor de rest van paragraaf 4.4. Toevoegingen aan de code worden in het blauw weergegeven.



Figuur 4.2: Een boom met rechte takken

```
// settings
SET #recursiondepth = 5
SET #defaultangle = 40
SET #defaultlength = 1
SET #defaultradius = 0.10
SET #lengthfactor = 0.97
SET #radiusfactor = 0.55

AXIOM -> A;

A -> F[+A][*A][-A];
```



Figuur 4.3: Een boom met gekromde takken

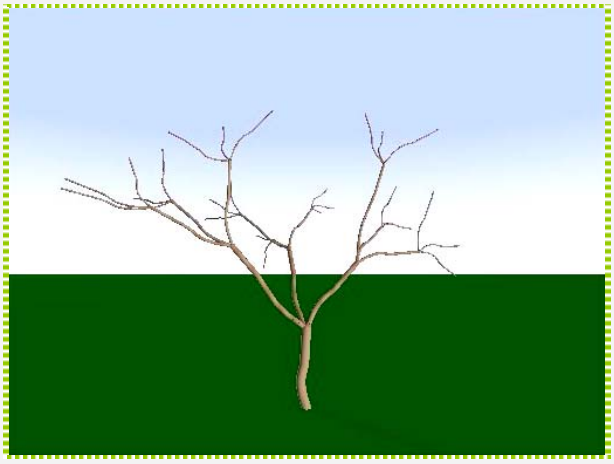
```
// settings
SET #recursiondepth = 5
SET #defaultangle = 40
SET #defaultlength = 1
SET #defaultradius = 0.10
SET #lengthfactor = 0.97
SET #radiusfactor = 0.55
SET #DefaultBLengthFactorF = 0.3
SET #DefaultBRadiusF = 0.1
SET #DefaultBLengthFactorS = 0.6
SET #DefaultBRadiusS = 0.0

AXIOM -> A;

A -> F[+A][*A][-A];
```

4.4.2 Stochastische functies

Een van de belangrijke eigenschappen van de natuur is dat deze niet volgens strakke regels werkt. Een tak groeit niet altijd onder precies dezelfde hoek uit de stam of vorige tak. Er zit een bepaalde willekeurige variatie in. Om onze bomen realistischer te maken moeten deze hier ook aan voldoen. Om dit te realiseren zijn er stochastische functies [8, 10] toegevoegd, zoals bijvoorbeeld een normaalverdeling.

In de grammatica worden stochastische functies weergegeven door functies, welke een waarde “trekken” uit een dataverzameling met een bepaalde verdeling. Deze functies kunnen aangeroepen worden als operator parameters. Ze hebben zelf ook een of meer parameters. De normaal verdeling bijvoorbeeld heeft als parameters de gemiddelde waarde en de variantie van de verdeling. In tegenstelling tot operator parameters zijn de parameters van de stochastische functies verplicht.

Hieronder een voorbeeld van een boom met een normaalverdeling op de hoek tussen twee takken.



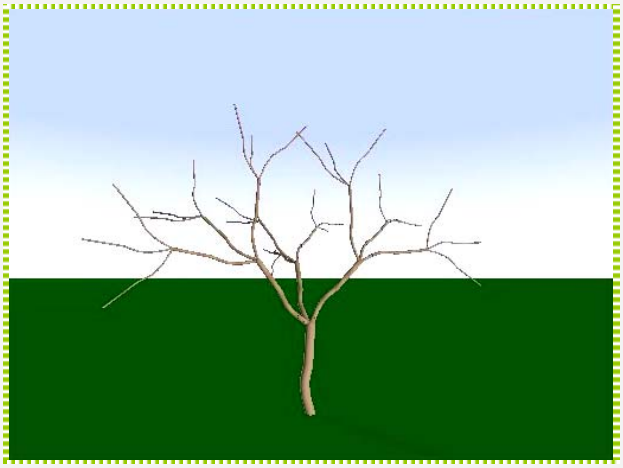
Figuur 4.4: Een boom met variabele hoeken

```
// settings
SET #recursiondepth = 5
SET #defaultangle = 40
SET #defaultlength = 1
SET #defaultradius = 0.10
SET #lengthfactor = 0.97
SET #radiusfactor = 0.55
SET #DefaultBLengthFactorF = 0.3
SET #DefaultBRadiusF = 0.1
SET #DefaultBLengthFactorS = 0.6
SET #DefaultBRadiusS = 0.0

AXIOM -> A;

A -> F[+(norm(#defaultangle,30))A]
     [*A][-A];
```

4.4.3 Regel parameters

Onze volgende stap was het toevoegen van parameters voor regels. Het idee hierachter is dat bomen hierdoor beïnvloed kunnen worden afhankelijk van de recursiediepte of aanroep. Een hoek kan bijvoorbeeld steeds iets groter worden, of een taklengte afhankelijk van de positie op een tak (door bijvoorbeeld de eerste tak met *#i* aan te roepen en de tweede tak met *2*#i*). Als een productieregel in zijn declaratie parameters heeft dan moet elke aanroep ook parameters bevatten, en wel evenveel als bij de declaratie. Productieregel parameters zijn dus bij de aanroep niet optioneel. Ze zijn dit wel bij de declaratie, regels zonder parameters blijven dus ook toegestaan.

Hieronder wederom een voorbeeld:



Figuur 4.5: Een boom met toenemende variatie in hoeken

```
// settings
SET #recursiondepth = 5
SET #defaultangle = 40
SET #defaultlength = 1
SET #defaultradius = 0.10
SET #lengthfactor = 0.97
SET #radiusfactor = 0.55
SET #DefaultBLengthFactorF = 0.3
SET #DefaultBRadiusF = 0.1
SET #DefaultBLengthFactorS = 0.6
SET #DefaultBRadiusS = 0.0

AXIOM -> A(0);

A(#i) -> F
         [+(norm(#defaultangle,
                 #i*5))A(#i+1)]
         [*A(#i+1)]
         [-A(#i+1)];
```

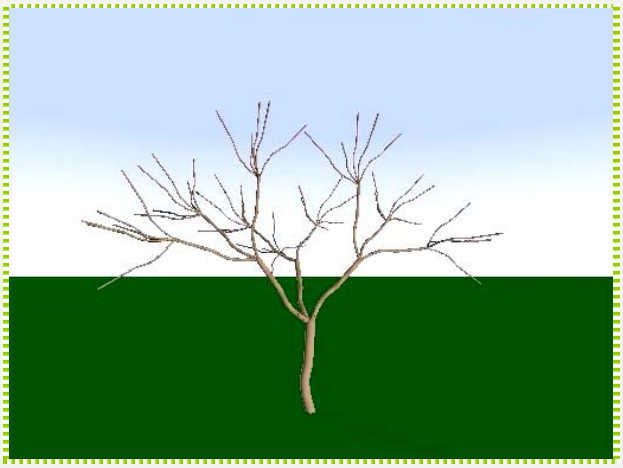
4.4.4 Aanroep condities voor regels

Tot slot hebben regels ook condities gekregen voor hun aanroep. Er kunnen nu verschillende versies van een productieregel bestaan, met verschillende aanroep condities. Als nu een regel wordt aangeroepen, wordt de eerste regel gekozen, waarvan aan de conditie voldaan wordt. De volgorde van het aflopen van regels is de volgorde van declaratie in de grammatica. Het is dus van belang dat de verschillende versies van een regel een aaneengesloten serie vormen, anders is het mogelijk dat de generatie van de boom halverwege stopt.

Een conditie wordt aan een regel toegevoegd door direct na de regelnaam en eventuele parameters een dubbele punt en een conditie toe te voegen. Een voorbeeld is: "*A(#i, #j): #i < #j -> …*".

Deze toevoeging is gedaan om een boom op verschillende momenten een verschillend gedrag te geven. Bijvoorbeeld vanaf een bepaald niveau in de boom (recursie) meer of minder takken toevoegen.

Hieronder nogmaals een voorbeeld.



Figuur 4.6: Een boom met meer takken op het hoogste niveau

```
// settings
...

AXIOM -> A(0);

A(#i) : #i < 2 -> F
      [+(norm(#defaultangle,#i*5))
       A(#i+1)]
      [*A(#i+1)]
      [-A(#i+1)];
A(#i) : #i >= 2 -> F
      [+(norm(#defaultangle,#i*5))
       A(#i+1)]
      [*A(#i+1)]
      [-A(#i+1)]
      [/A(#i+1)]
      [^A(#i+1)]
      [&A(#i+1)];
```

4.5 Takgeneratie

In de voorgaande paragrafen is beschreven hoe de grammatica werkt, wat deze kan en hoe deze tot stand is gekomen. Maar met een grammatica is er nog geen boom! Daarom wordt in deze paragraaf beschreven hoe de hiervoor beschreven grammatica's omgezet worden in bomen.

4.5.1 Gekromde takken: Twee pipelines

Standaard 2D L-systeem bomen kunnen gevisualiseerd worden met behulp van rechte lijnen, eventueel met verschillende dikte. Op dezelfde wijze kunnen 3D bomen gevisualiseerd worden met cilinders. Deze worden dan geplaatst met in de richting van de tak, met de gewenste richting. Om gaten op te vullen kunnen bollen gebruikt worden. Het resultaat is een boom met rechte takken (zie figuur 4.1)

Maar bomen met rechte takken zien er natuurlijk niet realistisch uit. Echte bomen hebben gekromde takken. En omdat wij realistische bomen willen maken, hebben onze bomen dat dus ook. Dit betekent dat de hierboven geschetste visualisatie methode niet meer voldoet. Daarom hebben we twee andere methodes bedacht en geïmplementeerd:

- Spheresweeps
- B-splines en cilinders.

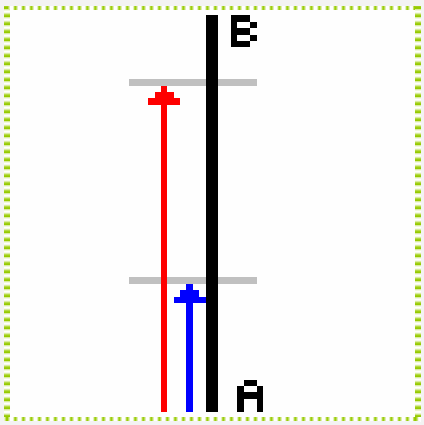
Beide methoden worden in de volgende subparagrafen beschreven. Maar eerst volgen nog wat algemene opmerkingen.

Beide methoden zijn gebaseerd op b-splines [11]. Dit betekent dat er controlepunten nodig zijn. De generator werkt met een viertal controlepunten. Deze worden als volgt bepaald:

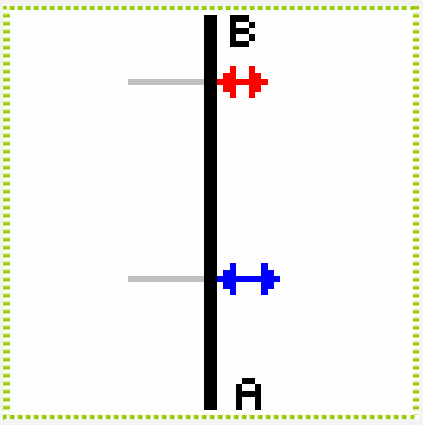
- Het eerste en laatste punt (C_1, C_4) zijn het begin en eindpunt van de tak (A en B)
- Twee factoren f_1 en f_2 bepalen de positie van de andere twee punten (C_2, C_3) ten opzichte van A. Dit is weergegeven in figuur 4.6. Speciale eigenschappen van deze waarden zijn: $0 \leq f_1 \leq f_2 \leq 1$.
- Vervolgens bepalen waarden r_1 en r_2 de afstand van de controlepunten tot de tak (figuur 4.7)
- Tot slot bepalen de hoeken α en β de uiteindelijke locatie van de controlepunten (figuur 4.8)

Alle genoemde variabelen (f_1, f_2, r_1, r_2, α en β) kunnen worden vastgelegd in de grammatica als standaardvariabelen en als parameters van operator *F*.

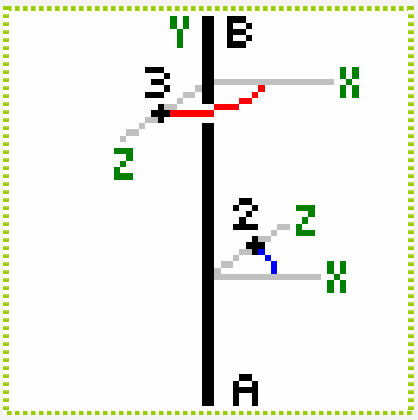
Daarnaast is er rekening gehouden met een goede overloop tussen twee opeenvolgende takken. De richting van de te plaatsen takken wordt namelijk bijgesteld aan de hand van de richting van de lijn tussen het 3e en het 4e controlepunt.



Figuur 4.7: f_1 (blauw) en f_2 (rood) bepalen de posities van C_2 en C_3 t.o.v. punt A



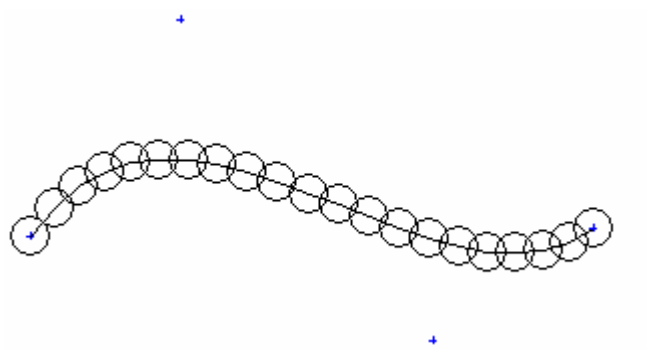
Figuur 4.8: r_1 (blauw) en r_2 (rood) bepalen de afstand van C_2 en C_3 tot tak (A,B)



Figuur 4.9: α (blauw) en β (rood) bepalen de uiteindelijke posities van C_2 en C_3

4.5.2 Pipeline 1: Spheresweeps

De eerste toegepaste methode is het gebruik van zogenaamde spheresweeps [12]. Een spheresweep is een translatiesweep (uitsmering over een pad) van een bol (Engels: sphere) over een kubische b-spline. De spheresweep is ingebouwd in POV-Ray en daarom voor ons gemakkelijk te gebruiken. De b-splines hoeven niet doorgerekend te worden, er hoeven enkel een zestal(!) controlepunten opgegeven te worden.



Figuur 4.10: Een spheresweep (in 2D)

Dit zijn er twee meer dan hiervoor beschreven staan. De twee extra benodigde punten zitten voor het begin en na het eindpunt van de spline en bepalen de start en eindrichting. Deze punten worden dan ook verkregen door het C_2 over C_1 heen te kopieëren (zelfde afstand, tegenovergestelde richting) en C_3 over C_4. Zo worden respectievelijk C_0 en C_5 gevonden. Nu hoeft er enkel een POV-Ray element in het POV-Ray bestand te worden weggeschreven.

Zoals gezegd is het een voordeel dat deze spheresweeps voor ons gemakkelijk in gebruik zijn. Voor de generator is het weinig rekenwerk en het levert een compacte POV-Ray file op. Een tweede voordeel is dat er altijd een mooie gladde vorm ontstaat.

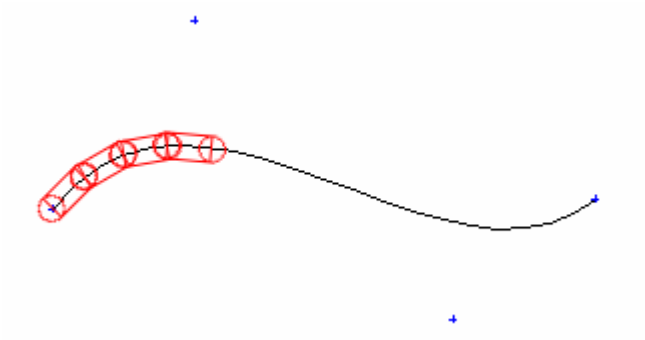
Daarnaast zijn er ook een aantal nadelen. In de eerste plaats kost het renderen van een spheresweep veel rekenkracht en daarmee veel tijd. Als tweede is er het nadeel dat er met

bijvoorbeeld intersecties met andere vormen niet altijd goede sweeps ontstaan. Dit komt omdat de (intersectie)vergelijkingen dan te complex worden. Er kunne dan gaten of onderbrekingen in de takken ontstaan.

Om deze methode te kiezen kan de standaard variable "*UseSweep*" op 1 gezet worden. Iedere andere waarde resulteert in de 2e methode. De standaardwaarde van "*UseSweep*" is 0.

4.5.3 Pipeline 2: Cilinders en b-splines

Onder andere omdat de spheresweep wat vervelende nadelen heeft, is er nog een andere methode geïmplementeerd. Deze methode gaat uit van een zelf door te rekenen b-spline die vervolgens benaderd wordt door een serie cilinders. Net als met de standaard aanpak (zie paragraaf 4.5.1) kunnen gaten bij de aansluitingen van twee cilinders opgevuld worden met bollen. Dit is weergegeven voor 2D in figuur 4.10.



Figuur 4.11: Een b-spline benaderd door balken en cirkels (in 2D)

Voordeel van deze aanpak is dat het renderen veel sneller gaat. Het weergeven van bollen en cylinders vergt lang niet zulke complexe berekeningen als spheresweeps. Als je een eenmaal berekende boom vaak wilt renderen is dit een groot voordeel. Een ander voordeel is dat de vorm altijd goed weergegeven wordt. Er zullen geen gaten of onderbrekingen optreden!

Een nadeel is dat de b-spline nu door de generator moet worden doorgerekend. Dat is dus voor ons meer werk. Daarnaast zal de POV-Ray file nu veel groter worden. Immers, een spheresweep wordt nu vervangen door een hele serie van cilinders en bollen. Derde nadeel is dat deze aanpak altijd een benadering opleverd. Het is een afweging tussen veel cilinders en hoekige krommen.

Om het laatste nadeel aan te pakken is er een optimalisatie toegevoegd. In plaats van de kromme op te delen in een vast aantal cilinders wordt nu per stukje gekeken wat de hoek is tussen het huidige en het volgende stuk. Is deze hoek kleiner dan een bepaalde drempelwaarde dan worden deze twee stukken samengenomen als één cilinder. Deze cilinder heeft als lengte de som van de twee spline-stukken. Dit proces wordt iteratief herhaald, tot een hoek groter dan de drempelwaarde gevonden wordt. Hierdoor kunnen veel cilinders, die op een (nagenoeg) rechte lijn liggen, vervangen worden door een langere cilinder.

De drempelwaarde en het aantal cilinders per kromme maken deel uit van het programma en kunnen niet door de grammatica beïnvloed worden.

4.6 Bladgeneratie

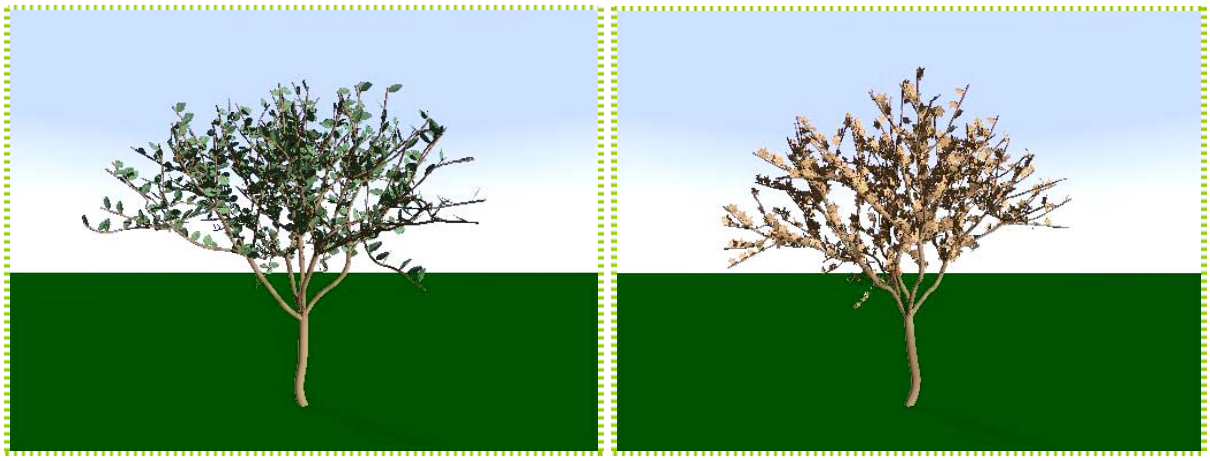
Een echte boom bestaat, behalve in de winter, uit meer dan alleen takken. Er zitten ook bladeren aan. Tijd dus om ook bladeren aan de generator toe te voegen.

Alvorens bladeren aan de boom toegevoegd kunnen worden, moeten er eerst bladeren zijn. In de gegenereerde bomen zijn bladeren vierkante polygons van 1 bij 1 die in het midden geknikt zijn. Deze worden geroteerd, getransleerd en geschaald om ze op de uiteindelijke positie te krijgen. Op het polygon staat een 32-bits RGBA-Texture. Er zijn hiervan drie varianten. Deze staan in figuur 4.11

Figuur 4.12: De drie blad-texturen

Nu er bladeren gedefinieerd zijn, moeten deze aan de boom geplaatst worden. Dit wordt gedaan tijdens het genereren van de takken. Na het genereren van een tak wordt de b-spline nogmaals afgelopen. In deze tweede iteratie door de b-spline worden op vaste intervallen bladeren tegen de tak aan geplaatst. Dit gebeurt door op de plaats waar een blad moet komen, de verschilvector te nemen die door dat punt van de b-spline gaat en een punt een klein stukje verder op de bspline: een lokale, lineaire benadering. De verschilvector wordt genormaliseerd en vervolgens uitgedrukt in een lengte (=1) en twee hoeken (een y-hoek en een z-hoek). Omdat een blad-object bij de definitie op de y-as ligt en een tak op de x-as, staat het blad nu loodrecht op de tak. Om niet alle bladeren dezelfde hoek te geven krijgt het blad ook een willekeurige x-hoek (rotatie om de x-as), die ligt tussen –90° en 90°. Om de bomen een iets beter uiterlijk te geven, worden pas vanaf een bepaald niveau (recursie) bladeren geplaatst. Dit gaat buiten de grammatica om.

Hieronder een voorbeeld van twee bomen met bladeren.



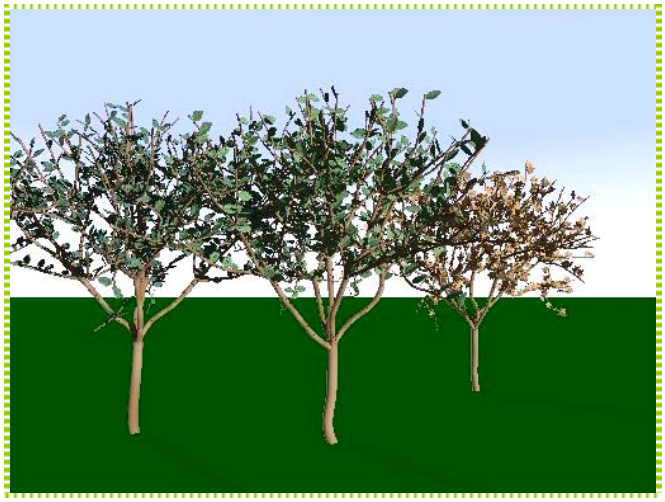
Figuur 4.13: Twee bomen mét bladeren

4.7 Bomen als POV-Ray objecten

Om een boom te kunnen renderen in POV-Ray is er naast de definitie van een boom, zoals die hierboven beschreven is, ook nog een deel overhead nodig. Deze overhead bestaat onder andere uit de definitie van een camera en lichtbronnen. Om deze overhead niet steeds opnieuw te hoeven genereren, worden de bomen gegeneerd als een POV-Ray object in een include-file. Dit bestand kan ingevoegd worden in een POV-Ray scene en vervolgens kan de boom geplaatst worden door een instantie van het boom-object aan te maken.

Het gebruik van een boom als object heeft enkele interessante voordelen. Zoals gezegd scheelt het in de eerste plaats overhead bij het genereren. Daarnaast ontstaat er meer vrijheid in het gebruik van de bomen. Camerapositie en lichtbronnen kunnen onafhankelijk van de generator geplaatst worden. Ook kan een complete boom gemakkelijk en overzichtelijk getransleerd, geroteerd en geschaald worden en in iedere willekeurige scene geplaatst worden. Tot slot kan een boom kan ook meerdere malen in een scene geplaatst worden en kunnen ook verschillende, afzonderlijk gegenereerde bomen in dezelfde scene geplaatst worden.

Een voorbeeld van meer bomen in één scene is te vinden in figuur 4.13.



Figuur 4.14: Meerdere bomen in één scene

5 Reflectie

De generator zoals hierboven beschreven kan redelijk natuurgetrouwe bomen genereren. Uiteraard zijn deze nog niet perfect. In dit hoofdstuk zullen de belangrijkste inperfecties en mogelijke verbeteringen besproken worden.

5.1 Huidige inperfecties

Een eerste, belangrijke inperfectie die opgemerkt kan worden, is dat we ons streven naar controle niet geheel nagekomen zijn. Het duidelijkste voorbeeld hiervan zijn de bladeren. Deze zijn op geen enkel punt in de grammatica vertegenwoordigd. Om bijvoorbeeld een ander bladtype te kiezen moet de code herschreven en opnieuw gecompiled worden.

Een tweede voorbeeld van het gebrek aan controle is de mogelijkheid tot het specificeren van de in- en output-file. Of eigenlijk meer het gebrek eraan. Dit gebrek is meer iets dat vergeten is, dan iets dat niet binnen het tijdsbestek van dit project te realiseren is.

Nog meer van deze gebreken waarover controle gewenst zou kunnen zijn, ofwel via de grammatica, ofwel via programma parameters, zijn bijvoorbeeld: de objectnaam van de boom en de texture van de takken.

Maar gebrek aan controle is niet de enige inperfectie. Er is bijvoorbeeld ook een visuele inperfectie. De takken zijn namelijk gewoon tegen elkaar geplaatst. Dit kan soms hele ongewenste overgangen geven, vooral als er een groot verschil in diameters is. Er ontstaat dan het idee van naalden die in een speldenkussen geprikt zijn.

Tot slot zijn er nog allerlei natuurlijke verschijnselen die van invloed zijn op de groei van een echte boom en die niet in de generator zitten. Enkele voorbeelden met gevolgen zijn:

De invloed van wind. Sterke of veel wind kan een boom naar een bepaalde kant doen overhellen. Hierdoor ontstaat een gekromde boom.

Een zelfde soort verschijnsel kan ontstaan door zwaartekracht. Grote, zware takken zullen met grote waarschijnlijkheid naar beneden gaan hangen.

Ook een belangrijke factor is zonlicht. Een boom zal sneller en dichter groeien aan de zon zijde dan aan de schaduw zijde. Ook de groei van bladeren wordt door zonlicht beïnvloed. Deze proberen namelijk een zo groot mogelijk opvlak naar de zon toe te richten om fotosynthese te bevorderen.

Als laatste voorbeeld is er de tijd van het jaar. Een boom ziet er in de zomer heel anders uit dan in de winter. De tijd van het jaar heeft vooral invloed op bladkleur en blad dichtheid.

5.2 Mogelijkheden ter verbetering

Hierboven zijn de belangrijkste inperfecties geschetst. Hieronder zullen enkele mogelijke oplossingen en uitbreidingen globaal geschetst worden.

Als eerste de bladeren. Hiervoor kunnen globale variabelen geïntroduceerd worden, die bijvoorbeeld de dichtheid van bladeren op een tak beïnvloeden en het type blad. Ook kunnen parameters worden toegevoegd aan de tak-operator (F) die bijvoorbeeld specificeren of er bladeren aan een bepaalde tak moeten komen. Dit zou bijvoorbeeld goed gebruikt kunnen worden in combinatie met aanroep condities.

Ook veel van de andere inperfecties in de controle kunnen op soortgelijke wijze afgevangen worden. Zo zouden de texture of kleur voor de takken en de naam voor het te genereren object in de grammatica geplaatst kunnen worden en de in- en output-files opgegeven kunnen worden aan de generator als programma parameters.

De overgangen tussen verschillende takken zouden aangepakt kunnen worden door hier een extra, verpakkende geometrie overheen te plaatsen. Hierbij moet gedacht worden aan methodes als ‘smooth junctions’ en ‘mesh skinning’.

Het toevoegen van natuurlijke invloeden is een lastiger probleem [13]. Dit vergt meer dan alleen het toevoegen van extra globale variabelen. Ook zullen speciale mechanismen, veel complexer dan simpele random-functies, moeten worden toegevoegd die de ‘groei’ van de boom beïnvloeden.

5.3 Conclusie

De bomengenerator die in dit document beschreven wordt, kan redelijk natuurlijke bomen genereren. Maar hij is in geen geval af of compleet. Er zitten nog een aantal inperfecties in, zowel wat betrefd controle, als wat betreft de productie. De inperfecties in de controle zijn over het algemeen gemakkelijk af te vangen en op te lossen. Bij de inperfecties in de productie is dit een stuk moeilijk. Toch zijn ook hiervoor enkele suggesties aangedragen.

6 Appendices

6.1 Appendix A: L-System grammatica

De door ons geconstrueerde parser is automatisch door *bison* en *flex* gegenereerd uit twee invoerbestanden: *trees.y* en *lex.l*. Deze door ons opgestelde bestanden bevatten respectievelijk de grammatica regels voor de L-Systems en te accepteren woorden ("tokens") [9].

De complete grammatica zoals deze door onze parser geaccepteerd wordt staat beschreven in *Trees.y*. Het formaat van het bestand is enigszins cryptisch (zie [9] voor uitleg over de notatie), maar geeft wel enigszins de opbouw van de grammatica weer zoals die geaccepteerd wordt door onze parser:

```
program:
     settings axiom opt_productions;

settings:
     setting settings | ;

setting:
     SET VARIABLE '=' REALCONSTANT ;

opt_productions:
     productions | ;

productions:
     production productions | production;

production:
     production_target opt_targetparameters
     opt_productioncondition ARROW rule ';' ;

opt_productioncondition:
     production_condition | ;

production_condition:
     ':' expr ;

production_target:
     commandname_production ;

opt_targetparameters:
     '(' opt_opt_targetparameters ')' | ;


opt_opt_targetparameters:
     targetparameters | ;

targetparameters:
     VARIABLE | targetparameters ',' VARIABLE ;
```

```
axiom:
     AXIOM ARROW rule ';' ;

rule:
     command rule | command ;

command:
     commandname commandparameters | commandname ;

commandname:
     commandname_turtle | commandname_production ;

commandname_turtle:
     'F' | 'f' | '+' | '-' | '*' | '/' | '&' | '^' | '['| ']' ;

commandname_production:
     'A' | 'B' | 'C' | 'D' | 'E' | 'G' | 'H' | 'I' | 'J' | 'K' |
     'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |
     'V' | 'W' | 'X' | 'Y' | 'Z' |
     'a' | 'b' | 'c' | 'd' | 'e' | 'g' | 'h' | 'i' | 'j' | 'k' |
     'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
     'v' | 'w' | 'x' | 'y' | 'z' ;

commandparameters:
     '(' opt_parameters ')' ;

opt_parameters:
     parameters | ;

parameters:
     expr | parameters ',' expr ;

expr:
       '-' expr
     | '+' expr
     | expr '+' expr
     | expr '-' expr
     | expr '*' expr
     | expr '/' expr
     | '(' expr ')'
     | VARIABLE
     | REALCONSTANT
     | IDENTIFIER '(' opt_parameters ')'
     | expr EQUALS expr
     | expr NOTEQUALS expr
     | expr GREATEREQUALS expr
     | expr SMALLEREQUALS expr
     | expr '>' expr
     | expr '<' expr ;
```

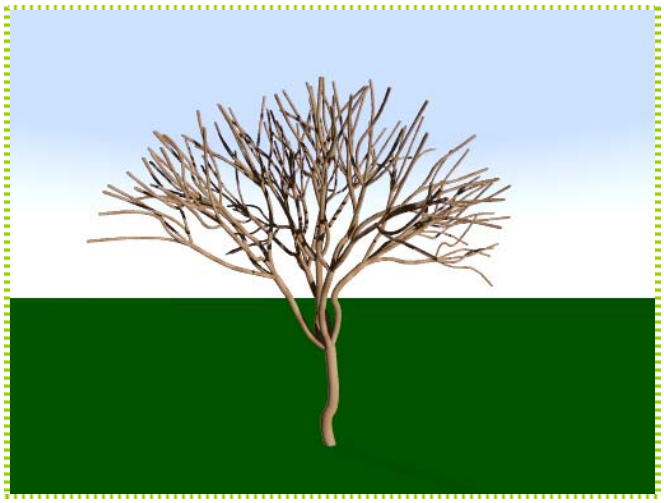
6.2 Appendix B: Inputfile in detail

Om een goede impressie te krijgen van de mogelijkheden voor het modelleren van een boom zullen we in dit paragraaf een inputfile (waarin zo veel mogelijk van de aanwezige features zitten) regel voor regel doorlopen en becommentariëren:

```
01    // settings
02    SET #recursiondepth = 5
03    SET #defaultangle = 30
04    SET #defaultlength = 1
05    SET #defaultradius = 0.10
06    SET #lengthfactor = 0.97
07    SET #radiusfactor = 0.6
08    SET #DefaultBLengthFactorF = 0.3
09    SET #DefaultBRadiusF = 0.1
10    SET #DefaultBLengthFactorS = 0.6
11    SET #DefaultBRadiusS = 0.0
12    SET #UseSweep = 0
13    SET #PI = 3.14159265358979323846264338327795
14
15    // axioma
16    AXIOM -> A(0);
17
18    // production rules
19    A(#i) : #i < 4 -> F(#defaultlength,
20             norm(#DefaultBLengthFactorF, 0.1),
21             norm(#DefaultBRadiusF, 0.05),
22             norm(#DefaultBAngleF, #PI),
23             norm(#DefaultBLengthFactorS, 0.1),
24             norm(#DefaultBRadiusS, 0.1),
25             norm(#DefaultBAngleS, #PI))
26             [+A(#i+1)]
27             [-A(#i+1)]
28             [*A(#i+1)]
29             [/A(#i+1)]
30             A(#i+1)
31    ;
32
33    A(#i) : #i >= 4 -> F(#defaultlength-0.1,
34             norm(#DefaultBLengthFactorF, 0.1),
35             norm(#DefaultBRadiusF, 0.05),
36             norm(#DefaultBAngleF, #PI),
37             norm(#DefaultBLengthFactorS, 0.1),
38             norm(#DefaultBRadiusS, 0.1),
39             norm(#DefaultBAngleS, #PI))
40             [+A(#i+1)]
41             [+&A(#i+1)]
42             [-A(#i+1)]
43             [-&A(#i+1)]
44             [*A(#i+1)]
45             [/A(#i+1)]
46             A(#i+1)
47    ;
```

Regelnr(s)	*Beschrijving*
1	Commentaar mag aanwezig zijn (als enkele-regel-commentaar (//) en als blok-commentaar (/* en */).
2	Dit stelt de maximum recursiediepte in op 5
3-5	De standaard angle wordt 30 graden, de standaard taklengte wordt 1 en de standaard takdikte wordt 0.1
6-7	Een lengtefactor van 0.97 en een takdiktefactor van 0.6 worden gedefineerd.
8-11	Deze twee parameters bepalen de controlpoints van de curve van de takken. Hierbij staat de laatste letter F van de setting voor First, en een S voor second. Opgemerkt kan worden dat er geen waarde ingesteld wordt voor #DefaultBAngleF en #DefaultBAngleS; deze twee settings krijgen als waarde 0 (hun standaard waarde).
12	Voor het genereren van de takken wordt geen gebruik gemaakt van de spheresweep pipeline, in plaats daarvan wordt de snellere cyliner-en-b-splines methode gebruikt.
13	#PI wordt gebruikt in enkele regels, maar heeft standaard geen waarde. Daarom moet deze variabele eerst gedefinieerd worden.
16	Als eerste moet A geëvalueerd worden met als parameterwaarde 0.
19-31	A heeft 1 parameter #i, welke eigenlijk gewoon gebruikt wordt als iteratieteller. Verder heeft A ook een conditie: #i < 4. De commando's voor A zijn eigenlijk slechts één F commando, gevolgd door 5 takken (aanroepen van A), allen aan het uiteinde van de F en allen in een andere richting, waarbij 1 ervan in dezelfde richting doorgaat als de richting waarin F zich bevindt. De tak die F genereert is altijd dezelfde lengte, en de posities van beide controlepunten van de b-spline worden met normaal verdelingen bepaald.
33-46	Deze definitie van A is een uitbreiding op de vorige, met als toevoeging dat de conditie nu veranderd is in: #i >= 4. En tevens worden ipv 5 takken nu 7 takken afgesplitst aan het einde van een vorige tak. De bedoeling van deze uitbreiding is om een complexere takstructuur te verkrijgen aan de uiteinden van de bovenste takken.

Door bovenstaande inputfile te verwerken door onze generator en het daaruit voortkomende bestand door POV-ray te laten renderen wordt het volgende resultaat behaald:

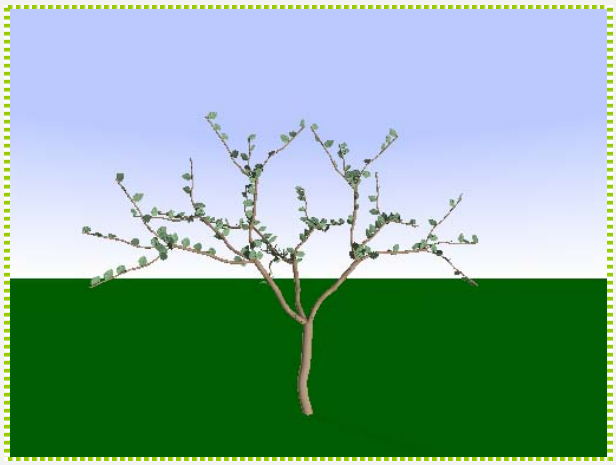


Figuur 6.1: Resulterend model a.d.h.v. inputfile

6.3 Appendix C: Outputfile in detail

In deze appendix zullen de outputfiles van de twee pipelines en de verschillende extra files besproken worden.

Alvorens te beginnen eerst de inputfile, die als uitgangspunt van dit paragraaf gebruikt wordt, samen met een plaatje van de ontstane boom.



Figuur 6.2: De boom die in deze appendix besproken wordt.

```
// settings
SET #recursiondepth = 5
SET #defaultangle = 40
SET #defaultlength = 1
SET #defaultradius = 0.10
SET #lengthfactor = 0.97
SET #radiusfactor = 0.55
SET #DefaultBLengthFactorF = 0.3
SET #DefaultBRadiusF = 0.1
SET #DefaultBLengthFactorS = 0.6
SET #DefaultBRadiusS = 0.0
SET #UseSweep = 0

AXIOM -> A;

A -> F[+A][*A][-A];
```

Om te beginnen zal de output van de spheresweep methode onder de loop genomen worden. Hieronder volgt een fragment uit de gegeneerde code. Om deze te genereren zou hierboven de UseSweep variabele op 1 gezet moeten worden.

```
#declare ProceduralTree =
union {
sphere_sweep {
      b_spline
      6,
      <-0.100000, -0.291000, 0.000000>, 0.055000
      <0.000000, 0.000000, 0.000000>, 0.055000
      <0.100000, 0.291000, 0.000000>, 0.055000
      <0.000000, 0.582000, 0.000000>, 0.055000
      <0.000000, 0.970000, 0.000000>, 0.055000
      <0.000000, 1.358000, 0.000000>, 0.055000
      tolerance 0.1
      material { TreeMaterial }
}

...

object {
      Blad3
      scale 0.249913
      rotate <-49.000000, 0, 0>
      rotate <0, 0, 6.462807>
```

```
      rotate <0, 0.000000, 0>
      translate <0.604799, 1.690771, 0.000000>
}

...
}
```

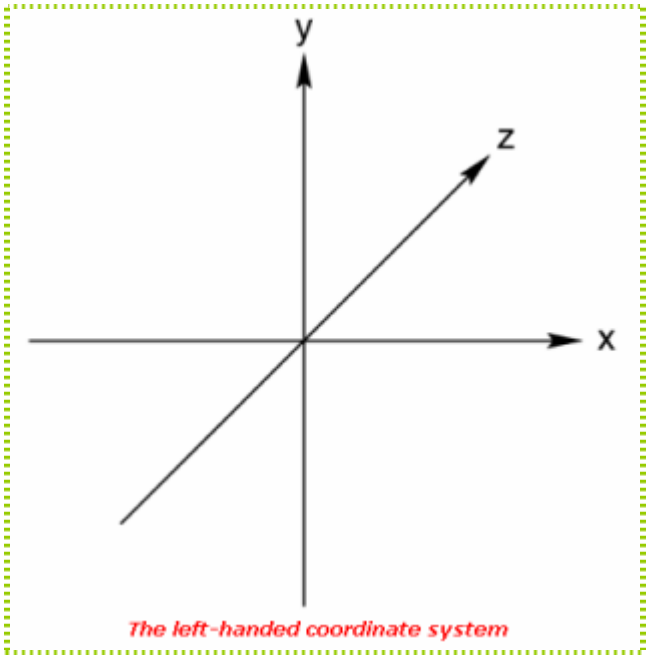
BESTAND: testboom.inc (SphereSweep)

Om te beginnen is er een declaratie. De boom wordt als object gebruikt, dus moet gedeclareerd worden. Dit wordt gedaan met *#declare <naam> = <geometrisch model>*.
Het geometrische model is een samengesteld model, uit veel spheresweeps voor de takken en nog meer object-instanties voor de bladeren. Daarom bestaat het object uit een grote union.

Het belangrijkste onderdeel van de union zijn de spheresweeps. Een spheresweep bestaat uit de volgende onderdelen:

- Een interpolatie functie. De opties zijn linear (recht, door alle punten heen), cubic_spline (een spline door alle punten heen) en b_spline (een kubische b-spline).
- Het aantal controle punten. Voor linear is dit minimaal 2, voor cubic_spline 4 en voor b_spline 6 (zie ook hoofdstuk 4.5.2).
- De controlepunten, bestaande uit een punt <x, y, z> en de straal op dat punt. Zo zou dus ook de straal over een tak gevarieerd kunnen worden.
- De optionele toevoeging tolerance zorgt voor mooiere sweeps, door een tolerantie in de berekeningen van de sweep toe te staan. Zo worden zwarte puntjes op de sweep voorkomen
- Tot slot geeft material de tak het van te voren gedefinieerde materiaal TreeMaterial. De definitie hiervan wordt bij de POV-file besproken.

Naast spheresweeps bevat de union ook per spheresweep een set bladeren. Een blad is hier een instantie van een blad object. Een object instantie wordt gemaakt met *object { <objectname> [transformaties] }*. De objectnaam is hier **Blad3**. Dit verwijst naar een elders gedefinieerd blad. Deze definitie wordt later in deze appendix besproken. De optionele transformaties zorgen ervoor dat het blad het juiste formaat heeft en met de juiste oriëntatie op de juiste plaats komt. Er wordt eerst geschaald. Het blad heeft nu het juiste formaat. Vervolgens wordt er geroteerd over de x-as om het blad voor of achterover te kantelen. Vervolgens wordt het blad over de z-as gedraaid om het blad zijwaards te kantelen. Daarna wordt het blad over de y-as gedraaid om hem over zijn as te draaien. Tot slot wordt het blad met een translatie op zijn plaats gezet. De rotaties vinden in POV-ray altijd globaal plaats om het volgende assenstelsel:



Figuur 6.3: POV-Ray's assenstelsel.

Op deze wijze wordt de gehele boom opgebouwd in de spheresweep methode.

Nu een fragment van de output van de andere methode.

```
#declare ProceduralTree =
union {

sphere {
      <0.000000, 0.000000, 0.000000>,
      0.055000
      material{ TreeMaterial }
}

cylinder {
      <0.000000, 0.000000, 0.000000>,
      <0.051625, 0.185876, 0.000000>,
      0.055000
      material{ TreeMaterial }
}

...

object {
      Blad3
      scale 0.249913
      rotate <-49.000000, 0, 0>
      rotate <0, 0, 6.462807>
      rotate <0, 0.000000, 0>
      translate <0.604799, 1.690771, 0.000000>
}
```

```
...
}
```

BESTAND: testboom.inc (Cilinders en b-splines)

In deze output file is bestaat het ProceduralTree-object wederom uit een union. Ditmaal is deze union gevuld met afwisselend sets van cilinders en bollen en sets van bladeren.

De bladeren worden op dezelfde wijze gegenereerd als bij de sphere sweeps en zullen hier niet wederom besproken worden. De aandacht zal uitgaan naar de takken.

In deze aanpak bestaat een tak niet uit een geometrisch model (de sphere sweep), maar uit een serie van geometrische modellen (cylinders & spheres). Het belangrijkste onderdeel wordt gevormd door de cilinders. Een cilinder bestaat uit:

- Twee punten, namelijk het middelpunt van de bovenkant en het midden van de onderkant, beiden cirkels. Deze cirkels zullen loodrecht op de lijn tussen deze twee punten worden geplaatst.
- De straal. Deze bepaalt de dikte van de cilinder.
- En een materiaal om hem zichtbaar te maken, equivalent aan die van de sphere sweeps.

Het andere geometrische model dat in de takken gebruikt wordt is de bol. Deze bestaat uit:

- Een punt, die het centrum van de bol aangeeft
- De straal.
- En wederom een materiaal.

Deze hiervoor besproken inc-files staan niet op zichzelf. Met alleen deze files kunnen er nog geen bomen gerenderd worden. Er ontbreken namelijk nog twee dingen: De definitie van **Blad3** en een POV-ray bestand met de benodigde overhead en een of meer instanties van het hierboven gedefinieerde object **ProceduralTree**. Deze zullen hieronder achtereenvolgens besproken worden.

Eerst een fragment uit het bestand met de definities van de bladeren.

```
#declare Blad3 =
union {
polygon {
     4, // number of points
     <0, 0>, <0.5, 0>, <0.5, 1>, <0, 1>

     pigment {
          image_map {
               tga "Blad3-32.tga" // the file to read
               map_type 0
               interpolate 4
          }
     }
```

```
      translate x*-0.5
      rotate y*10
}

polygon {
      4, // number of points
      <0.5, 0>, <1, 0>, <1, 1>, <0.5, 1>

      pigment {
            image_map {
                  tga "Blad3-32.tga" // the file to read
                  map_type 0
                  interpolate 4
            }
      }

      translate x*-0.5
      rotate y*-10
}

}
```

BESTAND: blad.inc (blad definitie)

Een blad is wederom een object. Instanties hiervan werden eerder aangemaakt in de output files van de twee methodes. Een blad bestaat uit twee schuin tegen elkaar geplaatste helften, vergelijkbaar met twee bladzijden van een boek. Het object is dus weer een union. De elementen van deze union zijn polygons, 2D elementen die gedefinieerd worden op het (x,y) vlak. Door middel van rotaties kunnen ze anders georiënteerd worden. Beide polygonen hebben vier punten die zowel hun coordinaten op het (x, y)-vlak voorstellen, als de (u, v)-coordinaten van de gebruikte texture. Verder hebben beide polygonen een zogenaamde *pigment*.

Een *pigment* is, naast een materiaal, een wijze om het uiterlijk van een geometrisch object te beschrijven. Voorbeelden van *pigment* zijn kleuren en textures (image_maps). In dit geval is er voor een texture gekozen. Zo'n texture wordt gedefinieerd door middel van een image_map. Hiervoor moet eerst het bestandstype en het bestand opgegeven worden. Hier is gekozen voor een TGA-file, omdat er gebruik wordt gemaakt van een alpha-channel en een 32-bits TGA-file dit ondersteund. Andere opties zijn bijvoorbeeld GIF en JPG. Verder moet er een map_type opgeven worden. De waarde 0 duidt planair aan, dus op een plat vlak. Andere opties zijn bol, cilinder en tours. Verder zijn er verschillende vormen van interpolatie. Opties hierbij zijn geen, lineair, bilineair en genormaliseerde afstand. Hier is voor het laatste gekozen.

Om het blad af te maken worden de twee helften getransleerd zodat het midden op de y-as ligt. Dit is gemakkelijk voor het verdere gebruik en om de kromming (rotatie over de y-as) aan te brengen.

De twee andere blad-types, object **Blad1** en object **Blad2**, zijn op identieke wijze gedefinieerd, uiteraard met andere textures.

Tot slot is er de POV-Ray file, die hieronder is weergegeven.

```
#include "colors.inc"
#include "textures.inc"

global_settings
{
        max_trace_level 5
        assumed_gamma 1
}

camera {location <1.5,1.5,5> look_at <0,2,0>}

light_source {<-140,0,300> rgb <1.0, 1.0, 0.95>*1.5}
light_source {<140,3,-300> rgb <1.0, 1.0, 0.95>*1.5}

sphere {                    // --- Sky ---
     <0, 0, 0>, 1
     texture {
          pigment {
               gradient y
               color_map {
                    [0.0 color rgb < 1.0, 1.0, 1.0 >]
                    [0.3 color rgb < 0.5, 0.6, 1.0 >]
               }
          }
          finish { diffuse 0 ambient 1 }
     }
     scale 10000
     hollow on
     no_shadow
}

plane { y, 0
     pigment { color Green }
}


#declare TreeMaterial=material {texture {DMFLightOak rotate x*90 } }

#include "blad.inc"
#include "testboom.inc"

object { ProceduralTree }
```

BESTAND: testboom.pov (de te renderen scene)

Dit is de definitie van de eigenlijke scene. Bovenaan worden twee files ingevoegd, met standaard definities van kleuren en materialen. Daarna volgen enkele standaard instellingen, een camera en twee lichtbronnen.

Vervolgens begint de daadwerkelijke scene. Eerst wordt er een grote bol gedefinieerd, die alles omgeeft en de lucht voorstelt. Daarna volgt een eenvoudig grondvlak, die enkel een kleur heeft. De boom hangt nu niet los in het niets.

Na deze overhead volgt het plaatsen van de boom. Eerst worden eenmaal de bladdefinities en de boomdefinitie ingevoegd. Dit had ook bovenin het bestand gekund, maar zo worden overhead en bomen van elkaar gescheiden. Tot slot wordt er één definitie van een boom geplaatst. Deze boom heeft geen verdere transformaties. Als er meer dan een boom geplaatst zou worden (meer instanties) zouden in de instantie van het object ook translaties, rotaties en schalingen geplaatst kunnen worden, evenredig aan die van de bladeren in de boom-definitie.

Dit beschrijft te gehele output en de overige benodigde bestanden om de getoonde scenes te kunnen renderen.

6.4 Appendix D: Referenties

1 P. Prusinkiewicz, M. Hammel, J. Hanant en R. Mech. **Visual models of plant development**. In G. Rozenberg en A. Salomaa, editors, *Handbook of formal languages*, Springer-Verlag, 1996.

2 Matthew Holton. **Strands, gravity and botanical tree imagery**. *Computer Graphics Forum*, 13(1):57-67, March 1994.

3 Helge von Koch. **Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes**, Acta Mathematica, Stockholm, 1906 (30), pp. 145-174.

4 The Logo Foundation, **http://el.media.mit.edu/logo-foundation/**.

5 Persistence Of Vision Raytracer, **http://www.povray.org**.

6 GNU Bison, **http://www.gnu.org/software/bison/**.

7 GNU Flex, **http://www.gnu.org/software/flex/**.

8 Agner Fog, **Pseudo Random Number generators** (http://www.agner.org/random)

9 Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs and Koen G. Langendoen, **Modern Compiler Design**. John Wiley and Sons, October 2002.

10 H.P. Lophuaä, **Statistiek voor studenten informatica**. 17 januari 2001

11 W.F. Bronsvoort, A. Noort, F.H. Post, J.S.M. Vergeest, **Geometrisch Modelleren**. Deel 2, september 2000, pp. 1-42.

12 J.J. Van Wijk, **Ray tracing of objects defined by sweeping a sphere**. *Computers and Graphics*, vol. 9, no. 3, 1985. pp. 283-290.

13 Radomir Mech and Przemyslaw Prusinkiewicz, **Visual Models of Plants Interacting with Their Environment**. University of Calgary, Proceedings of SIGGRAPH 1996. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 397-410.

14 P. Prusinkiewicz, M. Hammel, J. Hanant en R. Mech. **L-Systems: from the theory to visual models of plants**. In M.T. Michalewicz, editor, *Proceedings of the 2nd CSIRO Symposium on Computational Challanges in Life Sciences*. CSIRO Publishing, 1996.