Realtidsrendering – Lindenmayers systemer

Eksamensprojekt

Udarbejdet af:

Mikkel Vagtborg, s991344
mjvagtborg@hotmail.com

Rapporten indeholder i alt 83 sider.
5 appendikser:

Appendiks A:	4 sider
Appendiks B:	132 sider
Appendiks C:	5 sider
Appendiks D:	7 sider
Appendiks E:	3 sider

1 CD

Vejledere:

Niels Jørgen Christensen
Bent Dalgaard Larsen

Danmarks Tekniske Universitet
Informatik og Matematisk Modellering
DK 2800 - Kongens Lyngby
Bygning 321
reception@imm.dtu.dk
http://www.imm.dtu.dk

1 Abstract

Trees and other plants in computer applications, mainly computer games are often very similar. One of the reasons for the similarity is due to each of the plants often is modeled in an editor. Different looking plants require each model to be modeled independently, which is time consuming. Therefore one plant is simply copied to many locations.

The purpose of this report is to present a method that utilizes Lindenmayer's systems by extending this system with additional symbols. Due to models of plants contain a many polygons it is not possible to render the polygon model in real time. Therefore the method includes the generation of the imposter for the real polygon model during execution of the application in which the plant is inserted. To reduce the render time of the imposters the method also includes Level of Detail in the imposters.

Keywords: Lindenmayer's systems, L-systems, Level of Detail, Generating imposter online

2 Resumé

Træer og andre planter i computerapplikationer, specielt computerspil er ofte meget ens. En af grundene til denne ensartethed skyldes at planter ofte er modelleret i en editor. Planter med forskellig udseende kræver at hver model bliver modelleret separat, hvilket er tidskrævende. Derfor bliver en plante simpelthen kopieret til flere lokationer.

Målet med denne rapport er at præsentere en metode, der udnytter Lindenmayers systemer ved at udvide dette med ekstra symboler. Da modeller af planter indeholder mange polygoner er det ikke muligt at rendere polygonmodellen i realtid. Derfor inkluderer metoden generering af erstatningsmodellen for den rigtige polygonmodel under eksekvering af det program, hvor planten er indsat. For at reducere renderingstiden af erstatningsmodellen inkluderer metoden også Level of Detail i erstatningsmodellen.

Nøgleord: Lindenmayers systemer, L-systemer, Level of Detail, Online generering af erstatningsmodel

3 Problemformulering

Formålet med dette projekt er at undersøge om det er muligt at udvikle et system, der benytter Lindenmayers systemer til at generere modeller af planter, som kan benyttes i applikationer. Et af kravene til systemet er at det kan eksekveres i realtid.
Generering af planter indenfor computergrafik sker ofte ved at planterne bliver modelleret i en applikation og derpå importeret til det program, hvor de skal benyttes. Dette medfører at de planter der optræder i en scene ofte er det samme plante, der er kopieret til forskellige lokationer.
I dette projekt vil jeg undersøge om det er muligt at generere planter, som ikke blot kan vises i realtid men også om det er muligt at generere planter der på baggrund af en model kan variere i udseende.

4 Indholdsfortegnelse

5 Indledning

Lindenmayer systemer blev introduceret af Aristid Lindenmayer i 1968. Lindenmayer havde en stor interesse for hvordan planter udvikler sig. Dette ønskede han at kunne beskrive matematisk. Systemet blev navngivet efter ham og kom derfor til at hedde Lindenmayer-systemer, eller forkortet L-systemer. L-systemer er et regelbaseret system, der er udviklet til at beskrive hvordan celler og planter udvikler sig.
Planter indeholder store mængder af detaljer, hvor flere af detaljerne kan beskrives ved hjælp af matematik, f.eks. placerer frøene i blomsten på en solsikke sig i matematiske mønstre.
Ulempen ved disse systemer er at modellerne vil komme til at indeholde meget information før man får genereret en plante, der ser realistisk ud.

Metoden, som beskrives i rapporten skal ses som et alternativ til den måde man i nuværende spil og programmer genererer træer/planter på. Denne metode giver mulighed for at man kan generere varierende træer, så det ikke er den samme model, der bliver tegnet flere gange forskellige steder i scenen.

Med metoden lægges der vægt på, at det skal være nemt for en bruger at integrere det i sit program, dvs. at en bruger skal have nem adgang til selv at tilføje regler til systemet.

6 Litteratur gennemgang

Til at generere træer som skal benyttes i et større terræn findes der flere metoder. En af de nyeste metoder er SpeedTreeRT®, som er udviklet af Interactive Data Visulization [IDV]. IDV er blevet valgt som software leverandør til den nye generation af Microsofts Xbox®. Denne version går under navnet Xbox 360. SpeedTree er et software bibliotek, der er rettet mod at kunne generere realistiske træer i realtid. Der benyttes en editor som SpeedTreeCAD til at lave modellerne, som så kan importeres i en applikation ved hjælp af SpeedTreeRT. Motoren er rettet mod et lavt processor forbrug ved at benytte vertex buffers og andre GPU funktionaliteter. Yderligere er der også implementeret en fysik motor i SpeedTreeRT, så f.eks. vind dynamisk kan påvirke træerne. SpeedTree benytter et polygon mesh til de grove dele af træet og til blade benyttes der billboards. LoD er også med i SpeedTree og det fungerer ved at transformere mere og mere af træet til billboards, når afstanden til træet øges. Bark bliver lavet ved bump mapping, og modellen indeholder statiske selvskygger.

Billboards er en meget anvendt metode til at repræsentere modeller med. Den metode er nok den mest benyttede metode inden for spilverdenen. Problemet med denne metode er at finde ud af hvor billboards skal placeres for at modellen ser bedst ud. I [BCEMS] og [EMSFR] bliver der diskuteret billboard skyer, som er en metode, der selv finder ud af hvor billboards skal placeres. Ideen i denne metode er at modellen bliver projiceret ind på en stor mængde af billboards, som skal placeres på en bestemt måde så det ikke er muligt at se at det ikke er den rigtige polygon model der benyttes. Denne metode kræver en del søgning og sortering, så modellerne bør genereres offline. Det giver nogle ret gode renderingstider. I [EMSFR] er der en scene med 150.000 træer, som dog kun kan vises i deres test med 2-6 fps., men det må siges at være ret imponerende alt taget i betragtning af, hvor mange træer der er i scenen.

I [TfP] benyttes der en 3. metode til at lave modeller af træer. Den tager udgangspunkt i et sæt af fotografier af et virkeligt træ. Dette sæt af fotografier bliver benyttet til at generere teksturer til et sæt af billboards. I [TfP] benyttes der en 3D rekonstrueringsalgoritme til at generere sættet af teksturer. Denne metode, hvor et 2D billede transformeres til en 3D model, er en ret svær opgave, da man f.eks. mangler dybde information. Der benyttes 20-30 fotos af hvert træ for at kunne skabe en 3D model. Der er mange faktorer der spiller ind i denne metode. For at kunne identificere træet er det vigtigt at det skiller sig ud fra baggrunden. Fra 3D modellen skabes der et sæt af teksturer som benyttes på billboards.

I [PbI] bliver der introduceret en metode, hvor der udvælges en række punkter fra en model og så benytter man de punkter som en erstatning for den rigtig model. Denne metode kaldes punktskyer. Der er flere metoder til at identificere, hvilke punkter der skal benyttes. I [PbI] anvendes en metode der benytter dybde bufferen, andre metoder kan være at benytte en ray-tracer. Punktskyer understøtter også LoD ved at variere antallet af punkter, der benyttes som funktion af afstanden. Metoden der er implementeret i [PbI], er ret hurtig.

Der er flere måder at lave LoD på. I [MoD] diskuteres en metode til generering af meshs, hvor der er indbygget LoD. Alt efter hvilken afstand man er i, så vælger man flere punkter ud på meshet så det bliver mere detaljeret når man er tæt på. Denne metode vil ikke kun virke med modellerne af planter, da der er alt for mange polygoner i dem, at den mest detaljeret version af modellen ikke kan vises i realtid.

L-systemer er beskrevet meget udførligt i "The Algorithmic Beauty of Plants" [ABoP] af Przemyslaw Prusinkiewicz og Aristid Lindenmayer, som nok er de 2 personer der har beskæftiget sig mest med L-systemer. L-systemer er udviklet til at beskrive hvordan celler og planter udvikler sig. Systemet er et regelbaseret system, hvor det er muligt at tilføje tilfældigheder til systemet, for at skabe forskellige modeller ud fra samme regelsæt.

Przemyslaw Prusinkiewicz har været med til at skabe et modellerings sprog kaldet: L+C, som bliver præsenteret i [LC] og er implementeret i C++. Dette system kan benyttes til at fortolke L-systemer. Programmet er dog et kommercielt program så derfor vil det ikke blive benyttet i rapporten.

I forbindelse med et projekt ved instituttet IMM ved DTU, har en studerende lavet en poster [Poster] som diskuterer hvordan man kan benytte L-systemer til at generere træer og hvordan man kan benytte teksturificering til at generer en erstatningsmodel for den originale model.

Et generelt problem med modeller der er genereret med L-systemer er, at de ikke ser realistiske ud. Der er flere metoder til at gøre modellerne mere realistiske. I [LSC] introduceres en metode hvor der indføres subdivision til L-systemer. Dette kan være interessant, da modeller der laves med L-system har en tendens til at blive meget kantede. Ser man f.eks. på en sidegren så vil den stå helt skarpt på den gren, som sidegrenen vokser ud fra. Det vil ikke virke særligt naturtro. Subdivision vil kunne afhjælpe dette ved at gøre overgangen mere jævn.

Det er også muligt at lave nærmest fotorealistiske billeder som det er gjort i [MHP]. Billederne i denne artikel er utrolig flotte. De planter, der er modelleret bliver sammenlignet med fotografier af planter i naturen, og det er næsten ikke til at se forskel. Desværre betyder det også, at modellerne kommer til at indeholde alt for meget information til at kunne renderes hurtigt. I den omtalte artikel bliver der indsat 65.000 små hår for at modellere en krokusblomst.

[MPGD] handler om hvordan man kan indarbejde de fysiologiske processer, der foregår i en plante når den gror. Det er lidt det samme der tages op i [SMPPE], hvor man tager højde for lysforhold, hvilket gør at områder af planten, der er i skygge vokser langsommere end områder der bliver belyst.

7 Terminologi og Notationer

Da mange af udtrykkene inden for området ikke har en korrekt oversættelse til dansk har jeg valgt at benytte det engelske udtryk.

7.1 En turtle.

Når et L-system oversættes fra en streng af symboler til en geometrisk model så benytter man en turtle. Denne turtle består at en position (x, y, z) samt 3 enhedsvektorer $\vec{H}, \vec{L}, \vec{U}$ som holder styr på turtlens orientering i rummet. $\vec{H}, \vec{L}$ og $\vec{U}$ står parvis vinkelret på hinanden og opfylder ligningen $\vec{H} \times \vec{L} = \vec{U}$.

7.2 Teksturificering

Udtrykket teksturificering dækker over den proces, hvor man transformere en polygon model om til et sæt af teksturer, der benyttes i renderingsfasen i stedet for polygon modellen.

7.3 Rendering i realtid

Hvad der forbindes med realtids rendering er at programmet kan holde en fast framerate. Det er svært at sætte en fast grænse for hvornår antallet af frames per sekund er højt nok til at være realtid, men den mest anvendte grænse er 17-18 fps. så den vil jeg benytte i rapporten.

7.4 Online/offline rendering

Med online rendering menes der, at genereringen af erstatningsmodellen sker under eksekvering af applikationen.
Offline rendering betyder at erstatningsmodellen er genereret på forhånd f.eks. ved opstart af applikationen eller er genereret af en anden applikation, hvor resultatet så bliver læst ind i applikationen.

7.5 Level of Detail

Level of Detail, LoD, betyder at antallet af detaljer i et objekt bliver tilpasset til antallet af pixel som objektets projicering optager. Ses objektet tæt på, så er detaljeringsgraden høj, mens på afstand bliver detaljeringsgraden reduceret for at øge renderingshastigheden af objektet. På stor afstand falder flere detaljer sammen i en pixel, så de bliver unødvendige.
I denne rapport vil det mest detaljeret niveau blive kaldt LoD 1. Mindre detaljeret niveauer vil bliver betegnet med højere værdier.

7.6 Billboard

Et billboard er et rektangulært primitiv, som bliver benyttet i stedet for det rigtige objekt der skulle være på denne lokation. På billboardet bliver der lagt et billede af objektets projicering. Dele af primitivet kan gøres gennemsigtigt.

7.7 Symboler

Følgende symboler vil blive benyttet i rapporten:

M: Tilstandsmatricen, som er en 4x4 matrice i homogene koordinater.

T(h,l,u): Translation matricen, hvilket er en matrice der se således ud:

$$T(h,l,u) = \begin{bmatrix} 1 & 0 & 0 & h \\ 0 & 1 & 0 & l \\ 0 & 0 & 1 & u \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_H(v): Rotationsmatricen til rotation omkring H-aksen:

$$R_H(v) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(v) & -\sin(v) & 0 \\ 0 & \sin(v) & \cos(v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_L(v): Rotationsmatricen til rotation omkring L-aksen:

$$R_L(v) = \begin{bmatrix} \cos(v) & 0 & \sin(v) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(v) & 0 & \cos(v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R_U(v): Rotationsmatricen til rotation omkring U-aksen:

$$R_U(v) = \begin{bmatrix} \cos(v) & -\sin(v) & 0 & 0 \\ \sin(v) & \cos(v) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8 Analyse

For at kunne løse opgaven, der er beskrevet i problemformuleringen, skal der udarbejdes en metode til at indbygge LoD i L-systemer.
Der skal opfindes en metode til hvordan denne erstatningsmodel skal genereres, samt der skal findes en metode til at bestemme niveauet af detaljer, der skal anvendes som funktion af afstanden.

Erstatningsmodellen kan genereres på to tidspunkter, enten kan man vælge at pre-rendere den, hvilket koster tid under opstart af applikationen eller også kan man generere den online mens applikationen kører. Dette giver den effekt at applikationen starter hurtigt op, men da modellen først kan vises når den er genereret så det vil ikke betyde så meget. Derimod kunne det tænkes at modellen ændrer sig over tid og så vil det være nødvendigt at kunne generere den online.

Generering af en erstatningsmodel vil benytte den grafiske pipeline på grafikkortet, som også benyttes til at tegn scenen som vises på skærmen. Derfor skal der udarbejdes en algoritme, der kan dele denne pipeline uden at brugeren bliver påvirket af processen der genererer erstatningsmodellen.
Princippet i algoritmen skal være at der vælges en fast framerate til applikationen. Denne framerate skal vælges tilpas, så renderingen af scenen ikke tager alt CPU/GPU tiden. Den overskydende tid skal så afsættes til processen der generere erstatningsmodellen.

En erstatningsmodel kan ikke generes på en gang, da dette tager for lang tid og vil blokere for rendering af den scene, der skal vises på skærmen. Derfor skal processen, der generere erstatningsmodellen hele tiden holde styr på, hvor langt algoritmen er nået og kunne pause den så der kan skiftes mellem at tegne den rigtige scene og generering af erstatningsmodellen.

I litteratur kapitlet blev der diskuteret flere metoder til at generere træer på. Flere af dem kræver at der enten modelleres planter i en editor eller at der hentes data ind på en anden måde. L-systemer udskiller sig ved at det kun er et sæt af regler der skal til at generere planter og derfor er det ikke nødvendigt med en editor. L-systemer er også den eneste metode, som kan generere flere træer med forskelligt udseende ud fra et sæt regler. Ud fra dette synspunkt vurdere jeg at L-systemer er den eneste metode til at løse opgaven, der er stillet i problemformuleringen.

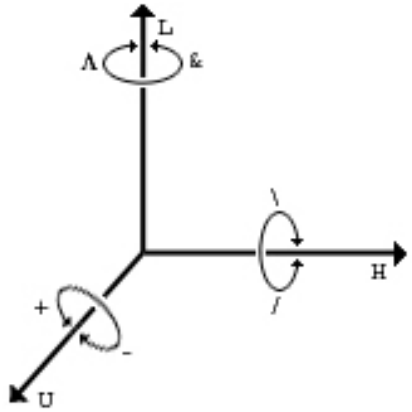
Af de metoder, der er diskuteret i litteratur afsnittet, der benyttes til at lave erstatningsmodeller, vurderer jeg at en erstatningsmodel bedst vil kunne genereres ved hjælp af billboards. Dette begrundes med at systemet skal være så ressource besparende så muligt, så modellen kan genereres løbende under eksekvering af applikationen. LoD skal implementeres på en måde der svare til metoden, der er omtalt i forbindelse med SpeedTreeRT, hvor der benyttes færre og færre billboards når afstanden til modellen øges.

9 L-systemer generelt

Kort kan L-systemer beskrives som et system, der består af et aksiom og et sæt regler for hvordan elementerne i aksiomet skal transformeres.
L-systemer bygger på et princip kaldet tegn genskrivning. Et tegn bliver ved hjælp af produktionsregler lavet om til en sekvens af tegn.
For at simulere et træ der vokser eller bakterier der gror, skal reglerne påføres parallelt.
Denne sekvens af tegn skal oversættes til en grafisk repræsentation. Til denne oversættelse benyttes en turtle som oversætter sekvensen med et tegn af gangen. En turtle er beskrevet i terminologi afsnittet. Den består af en tilstandsmatrice *M*, der er sammensat af turtlens 3 akser, samt dens position, kaldet *P*.

$$M = \left[\vec{H} \quad \vec{L} \quad \vec{U} \quad \vec{P}\right]$$

De tre akser er tegnet på figur 1, sammen med de basale rotationer, som vil blive forklaret efterfølgende.



Figur 1: Turtlen

Når turtlen traverserer hen over det ord der er genereret ved hjælp af produktions-reglerne bliver dens tilstandsmatrice opdateret.

9.1 Gennemgang af basis symbolerne i L-systemer

Flere af symbolerne i L-systems lægger op til at kunne fortolkes på flere måder. Derfor følger der en gennemgang af basis elementerne, som jeg har fortolket dem på, samt hvordan de påvirker tilstandsmatricen *M*:

F(*h*) Flyt turtlen længden *h* ud af *H*-aksen mens der tegnes en linie, med undtagelse af at hvis man befinder sig mellem { } så skal der ikke tegnes, men derimod gemmes det punkt man når frem til i det pågældende polygon, jvf. { } nedenfor.
F-elementet kan også benyttes uden at angive en parameter. Hvis dette gøres så vil det blive antaget, at *h* = 1.

$$M = M \cdot T(h,0,0)$$

f(h) Flyt turtlen længden h ud af H-aksen uden at der tegnes et segment. Angives der ikke et h, så antages det samme som med F, at $h = 1$.

$$M = M \cdot T(h,0,0)$$

+(v) Drej turtlen til venstre. Rotation omkring dens U-akse

$$M = M \cdot R_U(v)$$

-(v) Drej turtlen til højre. Rotation omkring dens U-akse (er lig med +(-v))

$$M = M \cdot R_U(-v)$$

&(v) Pitch turtlen ned af. Rotation omkring dens L-akse

$$M = M \cdot R_L(v)$$

∧(v) Pitch turtlen op af. Rotation omkring dens L-akse (er lig med &(-v))

$$M = M \cdot R_L(-v)$$

\(v) Rul turtlen til venstre. Rotation omkring dens H-akse

$$M = M \cdot R_H(v)$$

/(v) Rul turtlen til højre. Rotation omkring dens H-akse (er lig med \(-v))

$$M = M \cdot R_H(-v)$$

| U-vending, retningen af turtlen drejes 180°

$$M = M \cdot R_U(180)$$

$ Retter turtlens retning ind efter en vektor. Dette element opdaterer tilstandsmatricen så $\vec{L}$ kommer til en vandret orientering. I den oprindelige fortolkning blev den rettet ind efter en vektor der peger modsat tyngdekraften. I min fortolkning har jeg valgt at udvide den til en hvilken som helst vektor, så det også er muligt at få en plante til at rette sig ind mod lyskilder. Vektoren er statisk for hele modellen og bliver kaldt 'V'. Turtlens position bliver ikke påvirket af $-operationen. Positionen bliver kaldt P og er 4. søjle i M.

$$\vec{L} = \frac{\vec{V} \times \vec{H}}{\left|\vec{V} \times \vec{H}\right|} \quad \text{og} \quad \vec{U} = \vec{H} \times \vec{L}$$

$$M = \left[\vec{H} \quad \vec{L} \quad \vec{U} \quad \vec{P}\right]$$

[Gem den nuværende tilstand.
Påvirker ikke M

] Vend tilbage til tilstanden før den tilhørende [.
Påvirker ikke M

{ Påbegynd et polygon, alle punkter i polygonet får den farve der er sat inden {.
Påvirker ikke M

G(h) Avancer h i H-retningen uden at der gemmes et punkt i polygonet. Derimod skal der tegnes et segment. Hvis h ikke angives antages det at $h = 1$.
$M = M \cdot T(h,0,0)$

. Gem nuværende punkt i det nuværende polygon (kan ikke forekomme uden for en { }).
Påvirker ikke M

} Afslut polygon og vend tilbage til det tidligere polygon hvis der findes et.
Påvirker ikke M

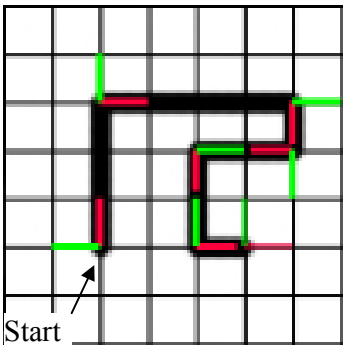
~(*obj*) Tegn objekt *obj* der er en reference til et objekt, der skal tegnes på den position, som turtlen står på når den kaldes.
Påvirker ikke M

!(s) Ændre bredden af segmenterne efter dette tegn. Angives der ikke en s værdi, så bliver bredden af grenene på pågældende tidspunkt for ! ganget med 0.9. Værdien 0.9 er jeg fundet frem til ved forsøg med forskellige modeller.
Påvirker ikke M

’(c) Ændre den diffuse farve på segmenterne efter dette tegn. c er en vektor med de 3 farvekanaler rød, grøn og blå. Angives det ikke en c, så bliver de 3 farvekanaler for den pågældende farve på tidspunktet for ’ multipliceret med 1.2. 1.2 er jeg fundet frem til ved at teste med modeller der benytter ’ uden en c vektor.
Påvirker ikke M

% Afskær alle tegn efter dette indtil der nås en]
Påvirker ikke M

Et simpelt eksempel i 2D, hvor turtlen bliver vist i startpositionen, efter hver af rotationerne og ved dens placering til sidst. Den røde akse er turtlens *H*-akse og den grønne er *L*-aksen.



Figur 2: 2D L-system

Det ord der benyttes til at tegne den sorte streg på figur 2 er:

FFF-FFFF-F-FF+FF+F

En model som bliver genereret med L-systemer består af 3 grafisk del mængder. Den første del mængde bliver kaldt grene og det er de linier der bliver genereret ved hjælp af F- og G-elementerne. Den anden delmængde er alle polygonerne og den sidste er alle de objekter der benyttes i forbindelse med ”~”-elementet. Denne sidste mængde vil blive kaldt for objekterne.

10 L-systemer formelt

For senere i denne rapport at kunne påvise hvordan udvidelserne, der er udviklet i projektet, påvirker L-systemer vil der i dette afsnit være en formel beskrivelse af L-systemer. Kapitel 1.2 i [ABoP] er benyttet som grundlag for dette afsnit da kapitel 1.2 omhandler den formelle beskrivelse af L-systemer.
Der er 2 trin i L-systemer, hvor det første trin er at udvikle aksiomet efter produktionsreglerne. Anden del består i at oversætte resultatet af del 1 til en grafisk fortolkning.

1. del af L-systemer kan beskrives ved hjælp af formelle sprog. Der er flere versioner af L-systemer, men den mest basale version er den deterministiske kontekst frie version af L-systemer. Derfor vil gennemgangen koncentrere sig om denne. Notationen der benyttes i dette afsnit er den samme som i [ICT].

Σ betegner et alfabet, og Σ^* betegner mængden af alle ord der kan genereres ved hjælp af elementerne i Σ, Σ^+ er alle ikke tomme ord over Σ. Et ord, G, består af 3 elementer: $G = <\Sigma, \omega, P>$. ω er et aksiom, for hvilket det gælder at $\omega \in \Sigma^+$ og P er en endelig mængde af produktionsregler. (Teoretisk set er det ikke noget i vejen for at ω er den tomme streng, men det pågældende L-system vil altid være lig med den tomme streng). For produktionsreglerne gælder det at $P \subset \Sigma \times \Sigma^*$. Hvis der ikke findes en produktion for et element så antages det at der er en produktion, der transformerer elementet til sig selv.

I det efterfølgende er der benyttet ( ) i stedet for de normale mængde tegn { }, som [ICT] benytter, da { og } også er bogstaver i alfabetet.

Alfabetet ser således ud:

Σ = (F f + - ∧ & \ / | $ [ ] { G . } ~ ! ’ %)

Et eksempel på et L-system er:

Aksiom:
ω = F

Produktion:
F → FF

Dette giver følgende udvikling over 3 iterationer:

F → FF → FFFF → FFFFFFFF

Antallet af F’er vil altid fordoble for hver iteration.
Hvis man ønsker et system der kun skal forøge antallet af F’er med 1 pr gang så bliver det straks noget sværere at generere. Dette kan løses ved f.eks. at udnytte %-operatoren:

ω = %
% → F%

Dette giver følgende udvikling:

% → F% → FF% → FFF%

Her ender man op med at have et hængende % som i dette tilfælde ikke har nogen effekt da der ikke er noget at fjerne efter %, jvf. definitionen af % .

Imidlertid er dette en uheldig måde at definere regler på, da det kunne være at der efter F'erne havde været nogle symboler noget der ikke skulle fjernes og så er man nødt til at gøre noget for at undgå dette. For at komme uden om dette problem benytter man følgende:
Man indfører et ekstra bogstav til Σ, som har den egenskab at når ordet oversættes vil man ignorere dette bogstav. Derved kan man opnå at definere simplere regler.
Eksemplet fra før skrives så som:
Man tilføjer bogstavet T til Σ og så ser produktions reglerne således ud:

ω = T
T → FT

Dette giver følgende udvikling:

T → FT → FFT → FFFT

Det giver det samme som før men denne gang har man ikke % med, og hvis man har nogle symboler efter F så vil de ikke blive påvirket af T.

Der er nogle betingelser for hvordan et ord kan se ud. F.eks. må der ikke være en], som ikke har en matchende [. Det samme gælder for } og {.

Måden man kan lave en kontekst fri grammatik, der overholder disse regler er at man har en produktion der hedder "P → [T]", hvor T kan generer alle elementer på nær "[", "]", ".", "{" og "}".

Det samme kan man gøre med "{" og "}" og derfor bliver P-produktionen udvides til:
"P → [T] , {D}", her kan man dog ikke benytte T mellem "{" og "}", da det også skal være muligt at have "." mellem disse tegn.

Så der skal være en produktionsregel der kan lave D om til T'er og "."'er. "D → DT.TD".

Det er dog også muligt at kombinere "[", "]", "{" og "}" så de bliver flettet ind i hinanden på følgende 4 måder:
[{ }] [{] } { [ ] } { [}]

Derfor skal P-produktionen også inkludere disse situationer.

Disse regler vil give et L-system hvor man kan udvikle alle tilladte ord.

$\omega = T$

$T \rightarrow TFT , TfT , T+T , T-T , T\wedge T , T\&T , T\backslash T , T/T , T|T , T\$T , T\sim T , T!T , T'T , T\%T , TPT , \Lambda$

$D \rightarrow T, DT.TD , \Lambda$

$P \rightarrow [T] , \{D\}, \{D[D\}T], [T\{D]D\}$

10.1 Stokastiske L-systemer

Som tidligere nævnt så findes der flere versioner af L-systemer, den første variant er stokastiske L-systemer. Denne variant omtales i [ABoP] afsnit 1.7. Her indføres der en sandsynlighed for hvilken retning en produktion kan tage. F.eks.

$\omega = A$

$A \xrightarrow{50\%} A$
$A \xrightarrow{50\%} B$

Når produktions reglerne anvendes, så er der 50% chance for at et A forbliver et A og 50% chance for at det bliver til et B.
For et ord G gælder det at $G = <\Sigma, \omega, P, \pi>$, hvor Σ, ω og P har samme definition som tidligere. π er en sandsynlighedsdistribution som gør at produktionerne blive til produktionssandsynligheder. Har man et element $a \in \Sigma$, så skal det gælde at alle sandsynligheder for de produktioner der transformerer *a* skal summere op til 1.
Denne version af L-systemer vil være oplagt at anvende til generering af modeller med forskellige udseender på baggrund af et sæt af produktionsregler.

10.2 Kontekst følsomme L-systemer

Afsnit 1.8 i [ABoP] omhandler denne version af L-systemer. Kontekst følsomme L-systemer benytter elementerne omkring det betragtede element til at finde ud af hvilken produktion der vælges. Denne version benyttes til at simulere hvordan forskellige dele af en plante påvirker hinanden. Der er 2 versioner af kontekst følsomme L-systemer, 1L og 2L. 1L er L-systemer der afhænger af enten elementet før eller efter det betragtede element. 2L afhænger af elementet før og elementet efter det betragtede.

1L systemer hedder altid:

$a_v < a \rightarrow x$, *a* skal optræde efter a_v

eller

$a > a_h \rightarrow x$, *a* skal optræde før a_h

Det er *a* der transformeres til *x*. Elementet der betragtes kan altid identificeres ved at det er det element der står på den åbne side af > eller <.

I 2L systemer er produktionerne opbygget på følgende måde:

$$a_v < a > a_h \rightarrow x.$$

Et eksempel på et sådan L-system:

ω = AABAA

B < A → C

Ved at benytte produktionen én gang:

AABAA → AABCA

10.3 Parametriske L-systemer

Motivationen til at indfører parametre til L-systemer er at det ikke er alle dele af en plante der vokser med en konstant enhed hver gang. Det kan være muligt at sætte flere F-elementer sammen så dele af planten kommer til at vokse med forskellig hastighed. Men det kunne ske at en del skulle f.eks. vokse med en længde på $\sqrt{2}$, hvilket ikke ville være muligt at angive, uden muligheden for at kunne angive, hvor langt turtlen skal bevæges ved en F-operation, ved hjælp af en parameter. Parametriske L-systemer står beskrevet i [ABoP] afsnit 1.10. Parametrene kan også benyttes til at tage beslutning om en produktion skal benyttes eller ej.

Et element $A \in \Sigma$ med parametrene $a_1, a_2, \ldots, a_n \in \Re$, kaldes et modul og betegnes med $A(a_1, a_2, \ldots, a_n)$. For alle moduler gælder det at de tilhører sættet $M = \Sigma \times \Re^*$, hvor $\Re^*$ er sættet af alle endelige sekvenser af parametre.

Mængden af formelle parametre betegnes med Ψ. C(Ψ) betegner et logisk udtryk med parametre fra Ψ og E(Ψ) er et aritmetisk udtryk med parametre fra Ψ.

Den formelle beskrivelse af et ord G der bliver genereret ved hjælp af parametriske L-systemer er: G = <Σ, Ψ, ω, P>, hvor Σ er samme alfabet som tidligere.

ω er et aksiom, $\omega \in (\Sigma \times \Re^*)^+$.

Produktionerne P er nu blevet noget mere avanceret:

$$P \subset (\Sigma \times \Psi^*) \times C(\Psi) \times (\Sigma \times E(\Psi))^*$$

Et eksempel på et parametrisk L-system:

ω = A(5)B(2,3)

Produktionsregler:

A(t) : t > 2 → B(t+1,3)
B(t, s) : t + s > 4 → A(4)

Dette vil give udviklingen:

A(5)B(2,3) → B(6,3)A(4)

10.4 Parametriske 2L-systemer

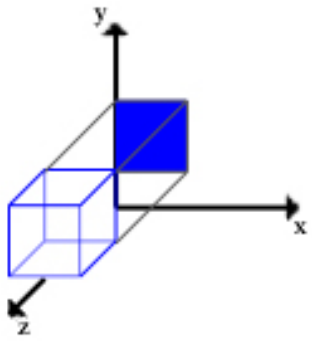
Hvis man kombinerer kontekst følsomme L-systemer med parametriske L-systemer så fås parametriske 2L-systemer. Der vil ikke blevet gået yderligere i detaljer med denne version, da jeg har valgt at holde mig til de ovennævnte versioner af L-systemer i denne opgave. En beskrivelse af denne type L-systemer findes i [ABoP] afsnit 1.10.2.

11 Teknikker

Til algoritmen der er udviklet i dette projekt er der brug for en række teknikker. Disse teknikker vil blive gennemgået i dette kapitel og vil i kapitel 12 blive sat sammen til den endelige algoritme.

11.1 Teksturificering

I 7.2 er princippet i teksturificerings processen forklaret. Med dette afsnit vil det blive beskrevet, hvordan i praksis der bliver genereret en tekstur ud fra en model.
Først skal der bestemmes en bounding boks for modellen for at finde ud af i hvilket område modellen udbreder sig. Derpå skal der besluttes hvilket plan som modellen skal projiceres ind på og fra hvilken side af planet projiceringen skal ses fra. I dette eksempel skal et objekt projiceres ind på xy-planet, hvor det punkt, som scenen betragtes fra, er placeret ud af den positive del af z-aksen. Projiceringen er en parallelprojicering.



Figur 3: Teksturificering

Teksturen bliver den størrelse, der angives ved den udfyldte blå firkant.

11.2 Billboard

En metode der ofte benyttes i computerspil er at polygon modellen projiceres ind på billboards, som er fordelt over modellen. Disse billboards bliver benyttet til at repræsentere modellen i stedet for polygonerne. Man kan vælge at benyttet flere eller færre billboards alt efter i hvilken afstand man betragter modellen fra. Ulempen ved denne metode er at der kommer en kraftig reducering af detaljerne i modellen, med mindre man benytter mange teksturer evt. med høj opløsning. Dog har denne algoritme den fordel at den ikke er så ressourcekrævende som de andre.

Når den teksturificerede model skal renderes, gøres dette ved at placere en eller flere teksturer på tilhørende billboards. Antallet af teksturer afhænger af metoden til at repræsentere modellen på. Det vil altid være en parallel projicering, der benyttes til at generere teksturerne for at bevare modellens dimensioner.
Der er 3 typer billboards, der vil blive diskuteret her.

Den første type kaldes skærm orienteret billboard. Dette er et billboard, der orienterer sig så den altid er parallel med skærmen. For at indstille sig roteres

billboardet omkring 2 akser. Denne type billboard er ikke så anvendelig til træer, da den roterer om et center punkt. Den kan bedre anvendes til objekter, der udbreder sig jævnt omkring et centrum.

Et billboard, der minde lidt om det første, er et enkel akse orienteret billboard, som er et billboard der også orienterer sig efter skærmen, men dette billboard kan kun rotere omkring en akse. Denne form for billboard er velegnet til træer da man ville placere dem så de rotere omkring en vertikal akse, der følger stammen.

Den sidste type er et statisk billboard som kaldes verden orienteret billboard. Denne form for billboards orienterer sig efter akserne i verdenen. Der vil ofte blive benyttet flere billboards af denne type som bliver sat oven i hinanden i en fælles akse.

De to bevægelige billboards, som orienterer sig efter skærmen kræver at de bliver opdateret med jævne mellemrum når det punkt som modellen ses fra har bevæget sig et stykke rundt om modellen. Uden en sådan opdatering vil modellen være ens fra alle sider.
Den statiske version er valgt ud fra at den ikke skal renderes igen, når sættet af teksturer er genereret.

Jeg har valgt at benytte 2 billboards, som placeres vinkelret på hinanden. Denne metode har den fordel at når de 2 teksturer er lavet så skal man ikke tænkt på hvilken side modellen betragtes fra. Ulempen ved denne metode er at der kun benyttes 2 teksturer til at vise en model fra 4 sider. Ses modellen fra en vinkel hvor kun den ene tekstur ses og går man til den modsatte side af modellen så vil det være det samme man ser, bare spejlvendt.

En løsning ville være at have 4 teksturer og så skifte mellem dem, mens observatørens synsvinkel er parallel med billboardet. Eller man kunne rendere modellen uden lys og benytte et normal map, der indeholder normaler for hver texel med information om hvilken retning lyset falder ind på den enkelte texels. Dette vil dog stadigt være den samme tekstur man ser fra begge sider men lyset skulle gerne give en lille farve nuance. Normal mappet skal udnyttes når teksturen renderes ved hjælp af et shader program.

11.3 Sektioner

For at kunne benytte teksturificeringsmetoden skal der udvikles en metode til at bestemme, hvor teksturerne skal placeres. Yderligere skal der udvikles en metode til at bestemme hvordan L-systemet skal deles op i sektioner for at LoD kan indføres.
Til at bestemme de forskellige sæt af teksturer indføres der 2 nye basis symboler til L-systemet, ”<” og ”>”. De kan sammenlignes med ”[” og ”]” men de har yderligere den effekt, at det der står mellem ”<” og ”>” tilhører en sektion. En sektion kan have undersektioner ved at angive flere sektions tegn inde i hinanden, f.eks.

< A < B > < C > >

De 2 nye symboler må ikke forveksles med dem de symboler der benyttes i kontekst følsomme L-systemer. Hvilken af de to betydninger symbolerne har vil altid kunne ses ud af den kontekst de står i.

Den yderste sektion der indeholder A, har også undersektionerne med B og C i sig. Denne hierarkiske opdeling kan benyttes når modellen betragtes fra forskellige afstande. Betragtes den fra stor afstand så vil sektionerne blive slået sammen til en så det kun er en tekstur, der benyttes til repræsentation af modellen. Når man kommer tættere på, vil modellen bestå af 3 teksturer, en der indeholder hvad A står for, en med B og en med C.

< Påbegynd ny sektion, *M* bliver arvet fra den omkringliggende sektion.

Påvirker ikke *M*

> Afslut sektion, der vendes tilbage til tilstanden lige før den tilhørende ”<”.

Påvirker ikke *M*

Ved indførelsen af sektioner til L-systems så kommer der også nogle flere betingelser for hvordan ord der bliver genereret kan se ud. Reglerne fra teori afsnittet er ikke længere globale regler for hele ordet, men gælder i en sektion.

Fra teori afsnittet kommer alfabetet, Σ nu til at se ud som:

Σ = (F f + - ∧ & \ / | $ [ ] { G . } ~ ! ’ % < >)

Et ord G ser nu lidt anderledes ud: G = (Σ, ν, P), med aksiomet ν = < ω >, hvor ω består af bogstaver fra Σ^+, og P er et sæt af produktionsregler.

Produktionsreglerne for det eksempel der kan udvikles til alle tilladte ord vil nu se således ud:

ω = T

T → TFT , TfT , T+T , T-T , T∧T , T&T , T\T , T/T , T|T , T$T , T~T , T!T , T’T , T%T , TPT , TQT , Λ

D → T, DT.TD , Λ

P → [T] , {D}, {D[D}T], [T{D]D}

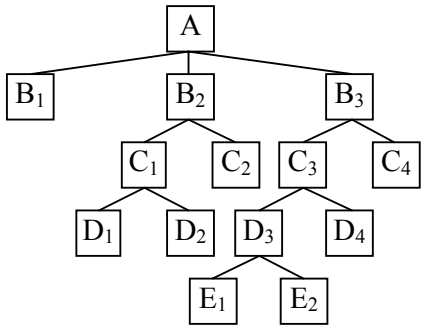
Q → < T >

Der må ikke være nogen af "[" og "{", som ikke afsluttes inden sektionen afsluttes.

Ofte ses der ikke mere end 3 niveauer af detaljer, men for at teste algoritmen vil der blive benyttet 5. Der er ikke valgt et variabelt niveau som indførelsen af sektioner ellers også ville kunne bruges til. Derimod skal der i hvert afstandsinterval tages stilling til hvor dybt i hierarkiet der skal laves nye teksturer, før nogle sektioner bliver slået sammen. Det er oplagt at i det afstandsinterval, der er tættest på træet skal alle sektioner have deres egen tekstur og i det fjerneste interval skal alle sektioner slås sammen til en tekstur. I det følgende vil jeg vise hvordan man med et ord, der har en sektionsdybde på 5, vil slå sektioner sammen ved hjælp af en lineær funktion af afstanden. Ordet der betragtes ses her:

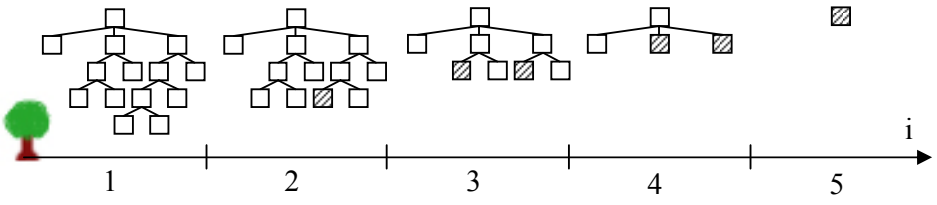
$$<A<B_1><B_2<C_1<D_1><D_2>><C_2>><B_3<C_3><C_4<D_3<E_1><E_2>><D_4>>>>$$

Grafisk vil dette kunne repræsenteres i en træstruktur, hvor hver knude er en sektion:



Figur 4: Sektionshierarki

Den lineære tilgang til dybden vil give en udvikling, som ses på figur 5. De skraverede sektioner repræsenterer sektioner, der er kollapset med underliggende sektioner i hierarkiet.



Figur 5: Sektionshierarkiet som funktion af afstanden

Højden af sektionshierarkiet kaldes *h*. Dybden fra hvilken underliggende sektioner skal kollapses kaldes *d*.

En forskrift for denne dybde, *d*, som funktion af afstandsintervallet vil i eksemplet på figur 5 være, hvor $h = 5$:

$$d = -i + 6$$

Forskriften kan generaliseres for en vilkårlig højde, *h*:

$$(1) \qquad d = \frac{1-h}{4} \cdot i + \frac{5}{4} \cdot h - \frac{1}{4}$$

Denne ligning er fremkommet ved at betragte de to endepunkter $i = 1$ og $i = 5$ og sætte dem ind i forskriften for en ret linie: $y = ax + b$:

$$1) \quad h = a \cdot 1 + b$$

$$2) \quad 1 = a \cdot 5 + b$$

Trækkes de 2 ligninger fra hinanden fås:

$$a = \frac{1-h}{4}$$

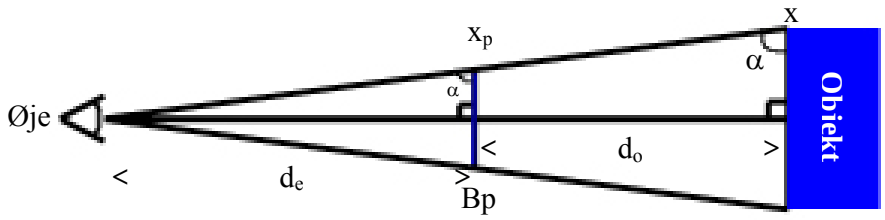
Ved at indsætte det i 2) fås:

$$b = \frac{5}{4} \cdot h - \frac{1}{4}$$

Det skal bemærkes at der er kombinationer af h og i hvor ligningen ikke giver et heltal. I de situationer skal der blot afrundes til nærmeste heltal.

Som alternativ til den lineære tilgang kan man se på hvordan det areal et objekt optager på skærmen aftager som funktion af afstanden til det.

Følgende eksempel viser en perspektivisk projicering:



Figur 6: Antal pixel som et objekt optager på skærmen

Det blå objekt bliver projiceret ind på billedplanet (Bp). Afstanden fra øjet til billedplanet betegnes med d_e og afstanden fra billedplanet til objektet betegnes med d_o. Det punkt der bliver betragtet er x, som er et af hjørnerne i objektet. Projiceringen af x betegnes med x_p.

De 2 retvinklede trekanter (Øje, d_e, x_p) og (Øje, d_e+d_o, x) er ensvinklede, derfor gælder det at:

$$\frac{d_e + d_o}{x} = \frac{d_e}{x_p}$$

Ved en isolering af x_p fås:

$$x_p = \frac{x \cdot d_e}{d_e + d_o}$$

Når $(d_e + d_o)$ fordobles så bliver x_p halv så stor og da det sker for alle punkterne i objektet så vil det arealet som objektet optager på skærmen formindskes med en faktor 4, når afstanden fordobles.

Når der skal vælges en funktion til at bestemme hvor mange niveauer af undersektioner, der skal vises så vælges en funktion der afhænger af x^2.

Der er 2 måder man kan gøre brug af den information at arealet, som et objekt optager af pixels er afhængig af afstanden i anden potens. Den første metode er at man kan lade afstanden af hvert interval fordobles jo længere man kommer væk fra modellen og så benytte den lineære metode til at bestemme, hvornår i sektionshierarkiet sektioner skal slås sammen.

Den anden måde ville være at benytte den faste længde af hvert interval og så benytte en funktion der afhænger af intervallet i anden potens til at bestemme hvornår sektioner skal kollapses.
Hvis der benyttes en ligning med forskriften: $y = ax^2 + b$, så fås ligningerne i yderpunkterne $i = 1$ og $i = 5$:

$$1) \quad h = a \cdot 1^2 + b$$

$$2) \quad 1 = a \cdot 5^2 + b$$

Benyttes samme fremgangsmåde som før, med at subtrahere 1 fra 2 så fås:

$$a = \frac{1 - h}{24}$$

b bliver til:

$$b = \frac{25h - 1}{24}$$

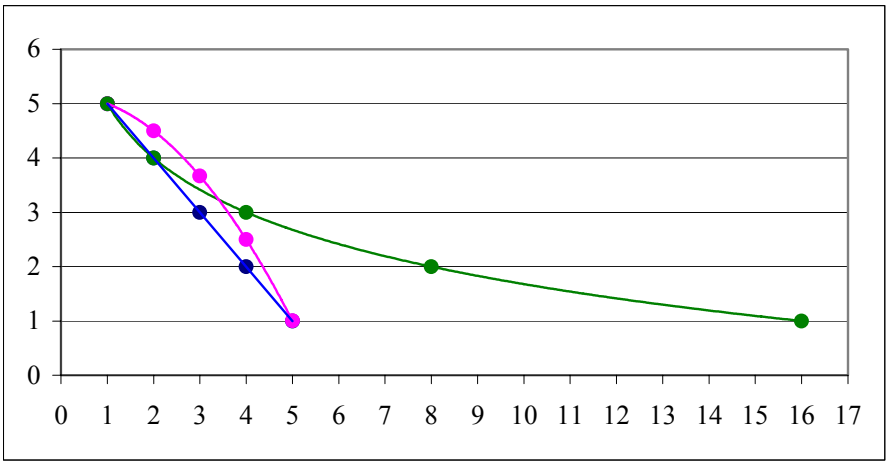
Sættes det ind fås en ligning med forskriften:

$$y = \frac{1 - h}{24} x^2 + \frac{25h - 1}{24}$$

Eksemplet fra tidligere med $h = 5$, får så forskriften:

$$y = -\frac{1}{6} x^2 + \frac{31}{6}$$

De 3 kombinationer til at bestemme den dybde i sektionshierarkiet, der skal slås sektioner sammen fra er afbilledet i følgende graf, hvor dybden er afbilledet ud af y-aksen og afstanden til træet ud af x-aksen:

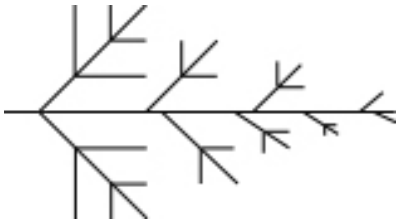


Figur 7: Forskellige metoder til bestemmelse af hierarki dybden

Den blå linie er den første version som er den lineære version.
Den pinkfarvede linie er den metode hvor man benytter den kvadratiske metode til at bestemme dybden, hvorpå der skal slås sektioner sammen. Den har den egenskab at antallet af sektioner der tegnes er højere end den lineære metode, når man er tæt på objektet. Dog lader det til at det er meget lidt den afviger fra den lineære tilgang så derfor vil der ikke blive gået videre med den.
Den grønne linie viser dybde ved den metode hvor intervallernes længde fordobles hele tiden. Dette bliver en logaritmisk bestemmelse af dybden. Denne version ser meget mere interessant ud, da den skiller sig meget ud fra de 2 andre.

I resultat kapitlet vil der blive vendt tilbage til de 2 af ligninger, for at vise hvordan det ser ud i praksis når de benyttes.

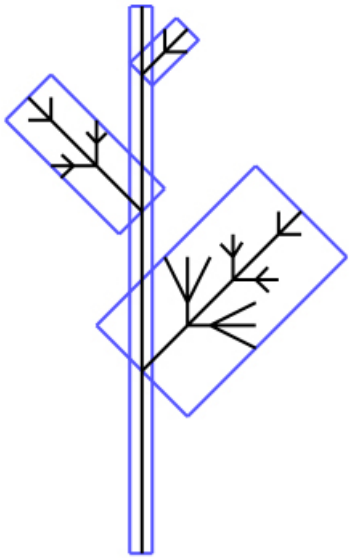
For at benytte den valgte teksturificeringsmetode mangler der en metode til at bestemme orienteringen af teksturerne. For hver sektion laves der 2 teksturer. Da L-systems ofte benyttes til at modellere planter og specielt træer så er det oplagt at se på hvordan en generaliseret gren på et træ ser ud:



Figur 8: En generaliseret gren

En gren har ofte den struktur, at der er en hovedakse, hvor der går sidegrene ud fra. Dette vil blive benyttet til at bestemme, hvor teksturen skal placeres i træet. Hver gang en ny sektion bliver oprettet vil teksturen placeres så dens kanter

ligger henholdsvist parallelt med hovedaksen og vinkelret på denne. De 2 teksturer ligger også indbyrdes vinkelret på hinanden.
Når der bliver oprettet en ny sektion vil alle operationer der udføres før en operation, der tegner noget blive gemt i en separat tilstandsmatrice. Dette gøres for at indstille sektionen efter den første tegne operation. Et eksempel på dette ses her:



Figur 9: Træ opdelt i sektioner

Træet er opdelt i 4 sektioner, hvor de 3 sidegrene er undersektioner til den sektion der indeholder stammen af træet.
Den tilstandsmatrice der bliver gemt, kaldes M_0. Alle punkter i en sektion holdes i et lokalt koordinatsystem, så M_0 benyttes til at transformere punkterne i en sektions koordinatsystem til forældre sektionens koordinatsystem.

11.4 Filter

Når en tekstur skal tegnes bliver det gjort ved at lægge teksturen på en quad. Der bliver benyttet et alfa test, så alle texels med en alfaværdi under 0 ikke bliver tegnet. Derved fjernes de områder af teksturen, som træet ikke dækker. På grund af begrænsninger i tekstur opløsningen vil der opstå alising. Dette vil have den tydeligste effekt på teksturen i overgangen mellem de områder, hvor til der er tegnet og de områder hvor der ikke er. Dette vil give en sort streg omkring træet, som ikke er ønskværdigt, med mindre man ønsker en tegneserie effekt.

Teksturen har 4 kanaler, de 3 første til farven af den enkelte texel, rød, grøn og blå. Den 4. kanal anvendes til alfa værdien, der angiver gennemsigtigheden af den enkelte texel. Metoden som en tekstur bliver lavet på er at først bliver alle texels i den sat til sort med alfa værdi på 0, hvorefter modellen tegnes.

Der er flere måder man kan afhjælpe problemet med alising. En metode vil være at forsøge at gætte en passende farve som alle texels initialiseres til i stedet for sort. Dette vil dog blive ret svært, da planter ofte har mange farver i sig.

En anden metode er at først renderes modellen hvor alle objekter er forstørret en smule. Denne rendering sker uden at sætte alfa værdien i teksturen. Derefter tegnes modellen normalt med alfa værdi. Så opnås der en tekstur hvor der er en bufferzone rundt om modellen hvor alfaværdien er 0 men farven svarer til den værdi som nabo texlen med alfaværdi over 0 har. Denne metode kræver at alle objekter i modellen kan forstørres, man kan ikke bare skalere scenen med en faktor, det skal være en skalering af hvert objekt individuelt.

Den valgte metode til at afhjælpe alising problemet er at benytter et filter i stedet. Først tegnes modellen med alfa værdien, hvorpå filteret derefter finder alle kanter og tværer farven af kanten ud så de nabo texels der ligger langs kanten med alfaværdi på 0 får en farve, som er tæt på modellens farve langs kanten.

Det har vist sig at der ikke er brug for et særligt avanceret filter, det er nok at benytte et simpelt 3x3 udtværingsfilter, som er vist her:

Et udsnit af teksturen på 3x3 kaldes G

```
1   if G(2,2)[4] = 0   //alfa værdien af center texlen
2       then n ← 0
3            res ← (0,0,0,0)
4            for i ← 1 to 3
5                do for j ← 1 to 3
6                    do if i != j && G(i,j)[4] > 0
7                        then res ← res + G(i,j)
8                             n ←n+1
9       res ← res / n
10      res[4] = 0
11      return res
12  else
13      return G(2,2)
```

Kantsituationerne er ikke inkluderet i denne pseudokode, men i disse situationer skal de manglende texels i G antages at have alfa værdi på 0, så de bliver ignoreret.

Figur 12 viser resultatet af at benytte filteret på den tekstur, der er vist på figur 10. Figur 11 er det tilhørende alfamap. Værdierne bliver afrundet til nærmeste heltal:

Tekstur før filter:

0	0	0	0	0
0	0	1	0	0
0	0	3	2	0
0	0	3	2	0
0	0	0	1	0

Figur 10: Tekstur før filter

0	0	0	0	0
0	0	1	0	0
0	0	1	1	0
0	0	1	1	0
0	0	0	1	0

Figur 11: Det tilhørende alfamap

Tekstur efter filter er påført:

0	1	1	1	0
0	2	1	2	2
0	2	3	2	2
0	3	3	2	2
0	3	2	1	2

Figur 12: Tekstur efter filter

Det ses at det område der er tegnet noget til på figur 10, bliver tværet ud over et større areal på figur 12.

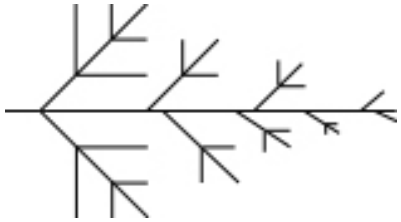
11.5 Bounding bokse

Der er specielt to typer af bounding bokse, der vil blive set på:

Den første version er Axis aligned Bounding Box. ABB har den fordel at den er hurtig at bestemme. Det er et spørgsmål om at bestemme de mindste(x,y,z) samt de største (x,y,z) blandt alle de betragtede punkter.

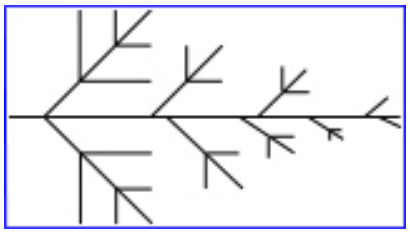
Anden version er en Object aligned Bounding Box. OBB er mere krævende at bestemme, da man skal benytte en teknik til at finde hoved akserne i objektet. F.eks. kan man benytte Principal Component Analysis, som er en metode der analyserer et objekt for at finde dets orientering i rummet. Orienteringen beskrives med 3 vektorer, hvor der derefter bestemmes en bounding boks i koordinatsystemet beskrevet af de 3 vektorer.

OBB vil generelt have et mindre spild areal end ABB, men er mere tidskrævende at beregne. Når man ser på de objekter som der skal bestemmes bounding bokse for, så er det ofte grene der har symmetri omkring grenens hovedakse. F.eks.



Figur 13: Generaliseret gren

Hvis algoritmen derfor kan sørge for at grenens hovedakse bliver den samme som x-aksen i det lokal koordinatsystem, vil man få nogle bounding bokse som vil lægge tæt omkring grenen selv ved at benytte ABB. Jeg har vurderet at det er en minimal forbedring at benytte noget andet en ABB.

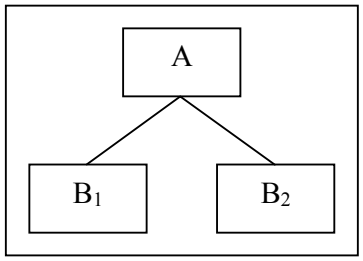


Figur 14: Generaliseret gren med bounding boks

11.5.1 Bounding boks af flere sektioner

En anden faktor, der også skal tages hensyn til, er hvordan bounding boksen bestemmes for foreningsmængden af flere sektioner.

Hvis man tager dette eksempel med 3 sektioner, hvor B_1 og B_2 er under sektioner til A:

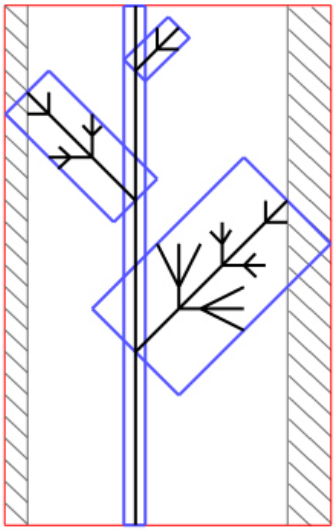


Figur 15: Fælles bounding boks af flere sektioner

Når den fælles bounding boks af flere sektioner skal bestemmes kan det udregnes det ved at transformere alle koordinaterne i B_1 og B_2 til A's koordinatsystem. Dette ville kræve mange matrice multiplikationer men vil give den mindste bounding boks, som ABB algoritmen kan beregne.

En anden metode, som ikke vil give den mindste bounding boks som ABB kan beregne, men derimod ikke kræver lige så mange matrix multiplikationer. Denne metode går ud på at der bestemmes en bounding med ABB i hver af sektionerne A, B_1 og B_2 i deres respektive koordinatsystem. Derefter transformeres B_1 og B_2 bounding boksene til A's koordinatsystem. Den fælles bounding boks bliver bestemt som bounding boksen af de 3 bounding bokse. Det bliver så 8 matricemultiplikationer for hver af sektionerne B_1 og B_2, som er langt færre end at skulle multiplicere alle punkterne i B_1 og B_2.

Da der ikke bliver bestemt bounding bokse af alle punkterne i foreningsmængden af punkter i A, B_1 og B_2 så vil der være situationer hvor der kommer spildarealer. Figur 16 viser et almindeligt problem der vil opstå.



Figur 16: Uudnyttet areal på en tekstur

De blå kasser er bounding bokse for hver af sektionerne. Den røde kasse indikerer den fælles bounding boks og det skraverede areal vil være uudnyttet areal på teksturen, når man bestemmer bounding bokse på denne måde. Dette spildareal kan mindskes ved anvendelse af en anden metode til at bestemme bounding bokse. I takt med at flere sektioner bliver slået sammen, så bliver det uudnyttet areal større.

På trods af at der opstår et spild areal på teksturerne når bounding bokse bliver bestemt på denne måde så vælger jeg at benytte denne. Alternativet kræver at alle punkter skal multipliceres med en eller flere matricer for at få en bedre udnyttelse af teksturerne.

En anden ulempe ved det lokale koordinatsystem er at vektoren der indeholder lysets position og den vektor der benyttes i $-elementet, skal transformeres til dette lokal koordinatsystem. For at udregne lysets positionen i en undersektions koordinatsystem så skal den matrice, der bringer koordinater fra undersektions koordinatsystem til forælderens koordinatsystem, inverteres og ganges med lysets positionen.
For vektoren til $-elementet sker det ved at isolere rotationerne i transformationsmatricen. Denne matrice inverteres og ganges med forælderens $-vektor.

11.6 Yderligere symboler der er tilført til L-systemer

For at kunne sætte materiale konstanter, samt tekstur på modellerne er der blevet tilført et par ekstra symboler til L-systemer.

11.6.1 Tekstursymbol

Det er muligt at sætte teksturer på de modeller der skabes. Dette gøres ved at benytte '#'. Denne tekstur vil bliver brugt på grene, som bliver genereret ved hjælp af F eller G operationer. Af implementeringsgrunde kan der kun benyttes en # pr. sektion. Hvis der angives flere så er det den sidste tekstur i sektionen der benyttes. Denne bliver arvet med ned gennem sektionshierarkiet så derfor kan # også sættes til NULL, for at fjerne den aktuelle tekstur. Teksturen bliver blended ind i den diffuse farve.

#(*tex*)	Sætter en tekstur for hele sektionen, hvor *tex* er en tekstur man selv har genereret. Påvirker ikke *M*

11.6.2 Materialesymbol

Det er muligt at angive et materiale, der skal benyttes til grene og polygoner. Det er muligt at have flere materialer i en sektion. Alle grene og polygoner får tildelt det materiale, der er angivet på det tidspunkt som "F", "G" eller "{" står i ordet. Tegnet, der angiver et materiale, er @.

@(*mat*)	Angiver et absolut materiale, hvor *mat* er et materiale med specular, diffuse, ambient og highlight. Påvirker ikke *M*

' operatoren ændrer på den diffuse farve.

11.6.3 L-system med alle udvidelser

Alfabetet bliver derved udvidet til:

Σ = (F f + - ∧ & \ / | $ [ ] { G . } ~ ! ' % < > # @)

og @ bliver tilføjet til T-produktionen:

ω = T

T → TFT , TfT , T+T , T-T , T∧T , T&T , T\T , T/T , T|T , T$T , T~T , T!T , T'T , T%T , TPT , TQT , T@T , T#T , Λ

D → T, DT.TD , Λ

P → [T] , {D}, {D[D}T], [T{D]D}

Q → < T >

12 Algoritmen

I dette afsnit vil de teknikker, der er beskrevet i det foregående afsnit blive sat sammen til den endelige algoritme. Algoritmen bliver benyttet i hver af sektionerne i sektionshierarkiet.

Overordnet går algoritmen ud på at der først skal bestemmes en bounding boks. Derpå skal de 2 teksturer renderes og til sidst skal de modificeres med det udtværingsfilter der er nævnt i afsnit 11.4.

Afsnit 12.1 er en overordnet gennemgang af hvordan en bounding boks for de enkelte sektioner bestemmes.

I afsnit 12.2 vises algoritmen i pseudokode. Denne algoritmen gennemløber hele sekvensen, der skal til at generere en erstatningsmodel uden at tage højde for, hvor lang tid den har brugt. Derfor kan den ikke i første omgang benyttes til online generering af erstatningsmodellen. I afsnit 12.3 vil pseudokoden bliver delt op i mindre tilstande for at kunne benyttes til online generering.

I algoritmen indgår en række variable. Hver sektion kender sit niveau i sektionshierarkiet ved hjælp af variablen "`my_lvl`". Variablen "`max_depth`" er information om dybden, hvorfra der ikke længere skal oprettes enkeltstående sektioner. Værdien af "`max_depth`" er bestemt ved hjælp af ligning (1) i afsnit 11.3. Afstandsintervallet bestemmes på 2 måder. For begge metoder gælder det at punktet hvor modellen betragtes fra er i afstanden u fra træet. Den første er den lineære version, hvor hvert afstandsinterval har samme længde D. Dette giver en forskrift for i:

$$i = \min(5, \lceil \tfrac{u}{D} \rceil)$$

I ligningen ses min(5,...), det skyldes at det sidste afstandsinterval er 5, højere værdier af i benyttes ikke.

Den logaritmiske version, hvor længden af afstandsintervallerne fordobles jo længere man kommer væk fra træet følger her, hvor D er længden af det interval der er tættest på træet:

$$i = \min\left(5, \max\left(1, \left\lceil \frac{\log(\frac{u}{D})}{\log(2)} + 1 \right\rceil\right)\right)$$

Denne ligning er bestem ud fra følgende observation. Der skal findes en forskrift for i, hvor der i grænsen mellem 2 afstandsintervaller gælder: $2^{i-1} = \frac{u}{D}$, hvilket vil sige, er man i afstanden D, så skal $i = 1$, er man i afstanden $2D$, så skal $i = 2$ osv.
Mellem afstanden 0 og D, der blive log-funktionen negativ, derfor er max(1, ...) med. Mellem de andre afstande der bliver i afrundet opad.

12.1 Bestemmelse af bounding bokse

Bestemmelsen af bounding bokse gøres på den velkendte måde ved at analysere alle punkterne i en mængde for de største og mindste værdier. Pseudokoden til at bestemme disse bounding bokse for de forskellige mængder af punkter findes i appendiks A.

Hver af sektionerne skal kunne bestemme deres bounding boks, samt den fælles bounding boks for sektionen og alle dens undersektioner. Metoden der benyttes står beskrevet i afsnit 11.5.1. Resultatet gemmes i 6 variable:

`x`, `y`, `z`, `xLength`, `yLength`, `zLength`

Hvis sektionen ikke bliver slået sammen med nogle undersektioner så er `x`, `y` og `z` henholdsvis lig med den mindste x, y, z-værdi for alle punkterne i grenene, polygonerne samt hjørnerne af alle bounding bokse for alle objekter i sektionen.

”`xLength`”, ”`yLength`” og ”`zLength`” er henholdsvis den maksimale x, y, z værdi for alle punkterne i grenene, polygonerne samt hjørnerne af alle bounding bokse for alle objekter i sektionen minus den minimale x, y, z værdi.

Hvis sektionen slås sammen med undersektioner så skal hjørnerne i undersektionernes bounding bokse også med i de 6 variable.

De 6 variable repræsenterer det hjørne af bounding boksen med den mindste værdi af alle punkters (x, y, z)-koordinater. Variablene ”`xLength`”, ”`yLength`” og ”`zLength`” er størrelsen af bounding boksen i x, y og z retningen.

12.2 Algoritmen i pseudokode

Renderingen af et sæt af teksturer kan som nævnt opdeles i 3 faser. 1. fase er bestemmelse af bounding boksen, som blev beskrevet i afsnit 12.1. I fase 2 bliver teksturerne renderet, som vi blive forklaret i dette afsnit. Fase 3 er modificeringen af teksturerne og dette er beskrevet i afsnit 11.4.

Renderingsfasen består af to metoder. Der er en metode, der renderer alle grene, polygoner og objekter i sektionen relativt til forældresektionen og en metode til at initialisere sættet af teksturer samt rendere alle grene, polygoner og objekter i sektionen.

12.2.1 DrawRelative

Tegner alle sektionens grene, polygoner og objekter, samt alle undersektioner.

DrawRelative()

```
1   PushMatrix()
2   MultMatrix(M0)
3   DrawBranchMesh()
4   DrawPolygons()
5   DrawSurfaces()
6   for i ← 1 to SubSections.Size
7     do SubSections[i].DrawRelative()
8   PopMatrix()
```

12.2.2 RenderNewTextureSet

Rækkefølgen hvormed de 2 projiceringer, der benyttes til at generere de 2 teksturer, udføres på er, at først projiceres modellen ind på xy-planet og derefter ind på xz-planet. Variablen 't' benyttes til at holde styr på hvilken projicering, der bliver genereret. Linierne 8-10 sætter projiceringen op ud fra den beregnede bounding boks. I linierne 14-16 bliver objektet flyttet ind mellem klipplanerne, der er sat op i linierne 8-10.

RenderNewTextureSet(max_depth)

```
 1  CalculateBoundingBox(max_depth)
 2
 3  ActivateDrawBuffer()
 4
 5  for t ← 0 to 1
 6    do
 7      ViewPort(0, 0, TEXTURE_SIZE, TEXTURE_SIZE)
 8      if t = 0
 9        then Ortho(x, x+xLength, y, y+yLength, z, z+zLength)
10        else Ortho(x, x+xLength, z, z+zLength, y, y+yLength)
11
12      ClearBuffer()
13
14      if t = 0
15        then Scale(1,1,-1)
16        else Rotate(-90, 1, 0, 0)
17
18      EnableLight()
19
20      DrawBranchMesh()
21      DrawPolygons()
22      DrawSurfaces()
23
24      if my_lvl >= max_depth
25        then for i ← 1 to SubSections.Size
26               do SubSections[i].DrawRelative()
27
28      DisableLight()
29
30      ModifyTexture()
31
32      CreateTexture()
33
34  DisableDrawBuffer()
```

12.3 Algoritmens tilstande

Som tidligere nævnt kan pseudokoden i 12.2 ikke benyttes i realtid, da dette kræver algoritmerne blive delt op i tilstande. Algoritmen skal fungere ved at der kaldes en update() funktion med to parametre: ”`eye`” og ”`render_time`”. Parameteren ”`eye`” angiver det punkt, som træet betragtes fra. Såfremt algoritmen ikke er i gang med at generere et detaljeniveau, så vil denne værdi blive benyttet til at bestemme om der skal beregnes er nyt LoD. Parameteren ”`render_time`” angiver, hvor lang tid algoritmen har til rådighed. Hvis algoritmen er i en tilstand, hvor den er i gang med et nyt LoD, så vil algoritmen bruge denne tid, der er afsat til algoritmen og fortsætte næste gang update() bliver kaldt.

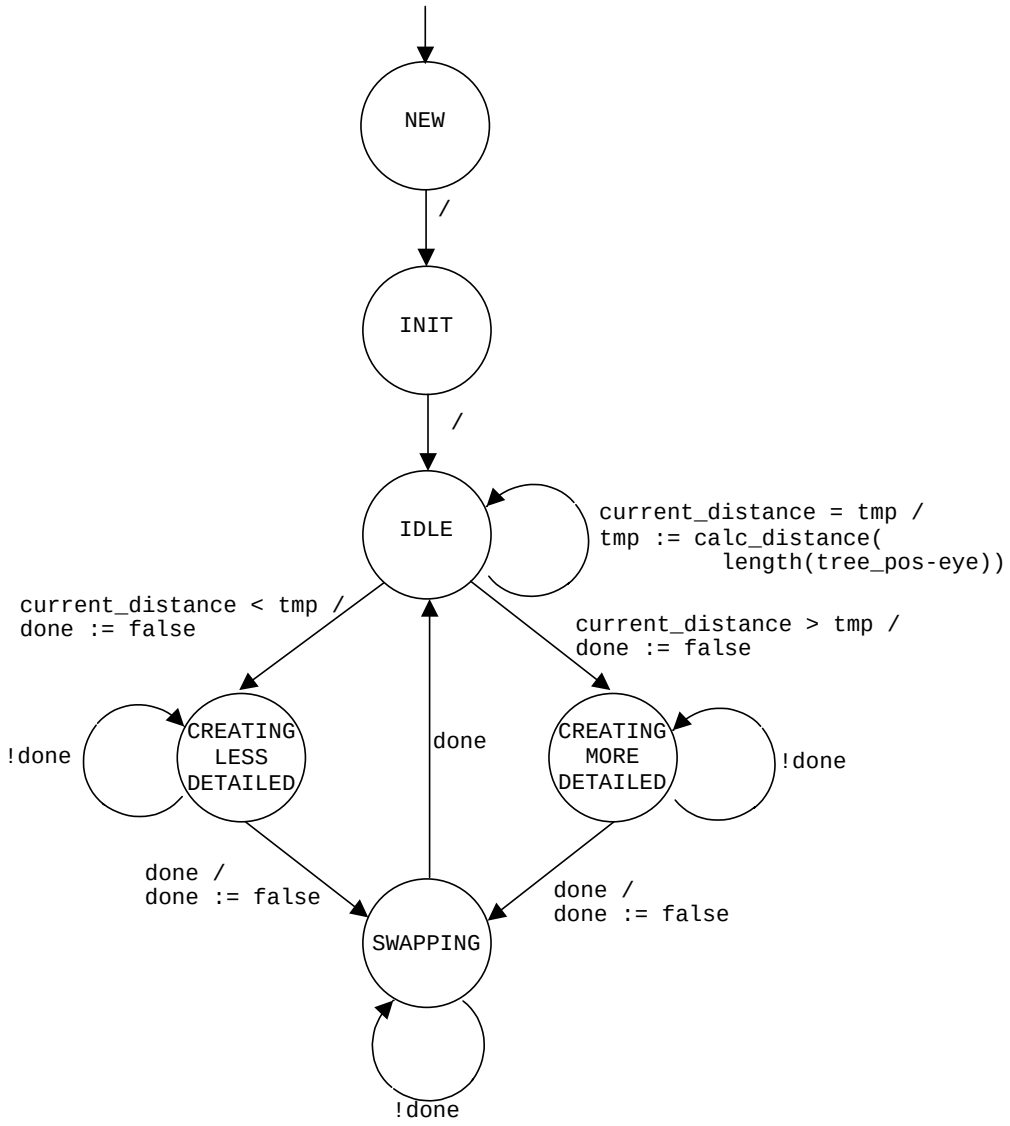
Alle algoritmerne benytter, at de kan kører i et stykke tid, stoppes og så fortsætte på et senere tidspunkt. Derfor er det vigtigt, at algoritmen er i en utvetydig tilstand hele tiden, som der kan fortsættes fra.

Algoritmen er delt i to klasser, Tree, som er hele modellen og LoDSection, som er en enkelt sektion. Tree indeholder update() funktionen.

Den metoden, der benyttes til at bestemme, hvornår algoritmen skal pause er at der i Tree-klassen er et ur, som startes, når update() kaldes. Alle sektionerne har adgang til dette ur og algoritmen skal med passende mellemrum checke om tiden er overskredet.

12.3.1 Træets tilstande

Algoritmen der benyttes til at bestemme hvornår der skal ske et skift mellem 2 sæt af teksturer, kan beskrives med dette tilstandsdiagram:



Figur 17: Træets tilstande

`current_dist:`	Parameter, der indeholder information om hvilket afstandsinterval som øjet er befinder sig i. Afstandsintervallet bestemmes på baggrund af afstanden til træet.
`calc_distance():`	Funktion der benyttes til at bestemme afstandsintervallet ud fra en afstand.
`tree_pos:`	En vektor der indeholder træets position i verdenskoordinater.

`eye:` En vektor der indeholder koordinater for det punkt som træet bliver set fra.

`NEW`
Træet starter i denne tilstand, som benyttes til at sætte nogle parametre der benyttes senere. Bl.a. i hvilket afstands interval træet befinder sig i til at starte med.

`INIT`
Det første sæt af teksturer bliver genereret i `INIT` tilstanden. Grunden til denne tilstand er at der ikke kan tegnes noget på dette tidspunkt.

`IDLE`
I tidspunkterne mellem 2 teksturificeringsprocesser, befinder algoritmen for træet sig i `IDLE` tilstanden. Hver gang der kaldes update() og træet er i denne tilstand vil der blive undersøgt om der skal skiftes til et andet sæt af teksturer.

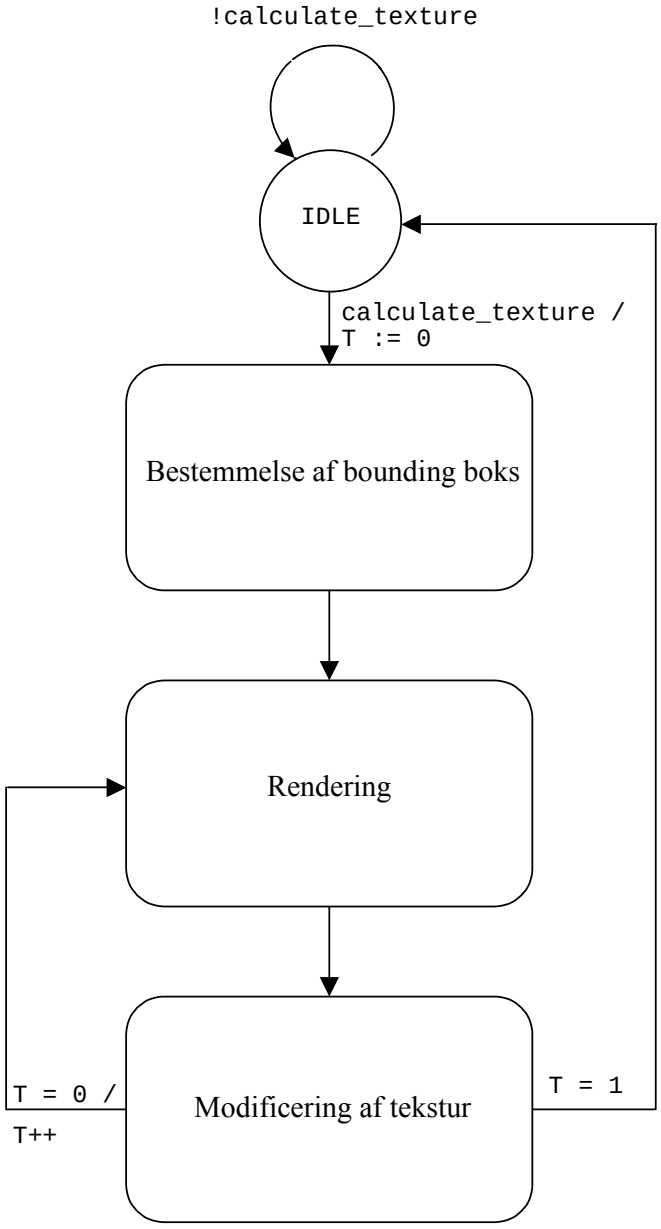
`CREATING LESS DETAILED`
Her kaldes den algoritme, der laver et nyt sæt af teksturer med parameteren: ”nuværende niveau – 1”. Når alle sektioner der skal benyttes til det næste LoD melder tilbage at de er færdige blive `done` sat til `true` og algoritmen fortsætter til `SWAPPING`

`CREATING MORE DETAILED`
Her kaldes den algoritme, der laver et nyt sæt af teksturer med parameteren: ”nuværende niveau + 1”. Når alle sektioner der skal benyttes til det næste LoD melder tilbage at de er færdige blive `done` sat til `true` og algoritmen fortsætter til `SWAPPING`

`SWAPPING`
Når algoritmen der laver et nyt sæt af teksturer er færdig, skal der skiftes i alle sektionerne til de nye teksturer.

12.3.2 En sektions tilstandsdiagram

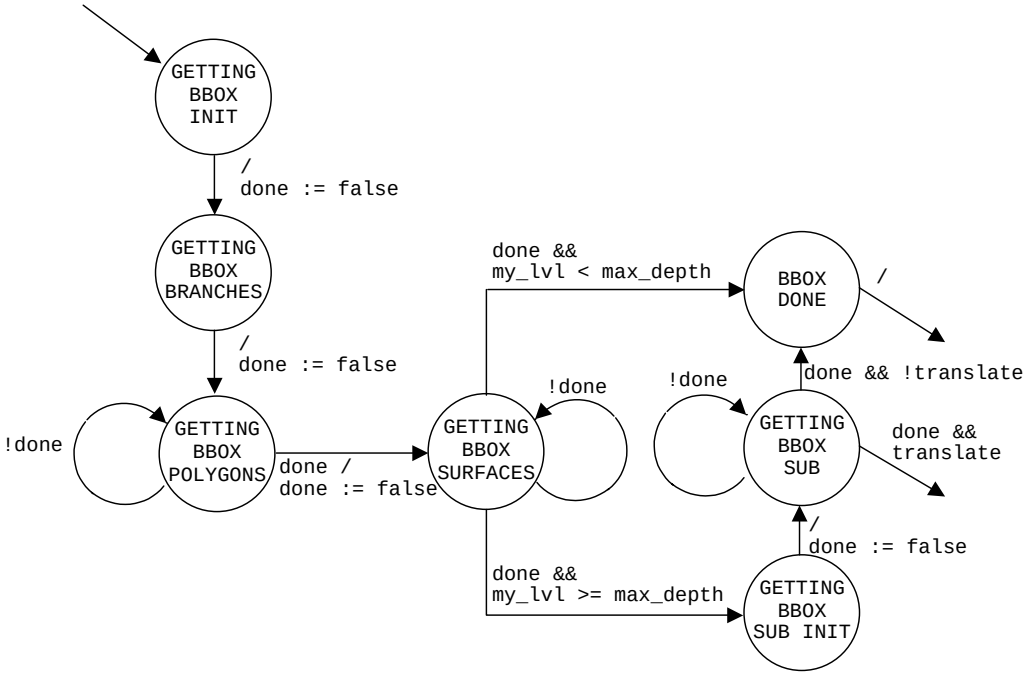
Her ses en oversigt over de tilstande, som en sektion gennemgår når den modtager et signal til at genere et nyt sæt teksturer. Når algoritmen ikke er i gang med at generere et nyt sæt teksturer, så befinder den sig i `IDLE`.



Figur 18: En sektions tilstande

Indholdet af de rektangulære tilstande følger herefter. Det skal nævnes at for sektioner, der er blevet slået sammen med andre sektioner, bliver de to tilstande ”Bestemmelse af Bounding Boks” og ”Rendering” benyttet separat. Det vil sige at man springer direkte ind i den tilsand og afslutter den ved at gå tilbage til `IDLE` tilstanden.

12.3.3 Bestemmelse af Bounding Boks



Figur 19: Tilstandsdiagram for bestemmelse af en sektions bounding boks

`GETTING BBOX INIT`
Initialiserer variable, som resultatet gemmes i.

`GETTING BBOX BRANCHES`
Bounding boksen af grenene bliver beregnet løbende med at der tilføjes grene til modellen. Derfor har denne tilstand ikke en loop, hvor den checker uret om algoritmen har overskredet tidsgrænsen.

`GETTING BBOX POLYGONS`
Hvert polygon udregner deres bounding boks samtidig med at der bliver tilføjet punkter til polygonet. Derfor skal algoritmen på dette tidspunkt samle alle bounding bokse sammen fra hvert af polygonerne. Det bliver løbende testet om den afsatte tid er overskredet, variablen `done` bliver først `true`, når alle polygoner er blevet analyseret.

`GETTING BBOX SURFACES`
På samme måde som med polygonerne bliver bounding boksen opdateret med alle objekters bounding boks. Når den er færdig besluttes der om der skal indsamles bounding boks for underliggende sektioner eller om sektionen er selvstændig. Til dette benyttes variablen ”`my_lvl`” og parameteren ”`max_depth`”. Hvis ”`my_lvl`” er mindre end ”`max_depth`” betyder det at det niveau den nuværende sektion er på, ikke skal slås sammen med underliggende sektioner og bestemmelsen af bounding boksen er derfor færdig.

`GETTING BBOX SUB INIT`
Intialisering af nogle parametre der benyttes til at bestemme bounding boks af de underliggende sektioner.

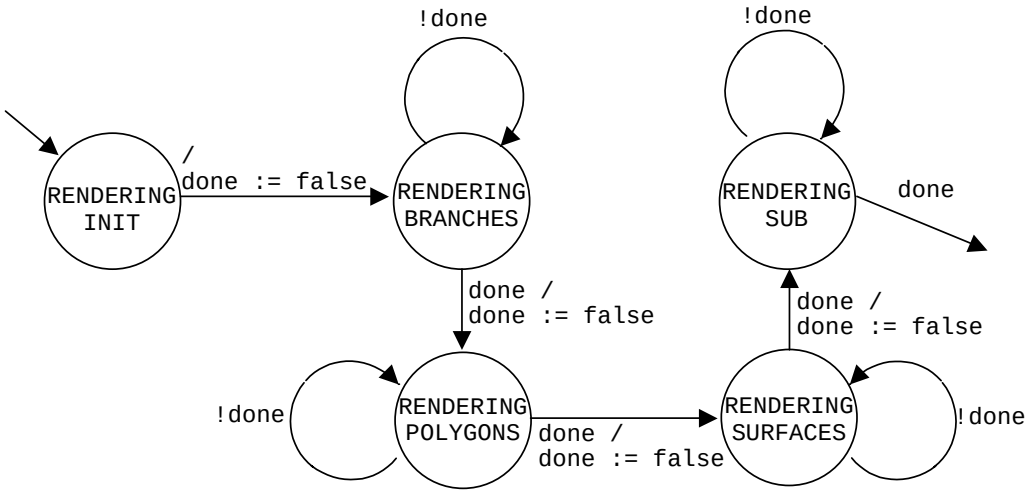
`GETTING BBOX SUB`
Her kaldes getBoundingBox på hver af de underliggende sektioner. Alle de underliggende sektioner løber tilstandsdiagrammet igennem for at bestemme deres bounding boks og returnere resultatet hvor efter den fælles bounding boks bestemmes. Der benyttes et flag "`translate`" som indikerer om der skal konverteres til forældre sektionens parameter. Hvis det er tilfældet så ved algoritmen også at der ikke skal gås videre til renderingsfacen, da sektionen er slået sammen med forældren.

`BBOX DONE`
Kopierer de værdier der er fundet for bounding boksen over i de variable der er omtalt i afsnit 12.1:

`x`, `y`, `z`, `xLength`, `yLength`, `zLength`

12.3.4 Rendering

Denne del af algoritmen kan også kaldes på to tidspunkter. Enten er det i forbindelse med at en ny sektion skal have lavet nye teksturer eller også er det fordi at sektionen er blevet slået sammen med en forældre sektion og skal derfor tegne sit indhold på forælderens tekstur.



Figur 20: Tilstandsdiagram for rendering af en sektion

`RENDERING INIT`
I det tilfælde at den er blevet kaldet fordi der skal laves et nyt sæt af teksturer så sætter denne tilstand viewport og projicering op.
Hvis den kommer i til denne tilstand, fordi en forældre beder sektionen om at tegne indholdet af sektionen på forælderens tekstur, så sker der ikke noget i denne tilstand.

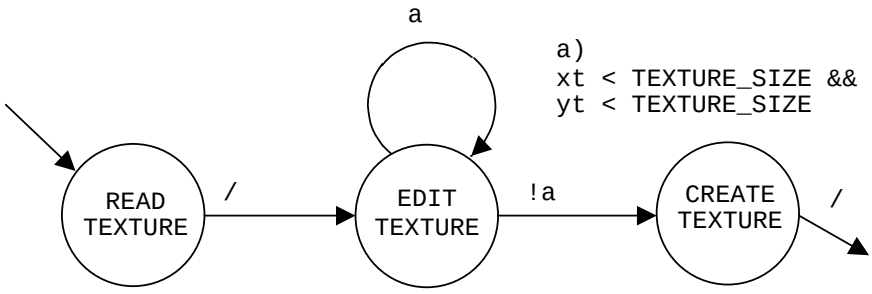
`RENDERING BRANCHES`
Alle grene renderes mens der løbende bliver holdt øje med om renderingstiden er overskredet.

`RENDERING POLYGONS`
Det samme princip som med grenen

`RENDERING SURFACES`
Samme princip som de 2 foregående tilstande.

`RENDERING SUB`
Hvis ”`my_lvl`” for sektionen er lig med ”`max_depth`” så bliver alle undersektionerne renderet.

12.3.5 Modificering af tekstur



Figur 21: Tilstandsdiagram for modificering af en tekstur

`READ TEXTURE`
I denne tilstand bliver data fra p-bufferen læst til processor hukommelsen.

`EDIT TEKSTURE`
Da filteret ikke kan nå at behandle hele teksturen på en gang benyttes der 2 variable: ”`xt`” og ”`yt`”, som er to heltal, der gennemløber de 2 dimensioner som teksturen har. Med passende mellemrum bliver der checket om den afsatte renderingstid er overskredet.

`CREATE TEKSTURE`
Den modificerede data bliver læst til tekstur hukommelsen.

13 Modellering af planter

En observationer, der er gjort under arbejdet med denne rapport er at ved modelleringen af planter skal man være opmærksom på, at der ikke bliver genereret for mange sektioner, idet det vil blive for processorkrævende at generere en erstatning for planten. Hvis man ikke er omhyggelig med < og > elementerne så vil antallet af sektioner stige eksponentielt.

Man kan godt generere modeller, der har overdrevent mange fraktallignende områder uden at det vil vise sig på erstatningsmodellen, da den kraftige reduktion af detaljer, der sker når en erstatning genereres for den rigtige model. Der skal dog også tages højde for dette når L-systemet modelleres for hvis man er interesseret i at det kan ses så bliver man sandsynligvis nødt til at bruge mange sektioner.

Det følgende træ som er baseret på Hondas træer i kapitel 2 i [ABoP], har mange fraktallignende områder.

13.1 Hondas Træ

I forhold til Hondas træer i kapitel 2 i [ABoP] er der blevet indført et materiale til stammen og grenene på træet, samt der er også sat blade og blomster på.

Den måde som opdelingen i sektioner er implementeret i dette L-system er ved at udnytte den måde som træet udvikles på. En gren vokser lige ud med mindre og mindre længdeforøgelse for hvert trin i udviklingen. Der sker så det at hver gang der forøges med et stykke så vokser der en side gren ud.

Sektions opdelingen sker på følgende måde. Hver gang der vokser en ny sidegren ud så er det en ny sektion, dog kun så længe grenen ikke er udviklet i mere end 5 trin, derefter vil det tilhører samme sektion. Metoden der holder styr på hvor langt en gren er nået i udviklingen er at et A-element transformeres til et B, som transformeres til et C osv. indtil der nås et E og derefter et H. E og H laver ikke nye sektioner med tilføjer kun elementer til den samme sektion.

Der er også blevet indført tilfældigheder for hvordan der udvikles sidegren i L-systemet. Dette er gjort for at kunne generere træer med varierende udseende uden at der skal skabes et nyt L-system for hvert træ. Tilfældighederne benyttes til at tage beslutning om der skal startes på en ny sidegren og derved om der skal startes en ny sektion.

Når modeller genereres på denne måde vil nogle af de teksturer, der bliver genereret, komme til at ligge parallelt med hinanden og overlappe hinanden. Dette medfører, at når den teksturificerede model vises, vil dybde checket ikke kunne beslutte, hvilken af teksturerne, der ligger tættest på det punkt de betragtes fra. Dette medfører at de områder, hvor teksturerne overlapper vil der hele tiden bliver skiftet mellem de aktuelle teksturer og det vil ikke se godt ud. For at undgå dette bliver alle sektioner drejet meget lidt så der ikke kommer noget problem i dybde checket. Konstanten "a_f" angiver denne vinkel som der drejes med og ved at prøve mig lidt frem så fandt jeg frem til at den skulle have værdi: 1 grad, for at give det bedste resultat. Herefter følger reglerne:

n = 10

#define r_1	0.900	
#define r_2	0.700	
#define a_0	20.000	
#define a_2	-30.000	
#define d	137.500	
#define w_r	0.707	
#define a_f	1.000	/* vinkel som sektionerne drejes med */
#include S	blad objekt	
#include T	blomst objekt	
#include M	materiale af stammen	
#include N	tekstur til stammen	

ω : @(M)#(N)'(0.8,0.5,0.5)A(1,0.5)

A(l, w) : $\xrightarrow{5\%}$!(w)F(l)/(d)A(l*r_1,w*w_r)

A(l, w) : $\xrightarrow{95\%}$!(w)F(l)<(a_0)/(af)B(l*r_2,w*w_r)>/(d)A(l*r_1,w*w_r)

B(l, w) : $\xrightarrow{10\%}$!(w)F(l)
[$+($a_2$)~T/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S]
C(l*r_1,w*w_r)

B(l, w) : $\xrightarrow{90\%}$!(w)F(l)
[$+($a_2$)~T/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S]
<-(a2)/(af)$C(l*$r_2$,w*$w_r$)>
C(l*r_1,w*w_r)

C(l, w) : $\xrightarrow{10\%}$!(w)F(l)
[$-($a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S]
D(l*r_1,w*w_r)

C(l, w) : $\xrightarrow{90\%}$!(w)F(l)
[$-($a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S]
<+(a$_2$)/(af)$D(l*$r_2$,w*$w_r$)>
D(l*r_1,w*w_r)

D(l, w) : $\xrightarrow{15\%}$!(w)F(l)
[$+($a_2$)~T/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S]
E(l*r_1,w*w_r)

D(l, w) : $\xrightarrow{85\%}$!(w)F(l)
[$+($a_2$)~T/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S]
<-(a$_2$)/(af)$E(l*$r_2$,w*$w_r$)>
E(l*r_1,w*w_r)

E(l, w) : $\xrightarrow{*}$!(w)F(l)
[$-($a_2$)~T-(2*$a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S-(2*$a_2$)~S]
[+(a_2)$H(l*$r_2$,w*$w_r$)]
H(l*r_1,w*w_r)

H(l, w) : $\xrightarrow{*}$!(w)F(l)
[$+($a_2$)~T/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S/(2*$a_2$)~S]
[-(a_2)$E(l*$r_2$,w*$w_r$)]
E(l*r_1,w*w_r)

Resultatet af dette L-system kan ses i resultat kapitlet.

13.2 Palme

Palmen er opbygge på en anden måde end træet fra det tidligere afsnit. L-systemet for den består af 4 regler. A som generere stammen af palmen. H som er en generator, der laver nye sektioner af palmeblade. L er et palmeblad, der slutter af med M som er spidsen af bladet.
Palmen har kun en sektionsdybde på 3, så 2 af erstatningsmodellerne bliver brugt 2 gange.

n = 35

#define a_0	15.00
#define a_1	30.00
#define a_2	35.00
#define a_3	1.00
#define a_4	137.50
#define r_0	1.10
#define r_1	1.20
#define r_2	0.95
#define s_0	0.90
#define e_0	0.05
#define l_0	0.50
#include B	blad objekt
#include C	materiale af en gren
#include D	materiale af stammen

ω : @(D)A(15,20,1)

A(h, b, s): h > 0 → !(s*r_2*s_0)F(0)!(s*r_2)F(1)A(h-1,b,s*r_2)

A(h, b, s): h = 0 → <@(C)H(b,s*r_2)>

H(b, s) : b < 1 → /(a_4)<[L(e_0,a_3)M]>

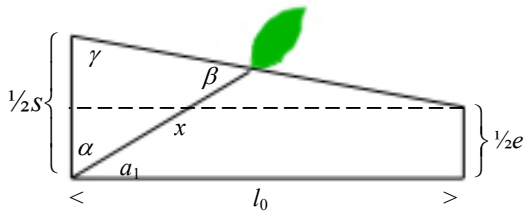
H(b, s) : b > 1 &
b < 2 → /(a_4)<[L(e_0,a_3)M] /(a_4)[L(e_0,a_3)M]>/(a_4)

H(b, s) : b > 2 → /(a_4)<[L(e_0,a_3)M] /(a_4)[L(e_0,a_3)M]>/(a_4)H(b-2,s)

L(s, a) : * →
L(s*r_0, a*r_1)&(a*r_1)
[+(a_2)f(0.5*s/(sin(tan^{-1}(2*l_0/(s*(1-r_0))+
90-a_2)*sin(tan^{-1}(2*l_0/(s*(1-r_0))+90-a_2))))~B]
[-(a_2)f(0.5*s/(sin(tan^{-1}(2*l_0/(s*(1-r_0))+
90-a_2)*sin(tan^{-1}(2*l_0/(s*(1-r_0))+90-a_2))))~B]
!(s*r_0)F(l_0)

M : * →
[+(a_1)f(0.1/sin(tan^{-1}(2*l_0/(0.1-e_0))+90-a_1))~B]
[-(a_1) f(0.1/sin(tan^{-1}(2*l_0/(0.1-e_0))+90-a_1))~B]
!(e_0)F(l_0)!(0)F(0)[+(a_0)~B-(2*a_0)~B]~B

I reglen for L indgår der et f-element. Grunden til at dette optræder er at bladene på en gren skal starte fra kanten af grenen og ikke midt i. Ligningen der står i f-elementet kommer fra den følgende figur, hvor der ses den ene halvdel af en gren:



Figur 22: Afstand som blad flyttes for at komme ud på kanten af grenen

Selve grenen vokser i l_0 retningen og bladet bliver placeret i en vinkel på a_1, så bladet skal placeres i afstanden x fra centrum af grenen.

$$\alpha = 90 - a_1$$
$$\beta = 180 - (\gamma + \alpha)$$

Fra trekanten med den stiplede linie som grundlinie fås:

$$\tan(\gamma) = \frac{2 \cdot l_0}{s - e}$$

I trekanten med vinklerne α, β og γ fås vha. sinus relationen:

$$\frac{\frac{1}{2}s}{\sin\beta} = \frac{x}{\sin\gamma}$$
$$x = \tfrac{1}{2}s \cdot \frac{\sin\gamma}{\sin\beta}$$

Her udnyttes: $\sin(180 - (\gamma + \alpha)) = \sin(\gamma + a)$

$$x = \tfrac{1}{2}s \cdot \frac{\sin(\tan^{-1}(\frac{2}{s-e}))}{\sin(\tan^{-1}(\frac{2}{s-e}) + 90 - a_1)}$$

13.3 Busk

Busken i kapitel 1 side 26 i [ABoP] er taget med som reference og for at vise hvad der kan ske hvis sektioner placeres forkert. Fejlplaceringen af sektions tegne vil der blive set på i kapitel 15.10.

Der er lavet nogle ændringer i L-systemet i forhold til det der står beskrevet i [ABoP]. For det første er der tilføjet sektioner og der er sørget for at niveauet af sektioner ikke overskrider 5, for at holde den eksponentielt voksende antal sektioner under kontrol, når modellen udvikles.
Omløbsretningen, som punkterne i bladene bliver genereret ved er opdateret så de bliver genereret med den omløbsretning, der svare til implementationen.

n = 7
#define δ 22.5
#include M materiale af buskens grene

ω : @(M)!(1)A_1

A_1	:	*	→	<&(δ)FL!A_2>/(5*δ)'<&(δ)FL!A_2>/(7*δ)'<&(δ)FL!A_2>
A_2	:	*	→	<&(δ)FL!A_3>/(5*δ)'<&(δ)FL!A_3>/(7*δ)'<&(δ)FL!A_3>
A_3	:	*	→	<&(δ)FL!A_4>/(5*δ)'<&(δ)FL!A_4>/(7*δ)'<&(δ)FL!A_4>
A_4	:	*	→	<&(δ)FL!A_5>/(5*δ)'<&(δ)FL!A_5>/(7*δ)'<&(δ)FL!A_5>
A_5	:	*	→	[&(δ)FL!A_5]/(5*δ)'[&(δ)FL!A_5]/(7*δ)'[&(δ)FL!A_5]
F	:	*	→	S/(5*δ)F
S	:	*	→	FL
L	:	*	→	[''' ∧∧{+f-f-f+|+f-f-f}]

13.4 Capsella

I kapitel 3 side 74 i [ABoP] er der et L-system for en plante ved navn Capsella. For at prøve systemet af på andre modeller end træer så er denne lille blomst også taget med.
Der er foretaget nogle mindre ændringer i det L-system. Sektioner er tilføjet og alle G-elementer er udskiftet med f-elementet, da det giver et bedre resultat. Yderligere bliver blade lavet med et sammenhængende polygon i stedet for 2. Der er indført en /(a_f) for at rotere blomsterne så de passer bedst ind i en sektion, så planerne blomsten projiceres ind på ligger i en position der ser bedst ud. Værdien af a_f er fundet ved at prøve sig frem.

n = 16
#define r_1 0.900
#define r_2 0.600
#define a_0 70.000
#define a_1 9.000
#define a_2 18.000
#define a_3 50.000
#define i_0 10.000
#define i_1 5.000
#define i_2 8.000
#define i_3 7.000
#define i_4 2.000
#define d_0 137.500
#define d_1 90.000
#define w_r 0.707
#define a_f 25.000
#include M materiale af blomstens stilk
#include N materiale af visne blomster
#include P materiale af blomsterne

ω : @(M)!(1)I(9)a(5)

a(t) : t > 0 → [&(a_0)L]/(d_0)I(i_0)a(t-1)
a(t) : t = 0 → [&(a_0)L]/(d_0)I(i_0)A
A : * → <&(a_2)u(4)FFI(i_0)I(i_1)<X(i_1)KKK>>/(d_0)I(i_2)A
I(t) : t > 0 → FI(t-1)
I(t) : t = 0 → F
u(t) : t > 0 → &(a_1)u(t-1)
u(t) : t = 0 → &(a_1)
L : * → <{.-(a_2)FI(i_3)+(a_2)FI(i_3)+(a_2)FI(i_3)-(a_2)|-
(a_2)FI(i_3)+(a_2)FI(i_3)+(a_2)FI(i_3)}>
K : * → [@(P)&(a_2){/(a_f).+(a_2)FI(i_4)-(a_2)-(a_2)FI(i_4)}]
[@(P)&(a_2){/(a_f).-(a_2)FI(i4)+(a_2)+(a_2)FI(i_4)}]/(d_1)
X(t) : t > 0 → X(t-1)
X(t) : t = 0 → ∧(a_3)[@(N)[+(a_2)ffff-(a_2)-(a_2)[fff[-(a_2)-(a_2)f{.].].].-(a_2)
-(a_2)ffff.+(a_2)+(a_2)fff.+(a_2)+(a_2)f.}]%

14 OpenGL implementeringen

Jeg har valgt at implementere den stokastisk version af L-systemer, da den type også inkludere parametriske L-systemer.

14.1 Tilstandsmatricen

H-, L- og U-aksen er blevet oversat til henholdsvistx-, y- og z-aksen

14.2 Generering af teksturer

De fleste af modellerne benytter teksturer med en dimension på 128x128. Denne størrelse er fremkommet ved at prøve sig lidt frem. Det er en balance mellem hvor detaljeret man ønsker at modellen skal være og hvor meget hukommelse og renderingstid en model må benytte.

Den fase, hvor data fra p-bufferen skal kopieres over til en tekstur kan implementeres på 3 måder:

Den 1 metode gå ud på at man renderer til en p-buffer, læser teksturen via CPU'en til processorens hukommelse ved hjælp af glReadPixels(). Derefter oprettes der en tekstur med glTexImage2D(). Dette er den langsom metode af de tre da man læser data fra GPU'en til CPU'en og tilbage igen til GPU'en. Påførelsen af filtret kan ske når data er læst fra grafikkort og inden data læses tilbage. Fordelen ved denne metode er at implementeringen ikke bliver nær så hardware specifik som de 2 følgende metoder som benytter fragment programmer.

2. metode går ud på at man igen renderer til en p-buffer, men opretter teksturen ved at læse data direkte fra p-bufferen til teksturen ved hjælp af glCopyTexImage2D. Denne metode burde køre hurtigere, specielt ved store datamængder, da den ikke kræver at der skal flyttes data fra grafikkortet og tilbage igen. Da data beholdes på grafikkortet er man nødt til at benytte et fragmentprogram for at modificere teksturen. Det kan gøres ved at data læses til teksturen hvorpå fragmentprogrammet aktiveres og denne tekstur renderes på en quad. Derpå skal data så kopieres til teksturen en gang til.

Sidste metode er den hurtigste, men derimod også den mest hukommelses krævende. Igen renderes der til en p-buffer men data forbliver i denne buffer, som så benyttes som en tekstur. Modificering af data skal også ske her i et fragmentprogram, på samme måde som i 2. metode, men man behøver ikke den sidste kopiering. Grunden til at denne metode er mere hukommelseskrævende er at i de 2 første metoder gemmer man teksturen som en rigtig tekstur med 4 kanaler, rød, grøn, blå og gennemsigtighed. En p-buffer inkluderer også en dybdebuffer, som kun er nødvendig under generering af erstatningen for den rigtige model.

Hvert punkt i de polygoner som bliver genereret med L-systemet skal have udregnet en normal for at lyset kan bestemmes. Metoden som dette gøres på er ved at udregne krydsproduktet af de 2 vektorer, der går fra det betragtede punkt til de 2 punkter, der er tilføjet lige før og lige efter det betragtede punkt. Hvis alle 3 punkter ligger på en linie, så bliver der i stedet forsøgt med det punkt der

er tilføjet 2 trin efter det betragtede punkt. Er det punkt ligeledes på linie med de første punkter, så bliver der forsøg med et punkt, der kommer efter det sidst forsøgte punkt. Hvis alle punkter bliver forsøgt så kan hvilken som helst normal vælges.

14.3 Testmaskinerne

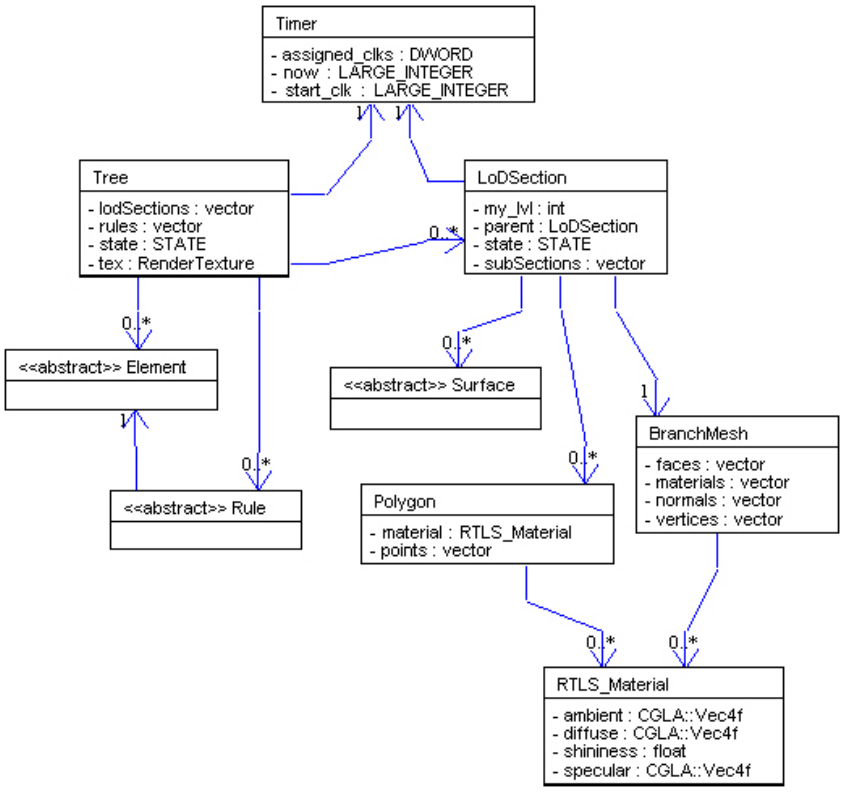
2 testmaskiner er benyttet. De fleste test er udført på maskine nr. 1.

Nr. 1:
Intel Pentium IV processor @ 3,2 GHz med HT teknologi
1 GB ram. Grafikkort: ATI Radeon 8500 med 64 MB ram

Nr. 2:
Intel Pentium IV processor @ 3,4 GHz
1 GB ram. Grafikkort: WinFast PX6600 GT med 128 MB ram

14.4 Klassediagram

En oversigt over de implementerede klasser ses på følgende figur, en mere detaljeret beskrivelse over klasserne findes i appendiks C. Kildekoden kan findes på den vedlagte CD og i appendiks B.



Figur 23: Klassediagram

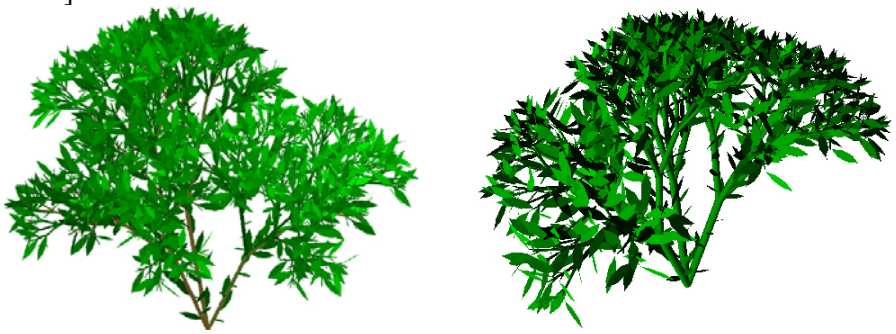
De 2 abstrakte klasser Element og Rule benyttes henholdsvis til at lave elementer i alfabetet og produktionsregler. Tree har nogle funktioner, hvor man tilføjer start elementer og produktionsregler.
Den abstrakte Surface klasse benyttes til at tilføje specielle objekt typer til systemet, som kan bruges i forbindelse med ”~” elementet. I implementeringen er der en X3DObject klasse som implementerer en måde, hvor X3DObject’er kan benyttes.

14.5 Test

En decideret unit test af systemet er ikke mulig at udføre da flere af elementerne kræver grafisk verifikation. Derimod er der lavet sammenlignende test af modellerne i [ABoP] og de modeller der fremkommer ved af tage L-systemerne i [ABoP] og udvikle dem i applikationen. Den første model der er sammenlignet med er Hondas træ fra side 56 i [ABoP]:

Figur 24: Sammenligning med Hondas træ fra [ABoP] til venstre og Hondas træ i OpenGL implementering til højre

Det er til at se en del ligheder, hvis man prøver at tælle efter så passer antallet af sidegrene. Længden af grene og bredden afviger men det er et tilpasningsspørgsmål.
En anden model som der også er blevet sammenlignet med er busken på side 26 i [ABoP]:



Figur 25: Sammenligning med busken fra [ABoP] til venstre og busken i OpenGL implementeringen til højre

Der ses samme struktur i de 2 modeller.

Den visuelle verificering har vist at implementeringen følger beskrivelsen i [ABoP] af elementerne i L-systemer.

14.6 Test programmer

Der er lavet to test programmer. Det først er et program til at vise en model i, kaldet Viewer. I dette program er fps. låst til 60 Hz idet at den ekstra tid skal bruges til at opdatere modellen. Programmet kræver at den maskine, som det eksekveres på har slået lodret synkronisering (V-sync) fra på grafikkortet, da dette vil forstyrre programmet. Den tid, der tildeles til algoritmen bestemmes ved at der er et ur, der tager tid for hvor længe renderingen af erstatningsmodellen tager og den overskydende tid tildeles algoritmen. Programmet tager tid for hvor længe algoritmen er i de forskellige tilstande under en simulering af at man nærmer sig eller bevæger sig væk fra modellen.

Det andet test program er et terræn hvor alle modellerne er indsat i for at undersøge om det er muligt at have flere modeller i en applikation på en gang. Denne applikation har også en øvre grænse på 60 fps. og kræver ligeledes at lodret synkronisering er slået fra på den maskine som det eksekveres på.

Programmerne ligger på den vedlagte CD og kan eksekveres der fra. I appendiks E findes der en brugervejledning til de 2 programmer.

14.7 Software benyttet i implementeringen

14.7.1 OpenGL

API til at generere 2D og 3D applikationer.
http://www.opengl.org

14.7.2 CGLA

Computer Graphics Linear Algebra, et bibliotek udviklet af Andreas Bærentzen ved DTU.

14.7.3 RenderTexture

Et bibliotek der udnytter p-buffer til off-screen rendering. Der er udviklet af Mark J. Harris.
http://www.markmark.net/misc/rendertexture.html

14.7.4 DevIL

Bibliotek til håndtering af billeder/teksturer.
http://openil.sourceforge.net/

14.7.5 X3DObject og BMesh

Mindre hjælpeprogrammer udviklet af Bent Dalgaard som er udleveret i forbindelse med øvelser i kurserne 02561 og 02563.

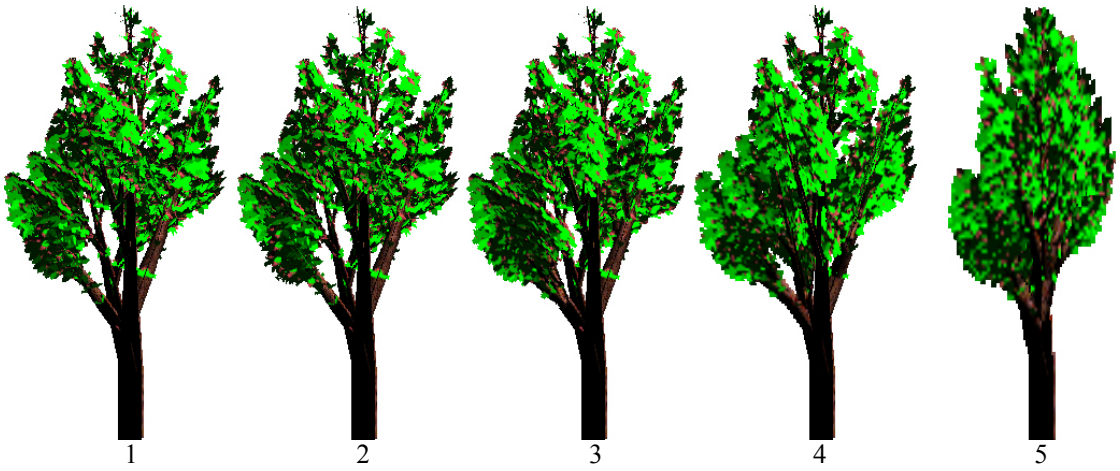
14.7.6 TerrainRoamer

Udviklet i 02563 - Virtual Reality kurset i samarbejde med Henning Degn, s991002 og Søren Hansen, s991331

15 Resultater

15.1 LoD modellerne

i figur 26 kan de 5 niveauer af detaljer ses. De er alle vist i samme størrelse for at man kan se forskellen på de 5 niveauer af detaljer.



Figur 26: De 5 detaljeniveauer for Honda træet

Forskellen på niveau 1 og 2 er ikke så stor, den er ret svær at se med denne model. Det er muligt at se en reduktion i detaljerne på niveau 3, men de er ikke så tydelige.
I niveau 4 er alle sidegrene til stammen en sektion, hvilket også kan ses. Niveauet er meget grynet, idet næsten alle detaljer er forsvundet. Når dette niveau af detaljer benyttes skal modellen være langt væk fra punktet som modellen betragtes fra.

Den teksturificerede model med flest detaljer består af 64 sektioner, hvilket vil sige at der benyttes 128 quads til at repræsentere modellen.
Antallet af quads der benyttes til de forskellige niveauer af detaljer kan ses her:

LoD	1	2	3	4	5
Quads	128	114	70	22	2

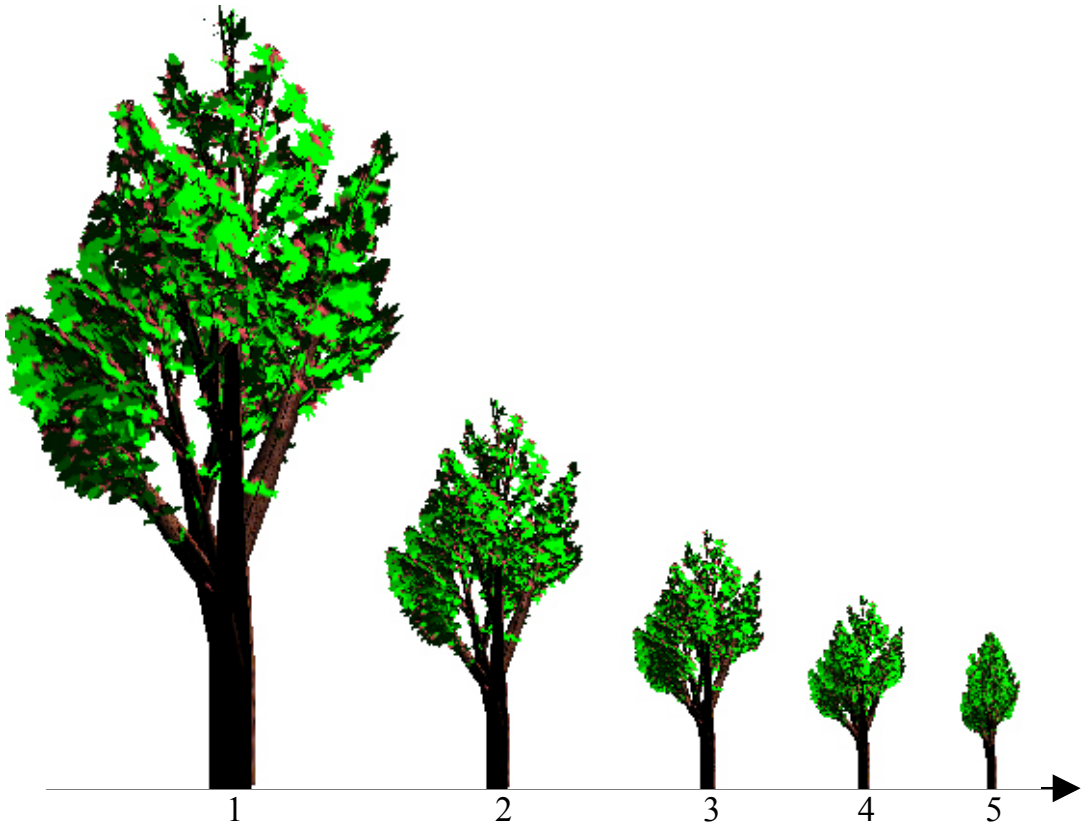
Tabel 1: Antal quads der benyttes til at repræsentere de forskellige detalje niveauer

Det ses, at antallet af sektioner falder mest fra detaljeniveau 3 til 4 og næsten lige så meget fra 2 til 3. Det lille fald fra 1 til 2 forklarer, at der er så lille forskel på de to modeller. De ekstra 44 sektioner, som benyttes til niveau 2 i forhold til niveau 3 giver ikke den helt store effekt. Derfor kunne det tyde på, at det ikke er nødvendigt med 5 niveauer af detaljer, men derimod vil 3 være et bedre antal. Det vil også betyde at modellerne heller ikke skal inddeles i alt for mange sektioner. Niveau 3 vil sandsynligvist kunne benyttes som det mest detaljerede niveau.

15.2 Udseende som funktion af afstand

De to metoder til at bestemme hvilken LoD, der skal vælges som funktion af afstanden til planten, er blevet testet ved at tage en model og benytte begge metoder. Her ses resultatet af de to metoder:

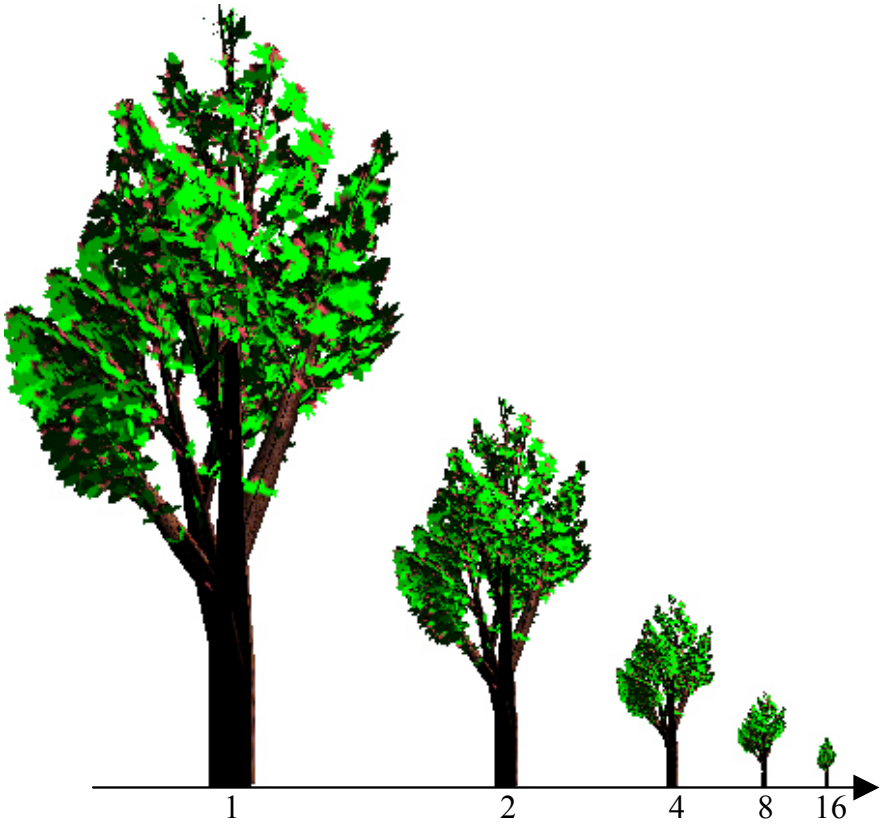
15.2.1 Lineær bestemmelse af LoD



Figur 27: Lineær bestemmelse af detaljeniveauet

Figur 27 viser hvordan modellen ser ud som funktion af afstanden til punktet som modellen betragtes fra. Værdierne under hvert træ er afstanden. Modellen til venstre er den med højest niveau af detaljer og for hvert træ fra venstre falder niveauet af detaljer med 1 niveau.
Fordelen ved denne metode er, at det er kun når man er tæt på modellen at detaljeringsgraden er høj, dvs. at modellen kræver mere processor tid for at blive renderet. Men derimod falder niveauet af detaljer også meget hurtigt. Det falder så hurtigt at den model med den laveste detaljeringsgrad bliver valgt for hurtigt. Det er ret tydeligt at modellen længst til højre har en lav detaljeringsgrad..

15.2.2 Logaritmisk bestemmelse af LoD

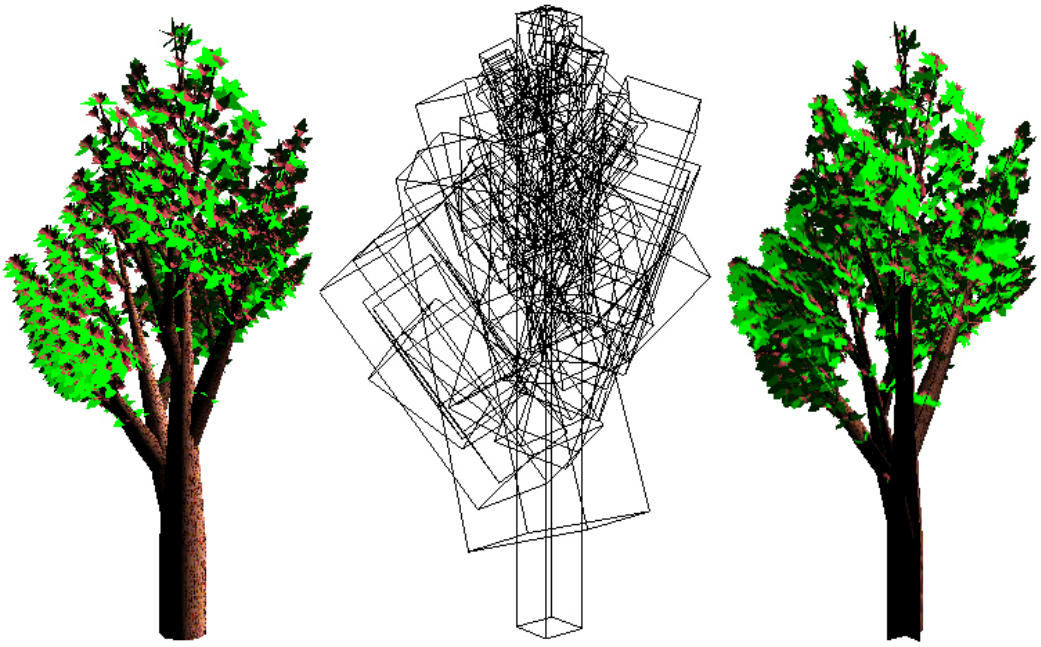


Figur 28: Logaritmisk bestemmelse af detaljeniveauet

Her vises afstanden til træet fra det punkt som modellen betragtes fra under stregen. I afstanden 1 vælges den mest detaljerede model og i afstand 2 den mindre detaljerede osv. I forhold til den lineære metode ser denne ud til at benytte et bedre tidspunkt for hvornår der skal skiftes til den mindst detaljerede model. I afstanden 4 benytter denne metode en mere detaljeret version i forhold til den der vælges med den lineære metode. Dette giver en bedre grafisk effekt, men det betyder også at i en scene med mange træer vil der generelt blive benyttet modeller med flere detaljer hele tiden. Derved vil det blive mere processor krævende at rendere alle planterne med denne logaritmiske tilgang i forhold til den lineære.

15.3 Sammenligning med polygon model

På figur 26 vises til venstre polygon modellen, i midten vises alle sektionerne som bounding bokse og til højre vises det endelige resultat af den teksturificerede model.

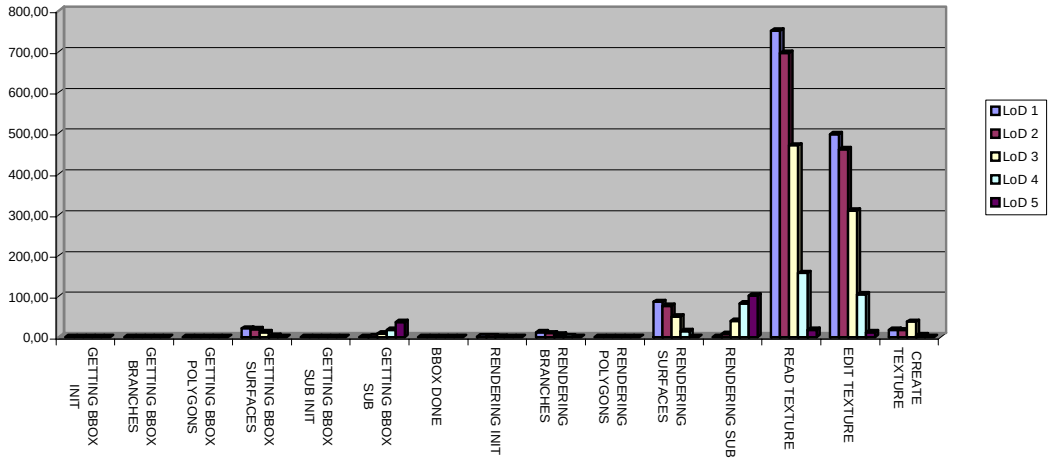


Figur 29: Sammenligning med polygon modellen

Modellen til venstre består af 632.898 polygoner og test maskine 1 kan vise den med ca. 8 fps. Mens den teksturificerede model, som tidligere vist benytter 128 quads og kan derfor vises uden problemer med mindst 60 Hz.
Det interessante ved denne model er at i polygon modellen er der mange matematiske mønstre der gør at man ikke forbinder den med et rigtigt træ og at alle de mønstre forsvinder i den teksturificerede model.

15.4 Tidtagning

Der er blevet målt hvor lang tid algoritmen anvender i de forskellige tilstande for Hondas træ og for alle 5 niveauer af detaljer ser der således ud:



Figur 30: Tiderne for de enkelte tilstande

Det viser sig at det er modificering af teksturen, der er den proces, der tager længst tid. Følgende tabel viser hvor lang tid det tager at beregne en erstatningsmodel for hvert af detaljeniveauerne angivet i millisekunder. Alle tidsmålinger findes i appendiks D.

LoD	1	2	3	4	5
Tid	1387,92	1291,59	937,61	388,81	169,32

Tabel 2: Beregningstiderne for de enkelte LoD for Hondas træ

Det tager ca. 1.4 sekunder at generer den meste detaljerede model på testmaskine 1. Faldet i genereringstid skyldes at der kommer færre og færre af de langsomme tekstur modificeringsprocesser. De 1.4 sekunder er lang tid når der skal tages højde for at det er tiden der skal bruges til at generere en model. Samtidigt med at den ene model genereres bliver scenen også renderet, derfor bliver den reelle tid, for genereringen af erstatningsmodellen, længere end 1.4 sekunder. Hvis der også er flere træer inden for en radius så vil det også blive problematisk at nå at generere dem alle.

Indledende forsøg med et fragment program til modificering af teksturen, hvor der blev benyttet version 2 af de metoder der er omtalt i implementerings afsnittet, gav følgende tider:

LoD	1	2	3	4	5
Hardware	4913	4407	2813	1091	406
Software	5551	4964	3152	1184	400

Tabel 3: Sammenligning mellem et fragment program og software implementeringen

Det er en forsvindende lille tidsbesparelses, der opnås ved at benytte et fragment program i stedet for at føre data over CPU'en, modificere det og

derefter fører data til tekstur hukommelsen. Tiderne nærmer sig hinanden, da antallet af sektioner mindskes.
Tidsmålingerne findes i appendiks D. En nærmere analyse af målingerne af hvilke tilstande, som benyttede mest tid viste, at det er kaldet til glCopyTexImage2D, som kopiere data til tekstur hukommelsen, der tager for lang tid. Da intet tyder på at der vindes tid på at benytte et fragment program blev denne version frafaldet i implementeringen.
Da version 3 også benytter glCopyTexImage2D, dog kun en gang, vil det ikke være interessant at implementere version 3.

15.5 Tilfældigheder

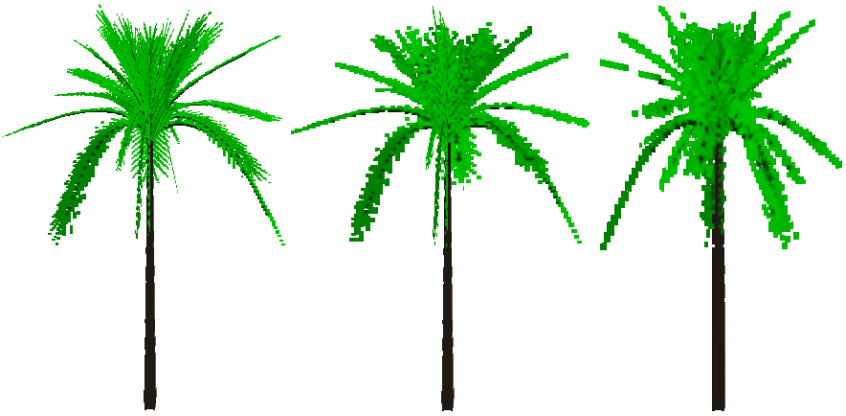
Med tilfældigheder indbygget i L-systemet kommer 3 træer, som er modelleret ud fra det samme L-system, som står beskrevet i 13.1 til at se ud som vist på figur 31:

Figur 31: Træer med indbygget tilfældigheder

L-systemet der ligger bag disse modeller var opbygget så det valgte tilfældigt om en sidegren skulle vokse ud eller ej. I det midterste træ ses tydeligt at den nederst sidegren fra stammen mangler i forhold til de 2 modeller ved siden af. Det er bestemt muligt at skabe L-systemer, hvor der kommer forskellige resultater ud af. Selvom der benyttes samme regler for at generere træerne bliver det unikke træer, der kommer ud af det. Det er L-systemer ret velegnet til at generere.

15.6 Palmen

Palmen indeholder kun 3 niveauer af detaljer. Hvis alle de 3 niveauer forstørres så de har samme størrelse, figur 32, kan man tydeligt se en reduktion af detaljerne over de 3 niveauer i forhold til dem mest detaljerede model til venstre:



Figur 32: Palmens 3 Niveauer af detaljer

Hvis de vises i det størrelses forhold som den logaritmiske metode til at bestemme antallet af detaljer giver, ser det ud som på figur 33:

Figur 33: Palmen vist som funktion af afstanden

Igen ses det at den logaritmiske metode er velegnet til at bestemme hvilket niveau af detaljer der skal benyttes.

Tiden det tager at udregne de 3 niveauer af detaljer er, angivet i millisekunder :

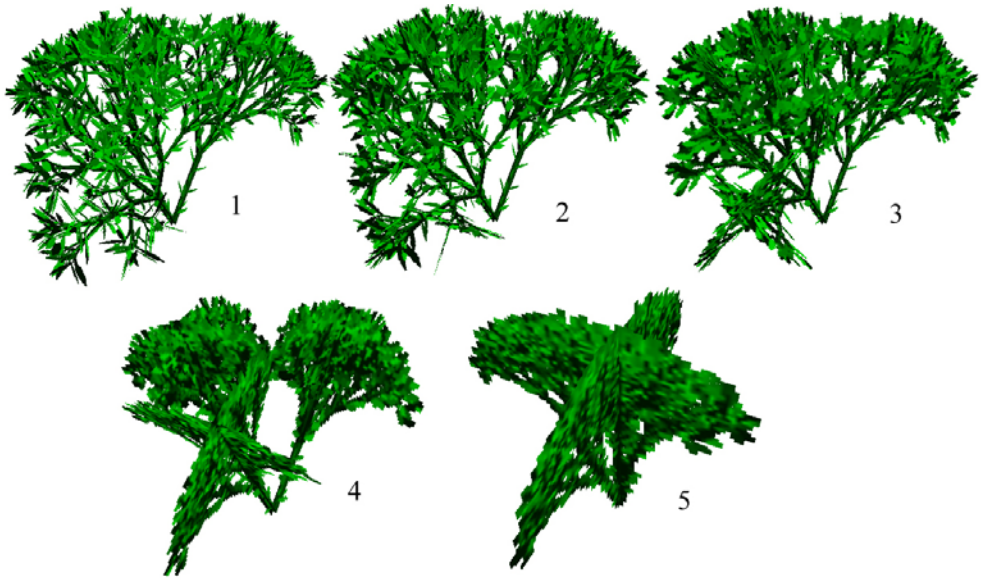
LoD	1	2	3
Tid	390	94	64

Tabel 4: Tidtagning for palmen

Det ses at dette er en forholdsvis hurtige model at modellere, det tager kun 390 millisekunder for den mest detaljerede model at blive genereret.

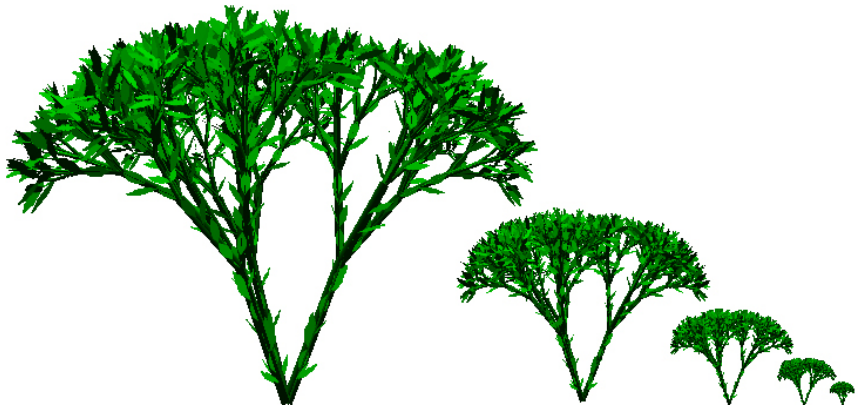
15.7 Busken

Busken i figur 34, har 5 niveauer af detaljer og 121 sektioner, hvilket er den model med de fleste antal sektioner:



Figur 34: Buskens 5 niveauer af detaljer

Det er til at se en reduktion af sektioner når der skiftes mellem modellerne.



Figur 35: Busken vist som funktion af afstanden

Figur 12 ligner måske ikke helt en busk, men metoden, som bestemmer detalje niveauet ser ud til at virke. Det er ikke til at se at det er forskellige modeller, der benyttes i de 5 afstande.

Tiderne det tager at udregne de forskellige niveauer er her angivet i millisekunder:

LoD	1	2	3	4	5
Tid	3650	1415	518	224	131

Tabel 5: Tidtagning for busken

Som det ses af tabellen så er busken en ret tidskrævende model at lave erstatnings modellen for.

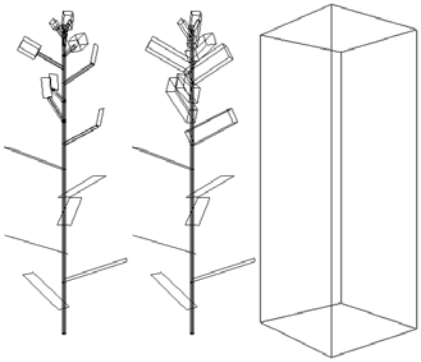
15.8 Capsella

Figur 36 viser en lille blomst, der er blevet delt op i sektioner. Blomsten er ret primitiv men er udmærket til at vise, hvordan sektionerne kommer til at blive slået sammen som funktion af afstanden. Blomsten har 3 niveauer og som det første vises de 3 niveauer i samme størrelse, så det tydeligt fremgår hvordan de 3 niveauer indeholder færre og færre antal detaljer:

Figur 36: Capsella'ens 3 niveauer af detaljer

Hele planten består af 3 dele. Den nederste del er det område hvor der vokser blade ud fra stilken. Derefter følger der et område med visne blomster og i toppen er der blomster der er sprunget ud.

For at vise hvordan sektionerne er placeret, vises i figur 37 planten med bounding boksene:



Figur 37: Capsellas sektioner

Til venstre er der flest sektioner. I midten ses at de visne blomster ikke mere benytter 2 sektioner men er slået sammen til en. Til højre er alle sektioner slået sammen til en stor sektion. Ved at se på sektionen til højre indses det, hvorfor det laveste niveau af detaljer er så grynet. Bounding boksen bliver meget stor i forhold til de bounding bokse der benyttes i de mere detaljerede versioner af planten. Når der benyttes samme størrelse af teksturer til alle sektioner, så kommer hver texel til at repræsentere meget mere af planten og derfor bliver planterne mere grynet.

Grunden til at der benyttes 2 sektioner for at modellere den visne blomst er at den peger lige opad og sidder på en gren der går ud fra hovedstilken. Hvis hele grenen og den visne blomst var en sektion så ville den visne blomst blive projiceret ned, så den peger i samme retning som grenen. Dette er hvad der sker

på modellen i midten, med på det tidspunkt skulle modellen gerne være så langt væk at det ikke er muligt at se.

Tiderne der tager at generere de 3 niveauer af detaljer, angivet i millisekunder:

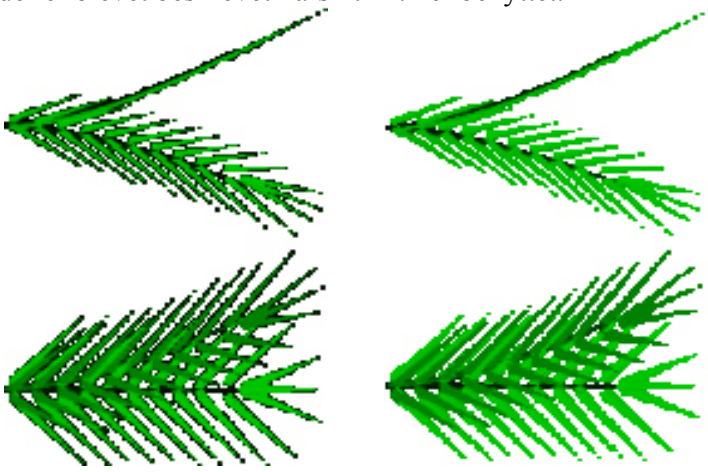
LoD	1	2	3
Tid	834	528	42

Tabel 6: Tidstagning for Capsella

Det er forholdsvis lang tid for så lille en plante, så den plante vil næppe kunne bruges i en applikation.

15.9 Filter

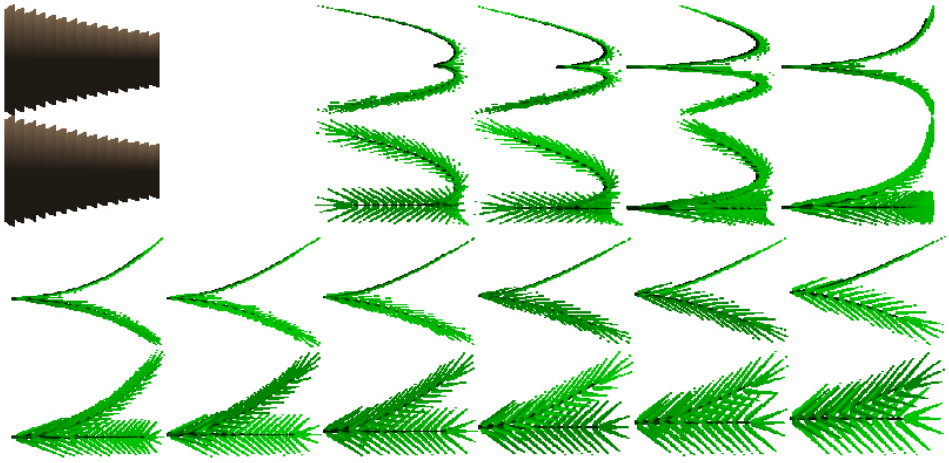
Resultatet af de teksturer, der benyttes til en sektion ses på figur 38 før og efter at filteret, der er blevet beskrevet i afsnit 11.4 er benyttet.



Figur 38: Filters effekt. Til venstre ses modellen før filteret er benyttet og til højre efter

Til venstre ses de 2 teksturer før filteret er anvendt og til højre efter det er anvendt. Der ses tydeligt af den sorte streg rundt om bladene er forsvundet. Så det simple udtværingsfilter er effektivt nok.

For palmen ser alle sektionerne således ud efter filteret er blevet anvendt. Teksturerne bliver ikke gemt i en stor tekstur, som det måske kunne se ud som på figur 39. De bliver alle gemt i enkelte teksturer. Den tomme sektion er centrum, hvor alle grenene vokser ud fra. Når en mindre detaljeret model skal benyttes slås alle grene sammen i den sektion.



Figur 39: Alle teksturerne for det mest detaljeret niveau af palmen

Filteret får fjernet alle de sorte streger og det ses også på den sammensatte model som er vist i kapitel 15.6

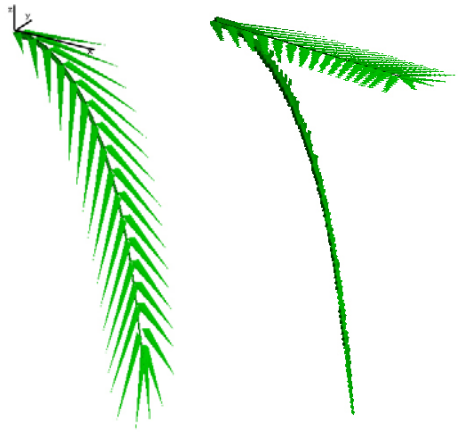
15.10 Svagheder i algoritmen

Når man reducerer antallet af informationer så meget som der gøres ved at teksturificere modellerne vil der opstå grafiske fejl. I dette afsnit bliver der brugt to modeller fra tidligere til at vise disse svagheder:

15.10.1 Palmen

Placeringen af teksturerne i en sektion, har deres styrke når sidegren og blade spreder sig jævnt ud fra center hovedgrenen i sektionen. Hovedgrene skal helst vokse i en nogenlunde ret linie. Den må gerne lave nogle drejninger men hvis den buer så meget som et palmeblad gør, da det bliver trukket ned af tyngdekraften, opstår der et problem.

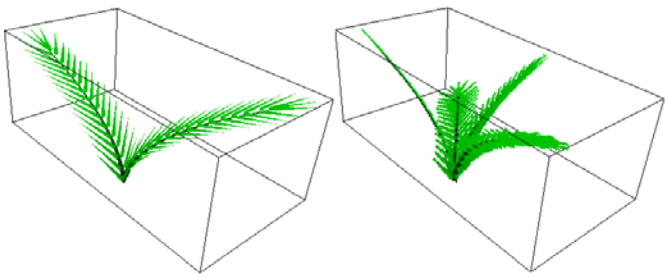
Figur 40 viser et palmeblad. Da palmebladet starter med at vokse vandret og derefter bliver bøjet, vil der ikke være udvikling omkring x-aksen, når man betragter det i xz-planet. Det giver følgende resultat: polygon modellen til venstre og den teksturificered model til højre:



Figur 40: Et palmeblad

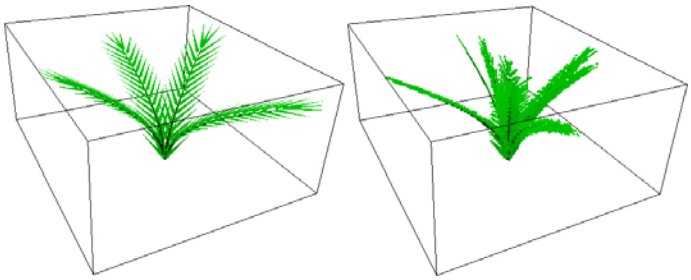
Dette ser ikke ret godt ud, idet grenen nærmest bliver delt i 2 separate dele. For at modvirke denne opdeling kan man dele grenen op i flere dele så der bliver benyttet flere teksturer til at repræsentere palmebladet, men så er det lige før man lige så godt kan benytte polygon modellen for der skal mange sektioner til for at få bladet til at se ud som om det bøjer.

Det er muligt at sløre denne opdeling ved at have flere palmeblade i en sektion. Se figur 41, som er et eksempel med 2 blade i en sektion. Der er stadigt en opdeling men knap så markant.



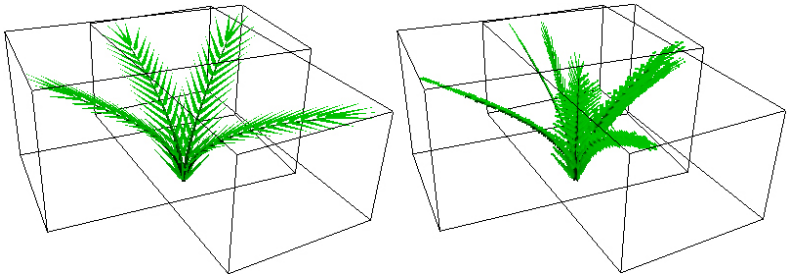
Figur 41: 2 palmeblade i en sektion

Bliver man ved med at sætte flere blade i en sektion sker der det, at detaljer mistes da bounding boksen bliver større og da teksturen har fast størrelse så kommer en texel til at dække et større område, dette kan ses på figur 42:.



Figur 42: 4 palmeblade i en sektion

Man er nødt til at prøve sig lidt frem for at få det bedste resultat. Hvis man prøver at dele de 4 blad op i 2 sektioner i stedet så ser det ud som på figur 43:



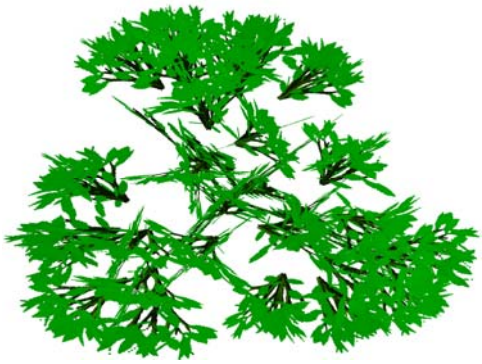
Figur 43: 2 sektioner, med hver 2 palmeblade

Dette giver et bedre resultat end på figur 42. Derfor er der valgt at benytte 2 palmeblade pr. sektion, som det også kan ses i L-systemet for palmen i afsnit 13.2. Det endelige resultat kan ses i figur 44:

Figur 44: Palmen, som polygon model til venstre og teksturificerede til højre

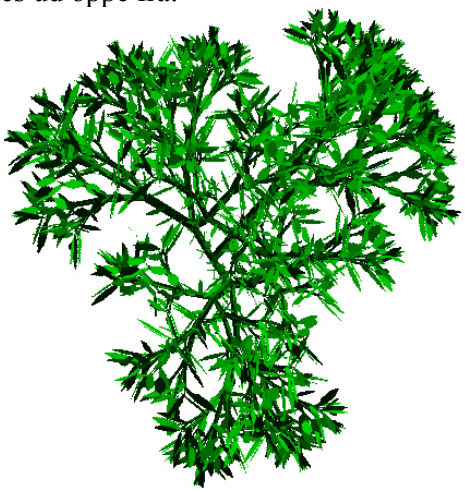
15.10.2 Busken

Hvis der i reglerne for busken i afsnit 13.3, havde været placeret ”<” ”>” rundt om A_1, A_2 osv. i stedet for der, hvor de er placeret, jvf. 13.3, så ville der ske det at sektionerne, ville skabe huller i planten, resultatet af dette kan ses i figur 45, hvor man ser busken oppe fra:



Figur 45: Dårligt valgt af placering af sektionstegn

Billedet er fra tidligt i forløbet, hvor der ikke var kommet lyssætning med i systemet.
Grunden til at der kommer denne opdeling er, at de 3 grene, som går ud fra hver knudepunkt i denne plante så vil komme til at ligge på en fælles tekstur, hvor projiceringen gør modellen usammenhængende.
Hvis der derimod benyttes reglerne som er beskrevet i afsnit 13.3 så kommer busken til at se således ud oppe fra:

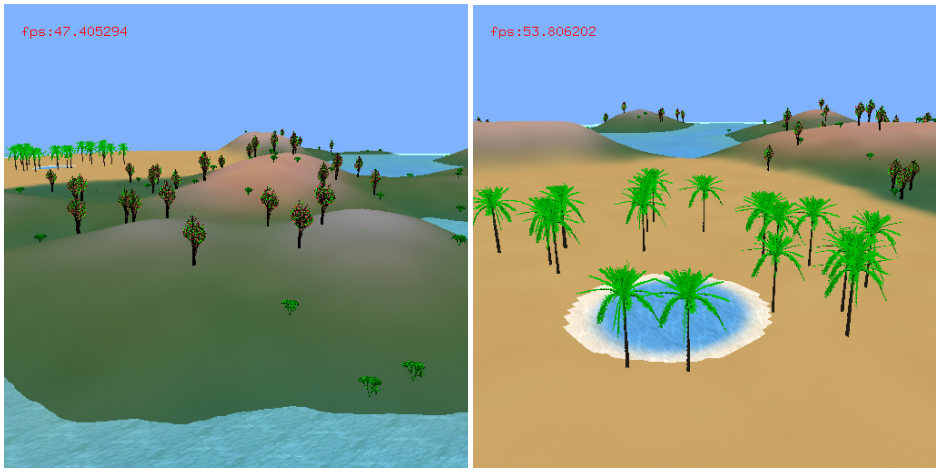


Figur 46: En bedre placering af sektionstegn

Denne model viser, hvordan man skal passe på med, placeringen af sektions elementerne.

15.11 Billeder fra en scene med flere typer træer

De 4 modeller er blevet sat ind i et terræn i figur 47:



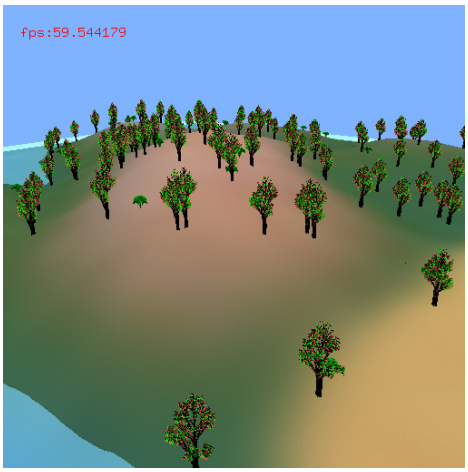
Figur 47: 2 billeder fra terræn med de 4 modeller

Der er sat 50 træer, 20 palmer, 10 buske og 10 capsella'er ind i scenen. Som det ses på figur 47 ligger fps. på 45-55 Hz, når der ikke genereres opdateringer af nogle af modeller. Der benyttes ikke nogen teknikker som frustum culling, occlusion culling eller lignende. Capsella'erne er så små, at de ikke kan ses på disse billeder.
Under opdateringen af LoD modeller kommer applikationen ikke under 20 fps. For at algoritmen kan nå at opdatere et nyt LoD, så skal man ikke bevæge sig ret hurtigt fremad. Algoritmen er ikke hurtigt nok til at generere et nyt LoD før man er gået forbi et niveau og der skal genereres et nyt LoD.

På figur 48 kan man se en Capsella i terrænet:

Figur 48: Et 3. billede fra terrænet

Hvis applikationen og pc'en presses til det yderste kan den klare 200 Honda træer, 50 buske, 20 palmer og 50 capsellaer i en scene. Dog bliver størrelsen af teksturerne sat ned til 64x64, scenen kan ses på figur 49:



Figur 49: terræn med 320 planter

Der kan fint holdes en framerate på 60 Hz, men der bliver benyttet en del ram, ca. 1.7 GB. Dvs. at hver plante i gennemsnit benytter ca. 5 MB hukommelse.

16 Forbedringer/Udvidelser

Modificeringen af teksturerne er et område i algoritmen, der med fordel kan blive optimeret eller ændret. Problemet er at denne del af algoritmen ikke kan udføres hurtigt nok på grund af de operationer, der flytter data mellem de forskellige typer hukommelse. Selve modificeringen er effektiv nok. I forbindelse med projektet, blev det forsøgt at implementere filteret i et fragmentprogram, hvilket ikke gav nogen hastighedsforøgelse. Den 3. måde, der er nævnt i OpenGL implementeringen, vil højst sandsynligt give en hastighedsforøgelse, men den kræver dog stadigt et kald til glCopyTexImage, som er den funktion, der er tidskrævende. Den 3. metode vil maksimalt kunne opnå en fordobling i hastigheden i de tilstande, der benyttes til at modificere teksturerne.
Man kunne prøve de andre metoder der er nævnt i afsnit 11.4 til modificering af teksturerne. En sidste metode, som ikke er nævnt i afsnit 11.4 til antialiasing er supersampling af teksturen. Denne metode vil ikke kunne fjerne den sorte streg, der opstår rundt om modellen, helt men vil nok gøre den mindre.

At pakke alle teksturerne i hver sektion i en fælles tekstur eller benytte en fælles p-buffer vil også være en mulighed, der ikke er undersøgt i denne rapport. Dette vil muligvis give en lille hastighedsforøgelse, når modellen skal renderes, da der ikke skal bindes flere teksturer, dog skal der bestemmes teksturkoordinater for hver sektion.
Dette vil reducere antallet af kald til glCopyTexImage, men spørgsmålet er om hele teksturen kan kopieres på en gang, uden at genere renderingen af den rigtige scene. Det vil muligvis være nødvendigt at kopiere data i sekvenser, men det kan sandsynligvist gøres med færre kald til glCopyTexImage, end hvad der er gjort i implementeringen.

Med hensyn til detaljeringsgraden, har resultatet vist at det er nok med 3 detaljeniveauer. Det er ikke til at se forskel på de mest detaljeret niveauer når der benyttes 5 niveauer.

De 2 billboards, der bliver benyttet til at repræsentere hver sektion, har vist sig at være tilstrækkelige. Den symmetri, der opstår, når modellen ses forfra og bagfra er ikke markant. I afsnit 11.2 er der et forslag til, hvordan man kan variere dette med et normalmap. En anden ting der vil forbedre udseendet af erstatningsmodellen er at de grove dele, som f.eks. stammen, med fordel kan renderes som polygoner, for at sløre at det er billboards, der er benyttet. Dette vil ikke give meget ekstra renderingstid, hvis man kun vælger de tykkeste af grenene ud.

I praksis har det vist sig at algoritmen vil fungere bedre, hvis man først benytter den til at generere alle niveauerne og gemme dem. Hvis dette gøres, vil det også blive muligt at udvikle træet videre ud fra L-systemets regler. Når disse regler er benyttet et tilpas antal gange, kan et nyt sæt af teksturer genereres.

For at gøre modellerne mere virkelighedstro ville subdivision eller en anden metode til udglatning af grenen være interessant. Dette kræver dog at der kan udføres subdivision på meshes, der er opdelt, da det ofte vil være imellem 2 sektioner at det er nødvendigt at foretage sådanne tilpasninger. Der er også

nævnt andre artikler i litteraturafsnittet, der handler om hvordan modellerne gøres mere virkelighedstro, men for alle metoderne gælder det at de kræver en del ressourcer og der skal også huskes på at mange detaljer forsvinder, når modellen projiceres ind på billboards.

17 Konklusion

I denne rapport har jeg dokumenteret en metode til generering af planter i realtid. Selve genereringen af planterne udføres ved hjælp af Lindenmayers systemer. Der findes flere versioner af L-systemer, som er beskrevet i rapporten. Jeg valgte at benytte den stokastiske kontekst frie version af L-systemer, da denne version er tilstrækkelig for de modeller, som jeg valgte at benytte. De andre versioner er dokumenteret i rapporten og vil også kunne implementeres. Af de 4 modeller, der blev benyttet i rapporten, var 3 af dem udvidelser af modeller fra [ABoP] og en modellerede jeg selv.

For at gøre renderingen af planter mulig i realtid er der blevet tilføjet et sæt af nye symboler til L-systemer. Disse nye symboler bliver benyttet til at indføre Level of Detail i modellerne. Selve metoden projicerer modellen ind på et sæt af billboards, som bliver placeret på en måde, der udnytter plantens struktur. De nye symboler inddeler L-systemet i et sektionshierarki.

Som valg af billboard, blev der valgt den statiske version. Det var mest interessant, da det ikke kræver opdateringer alt efter hvilken retning modellen betragtes fra.

Metoden blev delt op i mindre tilstande for at kunne generere erstatnings-modellen i realtid. Denne metode virker, men er ikke helt hurtig nok til at generere erstatningsmodellen, når denne først genereres, når der er brug for den. Denne online rendering af de forskellige detaljeniveauer blev udført ved at benytte en p-buffer, som gør det muligt at udføre off-screen rendering, der ikke påvirker renderingen af den scene, som brugeren skal se.

Jeg er kommet med nogle metoder til hvordan man kan forbedre og udvide metoden til også at inkludere vækst over tid.

Fordelen ved denne metode er at planterne ikke skal genereres i en editor, men bliver genereret udelukkende ud fra et regelsæt. Et regelsæt kan også benyttes til at generere planter der varierer i udseende. Det ses ofte i applikationer at det samme træ bliver genbrugt flere gange, hvilket ikke virker naturligt. Metoden er et alternativ, som ikke benytter en editor til at skabe modeller i. L-systemer er et interessant alternativ, hvor man i stedet skal tænke i matematiske formler og formelle sprog.

Metoden er blevet implementeret vha. OpenGL API'et for at vise hvordan den fungerer i praksis. Den blev implementeret, som et bibliotek, da det skal være nemt for en bruger at implementere i en applikation. I implementeringen er der nogle interfaces, der gør det muligt for en bruger selv at definere elementer, som kan benyttes til at lave regler og objekter som skal inkluderes i systemet.

Metoden har vist at det er muligt at rendere et stort antal unikke træer i en scene i realtid. Rapporten indeholder et eksempel med 320 planter i en scene uden at der opstår problemer med renderingen.

18 Kilder

[ABoP] The Algorithmic Beauty of Plants
Przemyslaw Prusinkiewicz's bog "The Algorithmic Beauty of Plants" fra 1996 (2. version) er hovedkilden til denne opgave. Den indeholde en grundig beskrivelse af L-systemer.
Den er gjort offentlig tilgængelig online på følgen hjemmeside:
http://algorithmicbotany.org/papers/

[ICG] Interactive Computer Graphics, 3rd edition
Af Edward Angel ISBN: 0-321-19044-0

[ICT] Introduction to Computer Theory, second edition,
Af Daniel I.A. Cohen ISBN: 0-471-13772-3

[Poster] Det er tidligere påbegyndt et projekt at en studerende (Bendix Stang), som omhandlede L-Systemer. Dette projekt blev aldrig færdig men i forbindelse med projektet blev der lavet en poster, som gav et eksempel på texturificering i L-systemer.

[IDV] Speed Trees
Information om SpeedTree findes på her: http://www.speedtree.com
Biblioteket er lavet af Interactive Data Visualization, Inc

[BCEMS] Billboard Clouds for Extreme Model Simplification
Af Xavier Décoret, Frédo Duran, François X. Sillion og Julie Dorsey
Handler om hvordan man kan repræsenter en polygon model ved hjælp af et sæt af billboards. De har udviklet en metode til at reducere en polygon model til en sky, som de kalder det, af billboards. Metoden benytter dog en del sortering og optimering derfor er den svær at bruge i et realtidssystem.

[EMSFR] Extreme Model Simplification for Forest Rendering
Af Anton Fuhrmann, Eike Umlauf, Stephan Mantler
Hvordan man kan render hele skove af træer i realtime. Der benyttes metoden som er beskrevet i [BCEMS].

[TfP] Volumetric Reconstruction and Interactive Rendering of Trees from Photographs
Af Alex Reche, Ignacio Martiny og George Drettakis
Beskrivelse af en metode hvor man tager et sæt af fotografier og laver en computer model af et træ ud fra dem.

[MoD] Reverse Engineering: Multilevel-of-Detail Models for Design and Manufacturing
Af A. Fischer og S. Park
Beskriver en metode til hvordan det i polygon mesh kan indbygges LoD

[PbI] Hardware-accelerated Point Generation and Rendering of Point-based Imposters
Af J. Andreas Bærentzen.

Beskriver en metode til hvordan det er muligt at generere en erstatningsmodel vha. punkter.

[MHP] Modelling Hairy Plants
Af Martin Fuhrer, Henrik Wann Jensen og Przemyslaw Prusinkiewicz
Omhandler en metode til at modellere hår på planter, der er benyttet L-systemer til at generer modellerne.

[MPGD] Modeling plant groth and development
Af Przemyslaw Prusinkiewicz
Beskrivelse af hvordan planter gror, vha. L-systemer

[LSC] L-System Description of Subdivision Curves
Af Przemyslaw Prusinkiewicz, Faramarz Samavati, Colin Smith og Radoslaw Karwowski
Beskrivelse af en metode til hvordan man kan udfører subdivision af L-systemer.

[LC] Design and Implementation of the L+C Modeling Language
Af Radoslaw Karwowski og Przemyslaw Prusinkiewicz
Et modelleringssprog udviklet i C++ til at eksekvere regler i et L-system.

[SMPPE] Simulation Modeling og Plants and Plant Ecosystems
Af Przemyslaw Prusinkiewicz
Realistiske planter, beskrivelse af hvordan planter vokser i henhold til lysintensitet og retning.

[E13b] Exercise 02561-13b
Opgave uddelt i forbindelse med kursus 02561 – Computer Graphics i 2001
Af Bent Dalgaard Larsen
Opgaven går ud på at benytte et stykke software som implementere de fleste af L-systemers elementer til at generere planter.

Appendikser

Realtidsrendering – Lindenmayers systemer

August 2005

Appendiks A: Pseudokode

Appendiks B: Kildekode

Appendiks C: Klassediagrammer

Appendiks D: Måledata

Appendiks E: Brugervejledning

Bagerst er der vedlagt en CD med kildekode og eksekverbar kode

Appendiks A

Realtidsrendering – Lindenmayers systemer

Pseudokode

1 CalculateBoundingBox

Pseudokode for bestemmelse af bounding boks for en sektion. Der analyseres om der skal inkluderes undersektioner i linie 4.

CalculateBoundingBox(max_depth)

```
 1   upper_right ← (-∞,-∞,-∞)
 2   lower_left  ← ( ∞, ∞, ∞)
 3
 4   if my_lvl < max_depth
 5     then GetBranchMeshBondingBox(lower_left, upper_right)
 6          GetPolygonsBoundingBox(lower_left, upper_right)
 7          GetSurfacesBoundingBox(lower_left, upper_right)
 8     else
 9         p ←  ((0,0,0), (0,0,0), (0,0,0), (0,0,0), (0,0,0),
10               (0,0,0), (0,0,0), (0,0,0))
11         GetBoundingBoxIncludingSubSections(p, false)
12         for i ← 1 to 8
13           do for j ← 1 to 3
14                do if lower_left[j] > p[i][j]
15                     then lower_left[j] ← p[i][j]
16                   if upper_right[j] < p[i] [j]
17                     then upper_right[j] ← p[i][j]
18
19   xLength ← upper_right[1] – lower_left[1]
20   yLength ← upper_right[2] – lower_left[2]
21   zLength ← upper_right[3] – lower_left[3]
22
23   x ← lower_left[1]
24   y ← lower_left[2]
25   z ← lower_left[3]
```

2 GetBoundingBoxIncludingSubSections

De 2 argumenter som denne metode skal bruge er ’`points`’, som er en vektor med 8 punkter, der repræsenterer de 8 hjørner på bounding boksen. Resultat vil blive skrevet til denne variabel. Det andet argument ’`translate_coordinates`’, fortæller om den bounding boks der bestemmes skal transformeres til forælderens koordinatsystem. Denne transformation skal den altid anvendes, hvis det er for en sektion, der bliver slået sammen med en øvre sektion i sektionshierarkiet.

GetBoundingBoxIncludingSubSections(points, translate_coordinates)

```
 1   bb ← ((0,0,0), (0,0,0), (0,0,0), (0,0,0), (0,0,0),
 2          (0,0,0), (0,0,0), (0,0,0))
 3   u ← (-∞,-∞,-∞)
 4   l ← ( ∞, ∞, ∞)
 5   GetBranchMeshBondingBox(l, u)
 6   GetPolygonsBoundingBox(l, u)
 7   GetSurfacesBoundingBox(l, u)
 8   for s ← 1 to SubSections.Size
 9     do SubSections[s].
10                 GetBoundingBoxIncludingSubSections(bb, true)
11       for i ← 1 to 8
12         do for j ← 1 to 3
13              do if l > bb[i][j]
14                   then l ← bb[i][j]
15                 if u < bb[i][j]
16                   then u ← bb[i][j]
17
18   points[1] ← (l[0], l[1], l[2])
19   points[2] ← (l[0], l[1], u[2])
20   points[3] ← (l[0], u[1], l[2])
21   points[4] ← (l[0], u[1], u[2])
22   points[5] ← (u[0], l[1], l[2])
23   points[6] ← (u[0], l[1], u[2])
24   points[7] ← (u[0], u[1], l[2])
25   points[8] ← (u[0], u[1], u[2])
26
27   if translate_coordinates
28     then for i ← 1 to 8
29            do points[i] ← M0 * points[i]
```

`M0` er matricen, der transformerer koordinaterne til forælderens koordinatsystem.

3 GetBranchMeshBoundingBox

Denne metode viser en helt generel søgning gennem alle grenenes punkter. Resultatet gemmes i de to variable, som den tager som argument. Argumenterne er 2 vektorer.
I implementeringen bliver denne metode udført løbende samtidigt med at grene bliver tilføjet og gemt for at undgå at bounding boksen skal bestemmes flere gange.

GetBranchMeshBoundingBox(lower_left, upper_right)

```
1   for i ← 0 to BranchMesh.numberOfVertecies
2     do for j ← 0 to 3
3        do if lower_left[j] > BranchMesh.getVertex(i)[j]
4            then lower_left[j] ← BranchMesh.getVertex(i)[j]
5           if upper_right[j] < BranchMesh.getVertex(i)[j]
6            then upper_right[j] ← BranchMesh.getVertex(i)[j]
```

4 GetPolygonsBoundingBox

Samme princip, som blev anvendt for grenene til at bestemme bounding boksen bliver anvendet for hvert af polygonerne.

GetPolygonsBoundingBox(lower_left, upper_right)

```
1   for p ← to Polygons.numberOfPolygons
2    do for i ← 0 to Polygons[p].numberOfVertecies
3      do for j ← 0 to 3
4         do if lower_left[j] > BranchMesh.getVertex(i)[j]
5           then lower_left[j] ← BranchMesh.getVertex(i)[j]
6            if upper_right[j] < BranchMesh.getVertex(i)[j]
7           then upper_right[j] ← BranchMesh.getVertex(i)[j]
```

5 GetSurfacesBoundingBox

Denne metode gennemløber alle objekter igennem for at få bounding boksen fra hver af dem. Ud fra alle bounding boksene bliver der lavet en fælles bounding boks.

GetSurfacesBoundingBox(lower_left, upper_right)

```
1   u ← (-∞,-∞,-∞)
2   l ← (∞,∞,∞)
3
4   for s ← to Surfaces.numberOfSurfaces
5     Surfaces[s].getBoundingBox(l, u)
6     do for j ← 0 to 3
7          do if lower_left[j] > l[j]
8               then lower_left[j] ← l[j]
9             if upper_right[j] < u[j]
10              then upper_right[j] ← u[j]
```

Appendiks B

Realtidsrendering – Lindenmayers systemer

Kildekode

1 Common.h

```
1   #ifndef _RTLS_COMMON_H_
2   #define _RTLS_COMMON_H_
3
4   namespace RealTimeLSystem {
5
6       #define VERTECIES_PER_BRANCH 8
7       #define TEXTURE_SIZE 128
8   }
9
10  namespace RTLS = RealTimeLSystem;
11
12  #endif
```

2 Branch.h

```
1   #ifndef _BRANCH_H_
2   #define _BRANCH_H_
3
4   #include <GL/glew.h>
5   #include <vector>
6
7   #include "CGLA/Vec3f.h"
8   #include "CGLA/Vec2f.h"
9   #include "CGLA/Vec3i.h"
10  #include "CGLA/Vec2i.h"
11
12  #include "Common.h"
13  #include "RTLS_Material.h"
14
15  namespace RealTimeLSystem {
16
17      class BranchMesh {
18      public:
19          BranchMesh(void);
20          ~BranchMesh(void);
21
22          int addVertex(const CGLA::Vec3f& vert,
23                        const RTLS::RTLS_Material& material,
24                        const CGLA::Vec3f& normal);
25
26          CGLA::Vec3f getVert(int i);
27
28          int addFace(const CGLA::Vec3i& face);
29
30          int addFace(int, int, int);
31
32          int add_tFace(const CGLA::Vec3i& tface);
33
34          int add_texCoord(const CGLA::Vec2f& texCoord);
35
36          void getBBox(CGLA::Vec3f& p0, CGLA::Vec3f& p1);
37
38          void drawAll() const;
39
40          bool drawSome(int amount);
41
42          int getNumberOfPolygons();
43
44      private:
45
46          int counter;
47
48          CGLA::Vec3f bb0, bb1;
49
50          std::vector<CGLA::Vec3i> faces;
51          std::vector<CGLA::Vec3f> vertices;
52          std::vector<CGLA::Vec3f> normals;
53          std::vector<CGLA::Vec2f> texCoords;
54          std::vector<CGLA::Vec3i> tfaces;
55          std::vector<RTLS::RTLS_Material> materials;
56      };
57  }
58  #endif
```

3 Branch.cpp

```
#include "branch.h"

using namespace CGLA;
using namespace std;

namespace RealTimeLSystem {

    BranchMesh::BranchMesh() {
        bb0 = Vec3f(0,0,0);
        bb1 = Vec3f(0,0,0);
        counter = 0;
    }

    BranchMesh::~BranchMesh(void) {
        vertices.clear();
        faces.clear();
        tfaces.clear();
        texCoords.clear();
    }

    void BranchMesh::drawAll() const {
        Vec3i face, tface;
        RTLS_Material material;

        glPushAttrib(GL_ALL_ATTRIB_BITS);

        glBegin(GL_TRIANGLES);
        for(int f=0; f < (int) faces.size(); f++) {
            face = (Vec3i) faces[f];
            tface = (Vec3i) tfaces[f];
            for(int v = 0; v<3; v++) {
                material = materials[face[v]];

                glMaterialfv(GL_FRONT, GL_AMBIENT,
                        material.getAmbientColor().get());
                glMaterialfv(GL_FRONT, GL_DIFFUSE,
                        material.getDiffuseColor().get());
                glMaterialfv(GL_FRONT, GL_SPECULAR,
                        material.getSpecularColor().get());
                glMaterialf(GL_FRONT, GL_SHININESS,
                        material.getShininess());

                glTexCoord2fv(texCoords[tface[v]].get());
                glNormal3fv(normals[face[v]].get());
                glVertex3fv(vertices[face[v]].get());
            }
        }
        glEnd();

        glPopAttrib();
    }

    bool BranchMesh::drawSome(int amount) {
        Vec3i face, tface;
        RTLS_Material material;

        int end = counter + amount;
        bool all_drawn = false;
        if(end > (int) faces.size()) {
```

```
60              end = (int) faces.size();
61              all_drawn = true;
62          }
63          if(counter == 0) {
64              glPushAttrib(GL_ALL_ATTRIB_BITS);
65          }
66
67          glBegin(GL_TRIANGLES);
68          while(counter < end) {
69              face = (Vec3i) faces[counter];
70              tface = (Vec3i) tfaces[counter];
71              for(int v = 0; v<3; v++) {
72                  material = materials[face[v]];
73
74                  glMaterialfv(GL_FRONT, GL_AMBIENT,
75                          material.getAmbientColor().get());
76                  glMaterialfv(GL_FRONT, GL_DIFFUSE,
77                          material.getDiffuseColor().get());
78                  glMaterialfv(GL_FRONT, GL_SPECULAR,
79                          material.getSpecularColor().get());
80                  glMaterialf (GL_FRONT, GL_SHININESS,
81                          material.getShininess());
82
83                  glTexCoord2fv(texCoords[tface[v]].get());
84                  glNormal3fv(normals[face[v]].get());
85                  glVertex3fv(vertices[face[v]].get());
86              }
87              counter++;
88          }
89          glEnd();
90
91          if(all_drawn) {
92              glPopAttrib();
93              counter = 0;
94          }
95
96          return all_drawn;
97      }
98
99      int BranchMesh::addVertex(const Vec3f& vert,
100                                const RTLS_Material& material,
101                                const Vec3f& normal) {
102         for(int j=0; j<3; j++) {
103             if(vert[j] < bb0[j]) {
104                 bb0[j] = vert[j];
105             }
106             if(vert[j] > bb1[j]) {
107                 bb1[j] = vert[j];
108             }
109         }
110         vertices.push_back(vert);
111         normals.push_back(normal);
112         materials.push_back(material);
113         return vertices.size()-1;
114     }
115
116     int BranchMesh::addFace(const Vec3i& face) {
117         faces.push_back(face);
118         return faces.size()-1;
119     }
120
```

```
121         int BranchMesh::addFace(int v0, int v1, int v2) {
122             faces.push_back(Vec3i(v0,v1,v2));
123             return faces.size()-1;
124         }
125 
126         void BranchMesh::getBBox(Vec3f& p0, Vec3f& p1) {
127 
128             p0 = bb0;
129             p1 = bb1;
130         }
131 
132         int BranchMesh::add_tFace(const CGLA::Vec3i& tface) {
133             assert(faces.size()-1==tfaces.size());
134             tfaces.push_back(tface);
135             return tfaces.size()-1;
136         }
137 
138         int BranchMesh::add_texCoord(
139                                 const CGLA::Vec2f& texCoord) {
140             texCoords.push_back(texCoord);
141             return texCoords.size()-1;
142         }
143 
144         Vec3f BranchMesh::getVert(int i) {
145             assert(i<vertices.size());
146             return vertices[i];
147         }
148 
149         int BranchMesh::getNumberOfPolygons() {
150             return faces.size();
151         }
152     }
```

4 Element.h

```
1   #ifndef _ELEMENT_H_
2   #define _ELEMENT_H_
3
4   #include "Common.h"
5   #include "Tree.h"
6
7   namespace RealTimeLSystem {
8
9       class Element {
10      public:
11          virtual ~Element() = 0;
12          virtual void print() = 0;
13          virtual void execute(Tree* tree) = 0;
14      };
15
16      class F : public Element {
17      public:
18          F() : l(1) {};
19          F(float _l) : l(_l) {};
20          void execute(Tree* tree);
21          void print();
22
23          float l;
24      };
25
26      class f : public Element {
27      public:
28          f() : l(1) {};
29          f(float _l) : l(_l) {};
30          void execute(Tree* tree);
31          void print();
32
33          float l;
34      };
35
36      class Size : public Element {
37      public:
38          Size() : w(0.2), absolute(false) {};
39          Size(float _w) : w(_w), absolute(true) {};
40          void execute(Tree* tree);
41          void print();
42
43          bool absolute;
44          float w;
45      };
46
47      class And : public Element {
48      public:
49          And(float _l) : l(_l) {};
50          void execute(Tree* tree);
51          void print();
52
53          float l;
54      };
55
56      class Hat : public Element {
57      public:
58          Hat(float _l) : l(_l) {};
59          void execute(Tree* tree);
60          void print();
```

```
61
62          float l;
63      };
64
65      class Uturn : public Element {
66      public:
67          Uturn() {};
68          void execute(Tree* tree);
69          void print();
70      };
71
72      class Minus : public Element {
73      public:
74          Minus(float _l) : l(_l) {};
75          void execute(Tree* tree);
76          void print();
77
78          float l;
79      };
80
81      class Plus : public Element {
82      public:
83          Plus(float _l) : l(_l) {};
84          void execute(Tree* tree);
85          void print();
86
87          float l;
88      };
89
90      class Slash : public Element {
91      public:
92          Slash(float _l) : l(_l) {};
93          void execute(Tree* tree);
94          void print();
95
96          float l;
97      };
98
99      class Backslash : public Element {
100     public:
101         Backslash(float _l) : l(_l) {};
102         void execute(Tree* tree);
103         void print();
104
105         float l;
106     };
107
108     class Push : public Element {
109     public:
110         Push() {};
111         void execute(Tree* tree);
112         void print();
113     };
114
115     class Pop : public Element {
116     public:
117         Pop() {};
118         void execute(Tree* tree);
119         void print();
120     };
121
```

```
122     class PushPolygon : public Element {
123     public:
124         PushPolygon() {};
125         void execute(Tree* tree);
126         void print();
127     };
128
129     class PopPolygon : public Element {
130     public:
131         PopPolygon() {};
132         void execute(Tree* tree);
133         void print();
134     };
135
136     class G : public Element {
137     public:
138         G() : l(1) {};
139         G(float _l) : l(_l) {};
140         void execute(Tree* tree);
141         void print();
142
143         float l;
144     };
145
146     class Dot : public Element {
147     public:
148         Dot() {};
149         void execute(Tree* tree);
150         void print();
151     };
152
153     class Percent : public Element {
154     public:
155         Percent() {};
156         void execute(Tree* tree);
157         void print();
158     };
159
160     class DollarSign : public Element {
161     public:
162         DollarSign() {};
163         void execute(Tree* tree);
164         void print();
165     };
166
167     class Color : public Element {
168     public:
169         Color(CGLA::Vec3f _color): color(_color),
170                                    absolute(true) {};
171         Color(float r, float g, float b):
172                                  color(CGLA::Vec3f(r,g,b)),
173                                  absolute(true) {};
174         Color(): color(CGLA::Vec3f(1.02f, 1.02f, 1.02f)),
175                       absolute(false) {};
176         void execute(Tree* tree);
177         void print();
178
179         CGLA::Vec3f color;
180         bool absolute;
181     };
182
```

```
183         class ApplySurface : public Element {
184         public:
185             ApplySurface(RTLS::Surface* surf): surface(surf){};
186             void execute(Tree* tree);
187             void print();
188 
189             RTLS::Surface* surface;
190         };
191 
192         class PushLoD : public Element {
193         public:
194             PushLoD() {};
195             void execute(Tree* tree);
196             void print();
197         };
198 
199         class PopLoD : public Element {
200         public:
201             PopLoD() {};
202             void execute(Tree* tree);
203             void print();
204         };
205 
206         class Texture : public Element {
207         public:
208             Texture(GLuint _tex): tex(_tex) {};
209             void execute(Tree* tree);
210             void print();
211             GLuint tex;
212         };
213 
214         class Material : public Element {
215         public:
216             Material(RTLS::RTLS_Material& mat):material(mat){};
217             void execute(Tree* tree);
218             void print();
219             RTLS::RTLS_Material material;
220         };
221     }
222 
223     #endif
```

5 Element.cpp

```
1   #include "Element.h"
2   #include <iostream>
3   #include <vector>
4
5   using namespace std;
6   using namespace CGLA;
7
8   namespace RealTimeLSystem {
9
10      Element::~Element() {
11      }
12      void F::execute(Tree* tree) {
13          tree->F(l);
14      }
15      void f::execute(Tree* tree) {
16          tree->f(l);
17      }
18      void Size::execute(Tree* tree) {
19          tree->Size(w, absolute);
20      }
21      void And::execute(Tree* tree) {
22          tree->And(l);
23      }
24      void Hat::execute(Tree* tree) {
25          tree->Hat(l);
26      }
27      void Uturn::execute(Tree* tree) {
28          tree->Uturn();
29      }
30      void Minus::execute(Tree* tree) {
31          tree->Minus(l);
32      }
33      void Plus::execute(Tree* tree) {
34          tree->Plus(l);
35      }
36      void Slash::execute(Tree* tree) {
37          tree->Slash(l);
38      }
39      void Backslash::execute(Tree* tree) {
40          tree->Backslash(l);
41      }
42      void Push::execute(Tree* tree) {
43          tree->Push();
44      }
45      void Pop::execute(Tree* tree) {
46          tree->Pop();
47      }
48      void PushPolygon::execute(Tree* tree) {
49          tree->PushPolygon();
50      }
51      void PopPolygon::execute(Tree* tree) {
52          tree->PopPolygon();
53      }
54      void G::execute(Tree* tree) {
55          tree->G(l);
56      }
57      void Dot::execute(Tree* tree) {
58          tree->Dot();
59      }
60      void Percent::execute(Tree* tree) {
```

```
61          tree->Percent();
62      }
63      void DollarSign::execute(Tree* tree) {
64          tree->DollarSign();
65      }
66      void Color::execute(Tree* tree) {
67          tree->Color(color, absolute);
68      }
69      void ApplySurface::execute(Tree* tree) {
70          tree->ApplySurface(surface);
71      }
72      void PushLoD::execute(Tree* tree) {
73          tree->PushLoD();
74      }
75      void PopLoD::execute(Tree* tree) {
76          tree->PopLoD();
77      }
78      void Texture::execute(Tree* tree) {
79          tree->setTexture(tex);
80      }
81      void Material::execute(Tree* tree) {
82          tree->setMaterial(material);
83      }
84
85      /* print */
86
87      void F::print () {
88          cout << "F(" << l << ")";
89      }
90      void f::print () {
91          cout << "f(" << l << ")";
92      }
93      void Size::print () {
94          cout << "!(" << w << ")";
95      }
96      void And::print () {
97          cout << "&";
98      }
99      void Hat::print () {
100         cout << "^";
101     }
102     void Uturn::print () {
103         cout << "|";
104     }
105     void Minus::print () {
106         cout << "-";
107     }
108     void Plus::print () {
109         cout << "+";
110     }
111     void Slash::print () {
112         cout << "/";
113     }
114     void Backslash::print () {
115         cout << "\\";
116     }
117     void Push::print () {
118         cout << "[";
119     }
120     void Pop::print () {
121         cout << "]";
```

```
122         }
123         void PushPolygon::print () {
124             cout << "{";
125         }
126         void PopPolygon::print () {
127             cout << "}";
128         }
129         void Dot::print () {
130             cout << ".";
131         }
132         void Percent::print () {
133             cout << "%";
134         }
135         void DollarSign::print () {
136             cout << "$";
137         }
138         void G::print () {
139             cout << "G(" << l << ")";
140         }
141         void Color::print() {
142             cout << "'(" << color[0] << "," << color[1] << ","
143                  << color[2] << ")";
144         }
145         void ApplySurface::print() {
146             cout << "~";
147         }
148         void PushLoD::print () {
149             cout << "<";
150         }
151         void PopLoD::print () {
152             cout << ">";
153         }
154         void Texture::print () {
155             cout << "#";
156         }
157         void Material::print () {
158             cout << "@";
159         }
160     }
```

6 LoDSection.h

```
1   #ifndef _LODSECTION_H_
2   #define _LODSECTION_H_
3
4   #include <vector>
5
6   #include "CGLA/Vec3f.h"
7   #include "CGLA/Vec3i.h"
8   #include "CGLA/Mat4x4f.h"
9   #include "CGLA/Quaternion.h"
10
11  #include "RenderTexture/RenderTexture.h"
12
13  #include "Common.h"
14  #include "Timer.h"
15  #include "Polygon.h"
16  #include "Branch.h"
17  #include "Surface.h"
18  #include "RTLS_Material.h"
19
20  namespace RealTimeLSystem {
21
22      class LoDSection {
23
24      public:
25          LoDSection(RTLS::LoDSection* parent,int lvl,
26                     unsigned char* tmp_tex,
27                     RTLS::Timer* _timer);
28          ~LoDSection(void);
29
30          void F(float l);
31          void f(float l);
32          void Plus(float l);
33          void Minus(float l);
34          void And(float l);
35          void Hat(float l);
36          void Uturn();
37          void Backslash(float l);
38          void Slash(float l);
39          void Pipe();
40          void DollarSign();
41          void Push();
42          void Pop();
43          void PushPolygon();
44          void PopPolygon();
45          void G(float l);
46          void Dot();
47          void Size(float w, bool     absolute);
48          void Color(CGLA::Vec3f color, bool absolute);
49          void ApplySurface(RTLS::Surface*  surf);
50
51          bool isAligning() {   return aligning; }
52          void forceEndAligning();
53
54          void drawTexturified(int maxLod);
55          void setLOD(int lod);
56          void addBranch(const CGLA::Mat4x4f start_mat,
57                         const CGLA::Mat4x4f end_mat,
58                         const RTLS::RTLS_Material material,
59                         const float width,
```

```
60                          const CGLA::Vec3f direction =
61                                                  CGLA::Vec3f(0,0,0));
62          void addPolygon(RTLS::Polygon polygon);
63
64          bool getBBoxIncludingSubLoDs(CGLA::Vec3f* p,
65                       bool translate_to_parent_coordinates);
66
67          bool createTexture(int max_depth);
68
69          const float     getCurrentSize() const;
70          const RTLS::RTLS_Material getCurrentMaterial()
71                                                          const;
72          const CGLA::Mat4x4f   getCurrentMatrix() const;
73
74          void addSubLoD(LoDSection* obj);
75
76          bool drawRelative(); //relative to parent lod
77
78          void drawBBox(int max_depth);
79
80          int getLvl() { return my_lvl; }
81
82          void swapTexture();
83
84          void justDraw();
85
86          void setBark(const GLuint bark) { barktex = bark; }
87
88          const GLuint getBark() const { return barktex; }
89
90          void setMaterial(const RTLS_Material& mat);
91
92          const CGLA::Vec3f getDollarVector() const {
93                                       return dollarVector; }
94
95          void setDollarVector(const CGLA::Vec3f& dollarVec)
96                           { dollarVector = dollarVec; }
97
98          const CGLA::Vec4f getLightPosition() const {
99                                     return light_position; }
100
101         void setLightPosition(const CGLA::Vec4f& light_pos)
102                              { light_position = light_pos; }
103
104         int getNumberOfPolygons();
105
106         LARGE_INTEGER* getTimings();
107
108         char* getStateLabel(int state);
109
110     private:
111
112         GLuint barktex;
113
114         enum STATE {
115             IDLE                =    0,
116             GETTING_BBOX_INIT   =    1,
117             GETTING_BBOX_BRANCHES =  2,
118             GETTING_BBOX_POLYGONS =  3,
119             GETTING_BBOX_SURFACES =  4,
120             GETTING_BBOX_SUB_INIT =  5,
```

```
121             GETTING_BBOX_SUB      =  6,
122             BBOX_DONE             =  7,
123             RENDERING_INIT        =  8,
124             RENDERING_BRANCHES    =  9,
125             RENDERING_POLYGONS    = 10,
126             RENDERING_SURFACES    = 11,
127             RENDERING_SUB         = 12,
128             READ_TEXTURE          = 13,
129             EDIT_TEXTURE          = 14,
130             CREATE_TEXTURE        = 15
131         };
132
133         void printState() const;
134
135         inline float degree2radian(float degree) {
136             return degree *   M_PI / 180.0f;
137         }
138
139         inline int next2border(int x0, int y0) {
140             int   neightbours = 0;
141             if(tmp_texture[x0*TEXTURE_SIZE*4+y0*4+3]!=0) {
142                 return 0;
143             }
144             if(y0 > 0 &&
145               tmp_texture[x0*TEXTURE_SIZE*4+(y0-1)*4+3]>0){
146                 neightbours++;
147             }
148             if(y0 < TEXTURE_SIZE-1 &&
149               tmp_texture[x0*TEXTURE_SIZE*4+(y0+1)*4+3]>0){
150                 neightbours++;
151             }
152             if(x0 > 0) {
153                if(tmp_texture[(x0-1)*TEXTURE_SIZE*4+y0*4+3]
154                   > 0) {
155                    neightbours++;
156                }
157                if(y0 > 0 &&
158                   tmp_texture[(x0-1)*TEXTURE_SIZE*4+
159                               (y0-1)*4+3] > 0) {
160                    neightbours++;
161                }
162                if(y0 < TEXTURE_SIZE-1 &&
163                   tmp_texture[(x0-1)*TEXTURE_SIZE*4+
164                               (y0+1)*4+3] > 0) {
165                    neightbours++;
166                }
167             }
168             if(x0 < TEXTURE_SIZE-1) {
169                if(tmp_texture[(x0+1)*TEXTURE_SIZE*4+y0*4+3]
170                   > 0) {
171                    neightbours++;
172                }
173                if(y0 > 0 &&
174                   tmp_texture[(x0+1)*TEXTURE_SIZE*4+
175                               (y0-1)*4+3] > 0) {
176                    neightbours++;
177                }
178                if(y0 < TEXTURE_SIZE-1 &&
179                   tmp_texture[(x0+1)*TEXTURE_SIZE*4+
180                   (y0+1)*4+3] > 0) {
181                    neightbours++;
```

```
182                 }
183             }
184             return neightbours;
185         }
186
187         inline void applyFilter(int x0, int y0,
188                                 int neighbours) {
189             float div_by = neighbours;
190
191             if(y0 > 0 && tmp_texture[x0*TEXTURE_SIZE*4+
192                                      (y0-1)*4+3] > 0) {
193                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
194                 tmp_texture[x0*TEXTURE_SIZE*4+(y0-1)*4+0] /
195                                                     div_by;
196                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
197                 tmp_texture[x0*TEXTURE_SIZE*4+(y0-1)*4+1] /
198                                                     div_by;
199                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
200                 tmp_texture[x0*TEXTURE_SIZE*4+(y0-1)*4+2] /
201                                                     div_by;
202             }
203             if(y0 < TEXTURE_SIZE-1 &&
204                tmp_texture[x0*TEXTURE_SIZE*4+(y0+1)*4+3]
205                > 0) {
206                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
207                 tmp_texture[x0*TEXTURE_SIZE*4+(y0+1)*4+0] /
208                                                     div_by;
209                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
210                 tmp_texture[x0*TEXTURE_SIZE*4+(y0+1)*4+1] /
211                                                     div_by;
212                 tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
213                 tmp_texture[x0*TEXTURE_SIZE*4+(y0+1)*4+2] /
214                                                     div_by;
215             }
216             if(x0 > 0) {
217                 if(tmp_texture[(x0-1)*TEXTURE_SIZE*4+
218                                y0*4+3] > 0) {
219                   tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
220                   tmp_texture[(x0-1)*TEXTURE_SIZE*4+y0*4+0]
221                                                   / div_by;
222                   tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
223                   tmp_texture[(x0-1)*TEXTURE_SIZE*4+y0*4+1]
224                                                   / div_by;
225                   tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
226                   tmp_texture[(x0-1)*TEXTURE_SIZE*4+y0*4+2]
227                                                   / div_by;
228                 }
229                 if(y0 > 0 &&
230                    tmp_texture[(x0-1)*TEXTURE_SIZE*4+
231                                (y0-1)*4+3] > 0) {
232                    tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
233                    tmp_texture[(x0-1)*TEXTURE_SIZE*4+
234                                (y0-1)*4+0] / div_by;
235                    tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
236                    tmp_texture[(x0-1)*TEXTURE_SIZE*4+
237                                (y0-1)*4+1] / div_by;
238                    tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
239                    tmp_texture[(x0-1)*TEXTURE_SIZE*4+
240                                (y0-1)*4+2] / div_by;
241                 }
242                 if(y0 < TEXTURE_SIZE-1 &&
```

```
243                     tmp_texture[(x0-1)*TEXTURE_SIZE*4+
244                                 (y0+1)*4+3] > 0) {
245                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
246                     tmp_texture[(x0-1)*TEXTURE_SIZE*4+
247                                 (y0+1)*4+0] / div_by;
248                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
249                     tmp_texture[(x0-1)*TEXTURE_SIZE*4+
250                                 (y0+1)*4+1] / div_by;
251                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
252                     tmp_texture[(x0-1)*TEXTURE_SIZE*4+
253                                 (y0+1)*4+2] / div_by;
254                  }
255               }
256               if(x0 < TEXTURE_SIZE-1) {
257
258                 if(tmp_texture[(x0+1)*TEXTURE_SIZE*4+y0*4+3]
259                    > 0) {
260                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
261                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
262                                 y0*4+0] / div_by;
263                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
264                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
265                                 y0*4+1] / div_by;
266                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
267                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
268                                 y0*4+2] / div_by;
269                  }
270                  if(y0 > 0 &&
271                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
272                                 (y0-1)*4+3] > 0) {
273                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
274                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
275                                 (y0-1)*4+0] / div_by;
276                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
277                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
278                                 (y0-1)*4+1] / div_by;
279                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
280                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
281                                 (y0-1)*4+2] / div_by;
282                  }
283                  if(y0 < TEXTURE_SIZE-1 &&
284                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
285                                 (y0+1)*4+3] > 0) {
286                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+0] +=
287                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
288                                 (y0+1)*4+0] / div_by;
289                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+1] +=
290                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
291                                 (y0+1)*4+1] / div_by;
292                     tmp_texture[x0*TEXTURE_SIZE*4+y0*4+2] +=
293                     tmp_texture[(x0+1)*TEXTURE_SIZE*4+
294                                 (y0+1)*4+2] / div_by;
295                  }
296               }
297            }
298
299            inline void     changeState(STATE to) {
300
301   #ifdef _TIMING
302                endTimer();
303   #endif
```

```
304             state =     to;
305
306     #ifdef _TIMING
307             startTimer();
308     #endif
309         }
310
311         void renderNewTextures(int max_depth);
312
313         void calculateBBox(int max_depth);
314
315         bool getSurfacesBBox(CGLA::Vec3f& lower_left,
316                              CGLA::Vec3f& upper_right);
317
318         bool getPolygonsBBox(CGLA::Vec3f& lower_left,
319                              CGLA::Vec3f& upper_right);
320
321         RTLS::Timer* timer;
322
323         unsigned char* tmp_texture;
324
325         int xt,yt;
326
327         RTLS::LoDSection* parent;
328
329         STATE state;
330
331         GLuint* current_texture[2];
332         GLuint* next_texture[4];
333
334         bool first2nextTextures;
335
336         int my_lvl, t;
337
338         int* last_vertecies;
339
340         bool aligning;
341
342         float xLength, yLength, zLength;
343         float x, y, z;
344
345         CGLA::Vec3f dollarVector;
346         CGLA::Vec4f light_position;
347
348         CGLA::Vec3f v0, v1;
349
350         CGLA::Vec3f *bb, *res;
351
352         CGLA::Vec3f upper_right, lower_left; //bounding box
353
354         CGLA::Vec4f pos[2][4];
355
356         BranchMesh branchMesh;
357
358         int counter;
359
360         int texCoord00, texCoord10, texCoord11, texCoord01;
361
362         std::vector<LoDSection*>::iterator iter;
363
364         std::vector<LoDSection*> subObjects;
```

```
365             std::vector<RTLS::Polygon> polygons;
366             std::vector<RTLS::Surface*> surfaces;
367             std::vector<CGLA::Mat4x4f> surface_mats;
368
369             std::vector<CGLA::Mat4x4f> mat_stack;
370             std::vector<RTLS::Polygon> polygon_stack;
371             std::vector<float> size_stack;
372             std::vector<RTLS::RTLS_Material>  material_stack;
373             std::vector<int*> last_vertecies_stack;
374
375             LARGE_INTEGER* timing;
376             void startTimer();
377             void endTimer();
378             LARGE_INTEGER start_clk, end_clk;
379         };
380     }
381
382     #endif
```

7 LoDSection.cpp

```
#include "LoDSection.h"

using namespace std;
using namespace CGLA;


namespace RealTimeLSystem {

    LoDSection::LoDSection(LoDSection* _parent, int lvl,
                           unsigned char* tmp_tex,
                           Timer* _timer) :
                parent(_parent), my_lvl(lvl),
                tmp_texture(tmp_tex), timer(_timer),
                state(IDLE), t(0) {

        if(parent != NULL) {
            mat_stack.push_back(
                              parent->getCurrentMatrix());
            size_stack.push_back(parent->getCurrentSize());
            material_stack.push_back(
                               parent->getCurrentMaterial());
        } else {
            mat_stack.push_back(identity_Mat4x4f());
            size_stack.push_back(0);
            material_stack.push_back(RTLS_Material());
        }

        dollarVector = Vec3f(0,0,0);

        branchMesh = BranchMesh();

        texCoord00 =
                branchMesh.add_texCoord(Vec2f(0.0f,0.0f));
        texCoord10 =
                branchMesh.add_texCoord(Vec2f(1.0f,0.0f));
        texCoord11 =
                branchMesh.add_texCoord(Vec2f(1.0f,1.0f));
        texCoord01 =
                branchMesh.add_texCoord(Vec2f(0.0f,1.0f));

        last_vertecies = 0;
        last_vertecies_stack.push_back(last_vertecies);

        aligning = true;

        for(int i=0; i<4; i++) {
            next_texture[i] = new GLuint();
            glGenTextures(1, next_texture[i]);
        }

        first2nextTextures = true;

        current_texture[0] = NULL;
        current_texture[1] = NULL;

        t=0;

        v0 = Vec3f(0);
        v1 = Vec3f(0);
```

```
60
61  #ifdef _TIMING
62          timing = new LARGE_INTEGER[16];
63  #endif
64      }
65
66      LoDSection::~LoDSection(void) {
67          mat_stack.clear();
68          size_stack.clear();
69          material_stack.clear();
70          last_vertecies_stack.clear();
71          surfaces.clear();
72          surface_mats.clear();
73          for(int i=0; i<4; i++) {
74              glDeleteTextures(1, next_texture[i]);
75          }
76          delete [] *current_texture, *next_texture;
77          delete [] last_vertecies;
78          delete [] timing;
79      }
80
81      void LoDSection::forceEndAligning() {
82          if(!aligning) {
83              return;
84          }
85          if(parent != NULL) {
86              Vec4f dollarVec4f =
87                       Vec4f(parent->getDollarVector(),1.0f);
88              Mat4x4f imat = mat_stack[0];
89              imat[0][3] = 0;
90              imat[1][3] = 0;
91              imat[2][3] = 0;
92              imat = invert(imat);
93              dollarVec4f = imat * dollarVec4f;
94              dollarVector =
95                      Vec3f(dollarVec4f[0]/dollarVec4f[3],
96                            dollarVec4f[1]/dollarVec4f[3],
97                            dollarVec4f[2]/dollarVec4f[3]);
98
99              light_position = parent->getLightPosition();
100             imat = invert(mat_stack[0]);
101             light_position = imat * light_position;
102         }
103
104         mat_stack.push_back(identity_Mat4x4f());
105         aligning = false;
106     }
107
108     void LoDSection::F(float l) {
109         if(aligning) {
110             forceEndAligning();
111         }
112         Mat4x4f start_mat = mat_stack[mat_stack.size()-1];
113
114         Mat4x4f end_mat;
115
116         if(l > 0) {
117
118             end_mat =
119                start_mat*translation_Mat4x4f(Vec3f(l,0,0));
120
```

```
121             mat_stack[mat_stack.size()-1] = end_mat;
122
123         }
124
125         if(polygon_stack.size() > 0) {
126             Vec4f end_pos;
127             if(l > 0) {
128                 end_pos = end_mat * Vec4f(0,0,0,1);
129             } else {
130                 end_pos = start_mat * Vec4f(0,0,0,1);
131             }
132             polygon_stack[polygon_stack.size()-1]
133                                         .addPoint(end_pos);
134         } else {
135             if(l > 0) {
136                 addBranch(start_mat, end_mat,
137                    material_stack[material_stack.size()-1],
138                    size_stack[size_stack.size()-1]);
139             } else {
140                 Vec3f h = Vec3f(start_mat[0][0],
141                                 start_mat[1][0],
142                                 start_mat[2][0]);
143
144                 addBranch(start_mat, start_mat,
145                    material_stack[material_stack.size()-1],
146                    size_stack[size_stack.size()-1], h);
147             }
148         }
149     }
150
151     void LoDSection::f(float l) {
152         if(aligning) {
153             forceEndAligning();
154         }
155         if(l > 0) {
156             Mat4x4f mat = mat_stack[mat_stack.size()-1];
157
158             mat = mat * translation_Mat4x4f(Vec3f(l,0,0));
159
160             mat_stack[mat_stack.size()-1] = mat;
161
162             if(polygon_stack.size() > 0) {
163                 Vec4f end_pos   =   mat * Vec4f(0,0,0,1);
164                 polygon_stack[polygon_stack.size()-1]
165                                         .addPoint(end_pos);
166             }
167
168             last_vertecies = 0;
169         }
170     }
171
172     void LoDSection::Push() {
173         if(aligning) {
174             forceEndAligning();
175         }
176         Mat4x4f mat = mat_stack[mat_stack.size()-1];
177
178         mat_stack.push_back(mat);
179
180         float size = size_stack[size_stack.size()-1];
181
```

```
182             size_stack.push_back(size);
183
184             RTLS_Material material  =
185                       material_stack[material_stack.size()-1];
186
187             material_stack.push_back(material);
188
189             if(last_vertecies) {
190                 int* save_verts =
191                                 new int[VERTECIES_PER_BRANCH];
192                 memcpy(save_verts, last_vertecies,
193                        sizeof(int)*VERTECIES_PER_BRANCH);
194                 last_vertecies_stack.push_back(save_verts);
195             } else {
196                 last_vertecies_stack.push_back(last_vertecies);
197             }
198         }
199
200         void LoDSection::Pop() {
201             if(aligning) {
202                 assert(0); //{[]} mismatch
203             }
204
205             mat_stack.pop_back();
206
207             size_stack.pop_back();
208
209             material_stack.pop_back();
210
211             last_vertecies =
212                           last_vertecies_stack[
213                                 last_vertecies_stack.size()-1
214                           ];
215
216             last_vertecies_stack.pop_back();
217         }
218
219         void LoDSection::Size(float w, bool absolute) {
220             if(absolute) {
221                 size_stack[size_stack.size()-1] = w;
222             } else {
223                 size_stack[size_stack.size()-1] -= w;
224             }
225         }
226
227         void LoDSection::And(float l) {
228             Mat4x4f mat = mat_stack[mat_stack.size()-1];
229
230             mat = mat *
231                rotation_Mat4x4f(CGLA::YAXIS, degree2radian(l));
232
233             mat_stack[mat_stack.size()-1] = mat;
234         }
235
236         void LoDSection::Hat(float l) {
237             Mat4x4f mat = mat_stack[mat_stack.size()-1];
238
239             mat = mat * rotation_Mat4x4f(CGLA::YAXIS, -
240                                          degree2radian(l));
241
242             mat_stack[mat_stack.size()-1] = mat;
```

```
243         }
244
245         void LoDSection::Uturn() {
246             Mat4x4f mat = mat_stack[mat_stack.size()-1];
247
248             mat = mat * rotation_Mat4x4f(CGLA::ZAXIS, M_PI);
249
250             mat_stack[mat_stack.size()-1] = mat;
251         }
252
253         void LoDSection::Minus(float l) {
254             Mat4x4f mat = mat_stack[mat_stack.size()-1];
255
256             mat = mat * rotation_Mat4x4f(CGLA::ZAXIS,
257                                          -degree2radian(l));
258
259             mat_stack[mat_stack.size()-1] = mat;
260         }
261
262         void LoDSection::Plus(float l) {
263             Mat4x4f mat = mat_stack[mat_stack.size()-1];
264
265             mat = mat * rotation_Mat4x4f(CGLA::ZAXIS,
266                                          degree2radian(l));
267
268             mat_stack[mat_stack.size()-1] = mat;
269         }
270
271         void LoDSection::Slash(float l) {
272             Mat4x4f mat = mat_stack[mat_stack.size()-1];
273
274             mat = mat * rotation_Mat4x4f(CGLA::XAXIS,
275                                          degree2radian(l));
276
277             mat_stack[mat_stack.size()-1] = mat;
278         }
279
280         void LoDSection::Backslash(float l) {
281             Mat4x4f mat = mat_stack[mat_stack.size()-1];
282
283             mat = mat * rotation_Mat4x4f(CGLA::XAXIS,
284                                          -degree2radian(l));
285
286             mat_stack[mat_stack.size()-1] = mat;
287         }
288
289         void LoDSection::PushPolygon() {
290             if(aligning) {
291                 forceEndAligning();
292             }
293             polygon_stack.push_back(
294                   Polygon(
295                      material_stack[
296                          material_stack.size()-1
297                      ]));
298         }
299
300         void LoDSection::PopPolygon() {
301             Polygon polygon = polygon_stack[
302                                         polygon_stack.size()-1];
303             polygon_stack.pop_back();
```

```
304             addPolygon(polygon);
305         }
306
307         void LoDSection::Dot() {
308             if(aligning) {
309                 forceEndAligning();
310             }
311             Mat4x4f mat = mat_stack[mat_stack.size()-1];
312             Vec4f pos = mat * Vec4f(0,0,0,1);
313             Vec3f point = Vec3f(pos[0]/pos[3],
314                                 pos[1]/pos[3],
315                                 pos[2]/pos[3]);
316             polygon_stack[
317                       polygon_stack.size()-1].addPoint(point);
318         }
319
320         void LoDSection::G(float l) {
321             if(aligning) {
322                 forceEndAligning();
323             }
324             Mat4x4f start_mat = mat_stack[mat_stack.size()-1];
325
326             if(l > 0) {
327
328                 Mat4x4f end_mat = start_mat *
329                             translation_Mat4x4f(Vec3f(l,0,0));
330
331                 mat_stack[mat_stack.size()-1] = end_mat;
332
333                 addBranch(start_mat, end_mat,
334                        material_stack[material_stack.size()-1],
335                        size_stack[size_stack.size()-1]);
336             } else {
337
338                 Vec3f h = Vec3f(start_mat[0][0],
339                                 start_mat[1][0],
340                                 start_mat[2][0]);
341
342                 addBranch(start_mat, start_mat,
343                        material_stack[material_stack.size()-1],
344                        size_stack[size_stack.size()-1], h);
345             }
346         }
347
348         void LoDSection::Color(Vec3f color, bool absolute) {
349             if(absolute) {
350                 material_stack[
351                    material_stack.size()-1]
352                      .setDiffuseColor(Vec4f(color,1));
353             } else {
354                 material_stack[material_stack.size()-1]
355                         .increaseDiffuseColor(Vec4f(color,1));
356             }
357         }
358
359         void LoDSection::DollarSign() {
360             if(aligning) {
361                 forceEndAligning();
362             }
363
364             Mat4x4f mat = mat_stack[mat_stack.size()-1];
```

```
365
366             Vec3f h = Vec3f(mat[0][0], mat[1][0], mat[2][0]);
367
368             Vec3f l = cross(dollarVector,h);
369
370             l = l / l.length();
371
372             Vec3f u = cross(h,l);
373
374             for(int i=0; i<3; i++) {
375                 mat[i][0] = h[i];
376                 mat[i][1] = l[i];
377                 mat[i][2] = u[i];
378             }
379
380             mat_stack[mat_stack.size()-1] = mat;
381         }
382
383         void LoDSection::ApplySurface(Surface* surface) {
384             if(aligning) {
385                 forceEndAligning();
386             }
387             Mat4x4f mat = mat_stack[mat_stack.size()-1];
388
389             surfaces.push_back(surface);
390             surface_mats.push_back(mat);
391         }
392
393         void LoDSection::addBranch(const Mat4x4f start_mat,
394                                    const Mat4x4f end_mat,
395                                    const RTLS_Material material,
396                                    const float width,
397                                    Vec3f direction) {
398
399             if(width < 0) {
400                 return;
401             }
402
403             Vec4f start_pos = start_mat * Vec4f(0,0,0,1);
404             Vec4f end_pos   =   end_mat * Vec4f(0,0,0,1);
405             Vec3f p;
406             Vec3f tmp, offset;
407             Vec3f dir = Vec3f(end_pos[0]/end_pos[3] –
408                               start_pos[0]/start_pos[3],
409                             end_pos[1]/end_pos[3] –
410                               start_pos[1]/start_pos[3],
411                             end_pos[2]/end_pos[3] -
412                               start_pos[2]/start_pos[3]);
413
414             float length = dir.length();
415
416             tmp = normalize(direction.length()==0?
417                             dir:direction);
418
419             if(tmp == Vec3f(1,0,0) || tmp == Vec3f(-1,0,0)) {
420                 p = cross(tmp, Vec3f(0,1,0));
421             } else {
422                 p = cross(tmp, Vec3f(1,0,0));
423             }
424             p.normalize();
425
```

```
426             p *= (width / 2.0f);
427
428             offset = Vec3f(start_pos[0]/start_pos[3],
429                            start_pos[1]/start_pos[3],
430                            start_pos[2]/start_pos[3]);
431
432             Quaternion q;
433             q.make_rot(-2.0f * M_PI /
434                            float(VERTECIES_PER_BRANCH), tmp);
435
436             int* verts = new int[VERTECIES_PER_BRANCH];
437
438             if(last_vertecies == 0) {
439                 last_vertecies = new int[VERTECIES_PER_BRANCH];
440                 for(int i=0; i<VERTECIES_PER_BRANCH; i++) {
441                     Vec3f ttt = p + offset;
442                     last_vertecies[i] =
443                             branchMesh.addVertex(p + offset,
444                                                  material,
445                                                  normalize(p));
446                     Vec3f tt = p + dir + offset;
447                     verts[i] = branchMesh.addVertex(p + dir +
448                                                     offset,
449                                                  material,
450                                                  normalize(p));
451                     p = q.apply(p);
452                 }
453             } else {
454                 for(int i=0; i<VERTECIES_PER_BRANCH; i++) {
455                     Vec3f ttt= p + dir + offset;
456                     verts[i] = branchMesh.addVertex(p + dir +
457                                                     offset,
458                                                  material,
459                                                  normalize(p));
460                     p = q.apply(p);
461                 }
462             }
463
464             for(int i=0; i<VERTECIES_PER_BRANCH; i++) {
465                 branchMesh.addFace(last_vertecies[i], verts[i],
466                    last_vertecies[(i+1)%VERTECIES_PER_BRANCH]);
467                 branchMesh.add_tFace(Vec3i(texCoord11,
468                                      texCoord10, texCoord01));
469                 branchMesh.addFace(verts[i],
470                    verts[(i+1)%VERTECIES_PER_BRANCH],
471                    last_vertecies[(i+1)%VERTECIES_PER_BRANCH]);
472                 branchMesh.add_tFace(Vec3i(texCoord10,
473                                            texCoord00,
474                                            texCoord01));
475             }
476
477             delete last_vertecies;
478
479             last_vertecies = verts;
480         }
481
482         void LoDSection::addPolygon(Polygon polygon) {
483             polygons.push_back(polygon);
484         }
485
486         void LoDSection::setMaterial(const RTLS_Material& mat){
```

```
487             material_stack[material_stack.size()-1]
488                                                 .setMaterial(mat);
489         }
490
491         void LoDSection::drawTexturified(int maxLod) {
492
493             if(mat_stack.size() == 0) return;
494
495             GLdouble* mat = (GLdouble*)
496                                     malloc(16*sizeof(GLdouble));
497
498             for(int i = 0; i<4; i++)
499                 for(int j=0; j<4; j++)
500                     mat[i*4+j] = mat_stack[0][j][i];
501
502             glPushMatrix();
503
504             glMultMatrixd(mat);
505
506             for(int t=0; t<2; t++) {
507
508                 if(current_texture[t] != NULL) {
509                     glBindTexture(GL_TEXTURE_2D,
510                                   *current_texture[t]);
511                     glEnable(GL_TEXTURE_2D);
512                     glTexEnvf(GL_TEXTURE_ENV,
513                               GL_TEXTURE_ENV_MODE, GL_REPLACE);
514
515                     glBegin(GL_QUADS);
516                     glTexCoord2f(0.0, 0.0);
517                     glVertex4fv( pos[t][0].get() );
518                     glTexCoord2f(0.0, 1.0);
519                     glVertex4fv( pos[t][1].get() );
520                     glTexCoord2f(1.0, 1.0);
521                     glVertex4fv( pos[t][2].get() );
522                     glTexCoord2f(1.0, 0.0);
523                     glVertex4fv( pos[t][3].get() );
524                     glEnd();
525
526                     glDisable(GL_TEXTURE_2D);
527                 }
528             }
529
530             if(my_lvl < maxLod && subObjects.size() > 0) {
531                 for(vector<LoDSection*>::iterator iter =
532                     subObjects.begin();
533                     iter != subObjects.end();
534                     iter++) {
535                         (*iter)->drawTexturified(maxLod);
536                 }
537             }
538
539             glPopMatrix();
540
541             free(mat);
542         }
543
544         bool LoDSection::createTexture(int max_depth) {
545
546     #ifdef _DEBUG
547             printState();
```

```
548   #endif
549           switch(state) {
550               case IDLE:
551                   state = GETTING_BBOX_INIT;
552                   for(int i=0; i<16; i++) {
553                       timing[i].HighPart = 0;
554                       timing[i].LowPart  = 0;
555                       timing[i].QuadPart = 0;
556                   }
557                   calculateBBox(max_depth);
558                   break;
559               case GETTING_BBOX_INIT:
560               case GETTING_BBOX_BRANCHES:
561               case GETTING_BBOX_POLYGONS:
562               case GETTING_BBOX_SURFACES:
563               case GETTING_BBOX_SUB_INIT:
564               case GETTING_BBOX_SUB:
565               case BBOX_DONE:
566                   calculateBBox(max_depth);
567                   break;
568               case RENDERING_INIT:
569               case RENDERING_BRANCHES:
570               case RENDERING_POLYGONS:
571               case RENDERING_SURFACES:
572               case RENDERING_SUB:
573               case READ_TEXTURE:
574               case EDIT_TEXTURE:
575               case CREATE_TEXTURE:
576                   renderNewTextures(max_depth);
577                   break;
578           }
579
580           return state == IDLE;
581
582       }
583
584       void LoDSection::renderNewTextures(int max_depth) {
585
586   #ifdef _TIMING
587           startTimer();
588   #endif
589           switch(state) {
590               case RENDERING_INIT:
591
592                   glViewport(0, 0,
593                              TEXTURE_SIZE, TEXTURE_SIZE);
594
595                   glEnable(GL_LIGHTING);
596                   glEnable(GL_LIGHT0);
597
598                   glMatrixMode(GL_PROJECTION);
599
600                   glLoadIdentity();
601
602                   if(t==0) glOrtho(x, x+xLength,
603                                    y, y+yLength,
604                                    z, z+zLength+0.1 );
605                   else     glOrtho(x, x+xLength,
606                                    z, z+zLength,
607                                    y, y+yLength+0.1 );
608
```

```
609                 glMatrixMode(GL_MODELVIEW);
610
611                 glLoadIdentity();
612
613                 glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
614
615                 glClear(GL_COLOR_BUFFER_BIT |
616                         GL_DEPTH_BUFFER_BIT);
617
618                 if(t==0) {
619                     glScalef(1.0f,1.0f,-1.0f);
620                 } else {
621                     glRotatef(-90.0f,1.0f,0.0f,0.0f);
622                 }
623
624                 glLightfv(GL_LIGHT0, GL_POSITION,
625                           light_position.get());
626
627                 changeState(RENDERING_BRANCHES);
628
629                 break;
630             case RENDERING_BRANCHES: {
631
632                 bool done = false;
633
634                 if(barktex) {
635                     glTexEnvi(GL_TEXTURE_ENV,
636                               GL_TEXTURE_ENV_MODE,
637                               GL_MODULATE);
638                     glEnable(GL_TEXTURE_2D);
639                     glBindTexture(GL_TEXTURE_2D, barktex);
640                 }
641
642                 while(!done &&
643                       !timer->isRenderTimeExceeded()) {
644
645                          done = branchMesh.drawSome(10);
646                 }
647
648                 if(barktex) { glDisable(GL_TEXTURE_2D); }
649
650                 if(done) {
651                     changeState(RENDERING_POLYGONS);
652                     counter = 0;
653                 }
654
655                 break;
656                                     }
657             case RENDERING_POLYGONS:
658
659                 while(counter < polygons.size() &&
660                       !timer->isRenderTimeExceeded()) {
661
662                     polygons[counter].draw();
663                     counter++;
664                 }
665
666                 if(counter == polygons.size()) {
667
668                     iter = subObjects.begin();
669                     changeState(RENDERING_SURFACES);
```

```
670                     counter = 0;
671                 }
672
673                 break;
674             case RENDERING_SURFACES: {
675
676                 bool done = (surfaces.size() == 0);
677
678                 while(!done &&
679                       !timer->isRenderTimeExceeded()) {
680
681                     GLdouble* mat = (GLdouble*)
682                                 malloc(16*sizeof(GLdouble));
683
684                     for(int i = 0; i<4; i++)
685                         for(int j=0; j<4; j++)
686                             mat[i*4+j] =
687                                 surface_mats[counter][j][i];
688
689                     glPushMatrix();
690
691                     glMultMatrixd(mat);
692
693                     surfaces[counter]->draw();
694
695                     glPopMatrix();
696
697                     free(mat);
698
699                     counter++;
700                     done = counter >= surfaces.size();
701                 }
702
703                 if(done) {
704                     changeState(RENDERING_SUB);
705                 }
706
707                 break;
708                                     }
709             case RENDERING_SUB:
710                 {
711                     bool done = false;
712
713                     if(my_lvl >= max_depth) {
714                         while(iter != subObjects.end()) {
715                             done = (*iter)->drawRelative();
716                             if(done) {
717                                 iter++;
718                             } else {
719 #ifdef _TIMING
720                                 endTimer();
721 #endif
722                                 return;
723                             }
724                            if(
725                             timer->isRenderTimeExceeded()){
726 #ifdef _TIMING
727                                 endTimer();
728 #endif
729                                 return;
730                             }
```

```
731                         }
732                     }
733
734                     glDisable(GL_LIGHTING);
735                     glDisable(GL_LIGHT0);
736
737                     changeState(READ_TEXTURE);
738
739                     break;
740                 }
741             case READ_TEXTURE:
742
743                 glReadPixels(0, 0, TEXTURE_SIZE,
744                              TEXTURE_SIZE, GL_RGBA,
745                              GL_UNSIGNED_BYTE,
746                              tmp_texture);
747                 xt = 0;
748                 yt = 0;
749
750                 changeState(EDIT_TEXTURE);
751
752                 break;
753             case EDIT_TEXTURE: {
754
755                 int neighbours;
756                 while(xt < TEXTURE_SIZE) {
757                     while(yt < TEXTURE_SIZE) {
758                         neighbours = next2border(xt,yt);
759                         if(neighbours) {
760                             applyFilter(xt,yt,neighbours);
761                         }
762                         yt++;
763                         if(timer->isRenderTimeExceeded()) {
764 #ifdef _TIMING
765                             endTimer();
766 #endif
767                             return;
768                         }
769                     }
770                     yt = 0;
771                     xt++;
772                 }
773
774                 changeState(CREATE_TEXTURE);
775
776                 break;
777                                 }
778             case CREATE_TEXTURE:
779
780                 if(first2nextTextures) {
781                     glBindTexture(GL_TEXTURE_2D,
782                                   *next_texture[t]);
783                 } else {
784                     glBindTexture(GL_TEXTURE_2D,
785                                   *next_texture[2+t]);
786                 }
787                 glTexParameteri(GL_TEXTURE_2D,
788                                 GL_TEXTURE_WRAP_S,
789                                 GL_CLAMP);
790                 glTexParameteri(GL_TEXTURE_2D,
791                                 GL_TEXTURE_WRAP_T,
```

```
792                                     GL_CLAMP);
793                 glTexParameteri(GL_TEXTURE_2D,
794                                     GL_TEXTURE_MAG_FILTER,
795                                     GL_LINEAR);
796                 glTexParameteri(GL_TEXTURE_2D,
797                                     GL_TEXTURE_MIN_FILTER,
798                                     GL_LINEAR);
799
800                 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
801                              TEXTURE_SIZE, TEXTURE_SIZE, 0,
802                              GL_RGBA, GL_UNSIGNED_BYTE,
803                              tmp_texture);
804
805                 t++;
806
807                 if(t<2) {
808                     changeState(RENDERING_INIT);
809                 } else {
810                     changeState(IDLE);
811                 }
812
813                 break;
814             }
815     #ifdef _TIMING
816             endTimer();
817     #endif
818         }
819
820         void LoDSection::calculateBBox(int max_depth) {
821
822     #ifdef _TIMING
823             startTimer();
824     #endif
825
826             if(state == GETTING_BBOX_INIT ||
827                state == GETTING_BBOX_SUB_INIT ||
828                state == GETTING_BBOX_SUB ||
829                state == GETTING_BBOX_BRANCHES ||
830                state == GETTING_BBOX_POLYGONS ||
831                state == GETTING_BBOX_SURFACES ) {
832
833                     if(state == GETTING_BBOX_SUB_INIT ||
834                        state == GETTING_BBOX_SUB ||
835                        my_lvl >= max_depth &&
836                          (state == GETTING_BBOX_INIT ||
837                           state == GETTING_BBOX_BRANCHES ||
838                           state == GETTING_BBOX_POLYGONS ||
839                           state == GETTING_BBOX_SURFACES )) {
840
841                             if(state == GETTING_BBOX_INIT ||
842                                state == GETTING_BBOX_SUB_INIT){
843                                 res = new Vec3f[8];
844                             }
845
846                             bool done =
847                                 getBBoxIncludingSubLoDs(res,
848                                                         false);
849
850                             if(!done) {
851     #ifdef _TIMING
852                                 endTimer();
```

```
853   #endif
854                                 return;
855                             }
856
857                             lower_left = res[0];
858                             upper_right = res[0];
859
860                             for(int i=1; i<8; i++) {
861                                 for(int j=0; j<3; j++) {
862                                     if(lower_left[j] >
863                                        res[i][j]) {
864                                      lower_left[j] = res[i][j];
865                                     }
866                                     if(upper_right[j] <
867                                        res[i][j]) {
868                                             upper_right[j] =
869                                                   res[i][j];
870                                     }
871                                 }
872                             }
873
874                             delete res;
875
876                         } else {
877                             if(state == GETTING_BBOX_INIT) {
878
879                             changeState(GETTING_BBOX_BRANCHES);
880                             }
881                             if(state == GETTING_BBOX_BRANCHES){
882                                branchMesh.getBBox(lower_left,
883                                                   upper_right);
884
885
886                             changeState(GETTING_BBOX_POLYGONS);
887                                 counter = 0;
888                             }
889                             if(state == GETTING_BBOX_POLYGONS){
890                                if(!getPolygonsBBox(lower_left,
891                                                    upper_right)){
892   #ifdef _TIMING
893                                     endTimer();
894   #endif
895                                     return;
896                                 }
897
898
899                             changeState(GETTING_BBOX_SURFACES);
900                                 counter = 0;
901                             }
902                             if(state == GETTING_BBOX_SURFACES){
903                                 if(!getSurfacesBBox(lower_left,
904                                                    upper_right)){
905   #ifdef _TIMING
906                                     endTimer();
907   #endif
908                                     return;
909                                 }
910                             }
911                         }
912                         changeState(BBOX_DONE);
913
```

```
914                         if(timer->isRenderTimeExceeded()) {
915   #ifdef _TIMING
916                             endTimer();
917   #endif
918                             return;
919                         }
920                 }
921
922                 if(state == BBOX_DONE) {
923
924                     xLength = upper_right[0] - lower_left[0];
925                     yLength = upper_right[1] - lower_left[1];
926                     zLength = upper_right[2] - lower_left[2];
927
928                     assert(xLength >= 0 &&
929                            yLength >= 0 &&
930                            zLength >= 0);
931
932                     x = lower_left[0];
933                     y = lower_left[1];
934                     z = lower_left[2];
935
936                     changeState(RENDERING_INIT);
937                     t = 0;
938                 }
939         }
940
941         bool LoDSection::getSurfacesBBox(Vec3f& min_crnr,
942                                          Vec3f& max_crnr) {
943             Vec3f t0, t1;
944             Vec3f* p = new Vec3f[8];
945             Vec4f tmp;
946             float temp;
947             while(counter<surfaces.size()) {
948                 surfaces[counter]->getBBox(t0, t1);
949
950                 p[0] = Vec3f(t0[0], t0[1], t0[2]);
951                 p[1] = Vec3f(t0[0], t0[1], t1[2]);
952                 p[2] = Vec3f(t0[0], t1[1], t0[2]);
953                 p[3] = Vec3f(t0[0], t1[1], t1[2]);
954                 p[4] = Vec3f(t1[0], t0[1], t0[2]);
955                 p[5] = Vec3f(t1[0], t0[1], t1[2]);
956                 p[6] = Vec3f(t1[0], t1[1], t0[2]);
957                 p[7] = Vec3f(t1[0], t1[1], t1[2]);
958
959                 for(int i=0; i<8; i++) {
960                     tmp = surface_mats[counter] *
961                                               Vec4f(p[i], 1.0f);
962                     for(int j=0; j<3; j++) {
963                         temp = tmp[j] / tmp[3];
964                         if(temp < min_crnr[j]) {
965                                               min_crnr[j] = temp; }
966                         if(temp > max_crnr[j]) {
967                                               max_crnr[j] = temp; }
968                     }
969                 }
970                 counter++;
971                 if(timer->isRenderTimeExceeded()) {
972                     delete p;
973                     return false;
974                 }
```

```
975             }
976             delete p;
977             return true;
978         }
979
980         bool LoDSection::getPolygonsBBox(Vec3f& min_crnr,
981                                          Vec3f& max_crnr) {
982             Vec3f t0, t1;
983
984             while(counter<polygons.size()) {
985                 polygons[counter].getBBox(t0, t1);
986
987                 for(int j=0; j<3; j++) {
988                     if(t0[j] < min_crnr[j]) {
989                                          min_crnr[j] = t0[j]; }
990                     if(t1[j] < min_crnr[j]) {
991                                          min_crnr[j] = t1[j]; }
992                     if(t1[j] > max_crnr[j]) {
993                                          max_crnr[j] = t1[j]; }
994                     if(t0[j] > max_crnr[j]) {
995                                          max_crnr[j] = t0[j]; }
996                 }
997                 counter++;
998                 if(timer->isRenderTimeExceeded()) {
999                     return false;
1000                }
1001            }
1002            return true;
1003        }
1004
1005        bool LoDSection::getBBoxIncludingSubLoDs(Vec3f* p,
1006                        bool translate_to_parent_coordinates) {
1007
1008    #ifdef _TIMING
1009            changeState(state);
1010    #endif
1011
1012            if(state == GETTING_BBOX_INIT || state == IDLE) {
1013                bb = new Vec3f[8];
1014                v0 = Vec3f(0);
1015                v1 = Vec3f(0);
1016
1017                changeState(GETTING_BBOX_BRANCHES);
1018                if(timer->isRenderTimeExceeded()) {
1019    #ifdef _TIMING
1020                    endTimer();
1021    #endif
1022                    return false;
1023                }
1024            }
1025
1026            if(state == GETTING_BBOX_BRANCHES) {
1027                branchMesh.getBBox(v0, v1);
1028
1029                changeState(GETTING_BBOX_POLYGONS);
1030
1031                counter = 0;
1032                if(timer->isRenderTimeExceeded()) {
1033    #ifdef _TIMING
1034                    endTimer();
1035    #endif
```

```
1036                    return false;
1037                }
1038            }
1039
1040            if(state == GETTING_BBOX_POLYGONS) {
1041                if(!getPolygonsBBox(v0,v1)) {
1042    #ifdef _TIMING
1043                    endTimer();
1044    #endif
1045                    return false;
1046                }
1047
1048                changeState(GETTING_BBOX_SURFACES);
1049                counter = 0;
1050            }
1051
1052            if(state == GETTING_BBOX_SURFACES) {
1053                if(!getSurfacesBBox(v0, v1)) {
1054    #ifdef _TIMING
1055                    endTimer();
1056    #endif
1057                    return false;
1058                }
1059                changeState(GETTING_BBOX_SUB_INIT);
1060            }
1061
1062            if(state == GETTING_BBOX_SUB_INIT ||
1063               state == GETTING_BBOX_SUB) {
1064                if(subObjects.size() > 0) {
1065                    if(state == GETTING_BBOX_SUB_INIT) {
1066                        iter = subObjects.begin();
1067
1068                        changeState(GETTING_BBOX_SUB);
1069
1070                        if(timer->isRenderTimeExceeded()) {
1071    #ifdef _TIMING
1072                            endTimer();
1073    #endif
1074                            return false;
1075                        }
1076                    }
1077                    bool done;
1078                    while(iter != subObjects.end()) {
1079                        done =
1080                           (*iter)->getBBoxIncludingSubLoDs(bb,
1081                                                            true);
1082                        if(done) {
1083                            for(int i=0; i<8; i++) {
1084                                for(int j=0; j<3; j++) {
1085                                    if(bb[i][j] < v0[j])
1086                                            v0[j] = bb[i][j];
1087                                    if(bb[i][j] > v1[j])
1088                                            v1[j] = bb[i][j];
1089                                }
1090                            }
1091                            iter++;
1092                        } else {
1093    #ifdef _TIMING
1094                            endTimer();
1095    #endif
1096                            return false;
```

```
1097                        }
1098                        if(timer->isRenderTimeExceeded()) {
1099    #ifdef _TIMING
1100                            endTimer();
1101    #endif
1102                            return false;
1103                        }
1104                    }
1105                }
1106            }
1107
1108            p[0] = Vec3f(v0[0], v0[1], v0[2]);
1109            p[1] = Vec3f(v0[0], v0[1], v1[2]);
1110            p[2] = Vec3f(v0[0], v1[1], v0[2]);
1111            p[3] = Vec3f(v0[0], v1[1], v1[2]);
1112            p[4] = Vec3f(v1[0], v0[1], v0[2]);
1113            p[5] = Vec3f(v1[0], v0[1], v1[2]);
1114            p[6] = Vec3f(v1[0], v1[1], v0[2]);
1115            p[7] = Vec3f(v1[0], v1[1], v1[2]);
1116
1117            if(translate_to_parent_coordinates) {
1118                Vec4f tmp;
1119                for(int i = 0; i<8; i++) {
1120                    tmp = mat_stack[0] * Vec4f(p[i], 1);
1121                    p[i] = Vec3f(tmp[0]/tmp[3],
1122                                 tmp[1]/tmp[3],
1123                                 tmp[2]/tmp[3]);
1124                }
1125                changeState(IDLE);
1126            } else {
1127                changeState(BBOX_DONE);
1128            }
1129
1130            delete bb;
1131
1132            return true;
1133        }
1134
1135        void LoDSection::swapTexture() {
1136            for(int i=0; i<2; i++) {
1137                if(first2nextTextures) {
1138                    current_texture[i] = next_texture[i];
1139                } else {
1140                    current_texture[i] = next_texture[2+i];
1141                }
1142            }
1143
1144            first2nextTextures = !first2nextTextures;
1145
1146            pos[0][0] = Vec4f(x        , y        , 0, 1);
1147            pos[0][1] = Vec4f(x        , y+yLength, 0, 1);
1148            pos[0][2] = Vec4f(x+xLength, y+yLength, 0, 1);
1149            pos[0][3] = Vec4f(x+xLength, y        , 0, 1);
1150
1151            pos[1][0] = Vec4f(x        , 0, z        , 1);
1152            pos[1][1] = Vec4f(x        , 0, z+zLength, 1);
1153            pos[1][2] = Vec4f(x+xLength, 0, z+zLength, 1);
1154            pos[1][3] = Vec4f(x+xLength, 0, z        , 1);
1155        }
1156
1157        bool LoDSection::drawRelative() {
```

```
1158
1159    #ifdef _DEBUG
1160            printState();
1161    #endif
1162    #ifdef _TIMING
1163            changeState(state);
1164    #endif
1165            switch(state) {
1166                case IDLE:
1167                    changeState(RENDERING_INIT);
1168                case RENDERING_INIT:
1169                    {
1170                        GLdouble* mat = (GLdouble*)
1171                                    malloc(16*sizeof(GLdouble));
1172
1173                        for(int i = 0; i<4; i++)
1174                            for(int j=0; j<4; j++)
1175                                mat[i*4+j] =
1176                                            mat_stack[0][j][i];
1177
1178                        glPushMatrix();
1179
1180                        glMultMatrixd(mat);
1181
1182                        free(mat);
1183
1184                        changeState(RENDERING_BRANCHES);
1185                        break;
1186                    }
1187
1188                case RENDERING_BRANCHES:
1189                    {
1190                        bool done = false;
1191
1192                        if(barktex) {
1193                            glTexEnvi(GL_TEXTURE_ENV,
1194                                      GL_TEXTURE_ENV_MODE,
1195                                      GL_MODULATE);
1196                            glEnable(GL_TEXTURE_2D);
1197                            glBindTexture(GL_TEXTURE_2D,
1198                                          barktex);
1199                        }
1200
1201                        while(!done &&
1202                              !timer->isRenderTimeExceeded()) {
1203                            done = branchMesh.drawSome(10);
1204                        }
1205
1206                        if(barktex) {
1207                            glDisable(GL_TEXTURE_2D);
1208                        }
1209
1210                        if(done) {
1211                            changeState(RENDERING_POLYGONS);
1212                        }
1213                        break;
1214                    }
1215                case RENDERING_POLYGONS:
1216
1217                    while(counter < polygons.size() &&
1218                          !timer->isRenderTimeExceeded()) {
```

```
1219                        polygons[counter++].draw();
1220                    }
1221
1222                    if(counter >= polygons.size()) {
1223                        iter = subObjects.begin();
1224                        changeState(RENDERING_SURFACES);
1225                        counter = 0;
1226                    }
1227
1228                    break;
1229                case RENDERING_SURFACES: {
1230
1231                    bool done = (surfaces.size() == 0);
1232
1233                    while(!done &&
1234                          !timer->isRenderTimeExceeded()) {
1235                        GLdouble* mat = (GLdouble*)
1236                                    malloc(16*sizeof(GLdouble));
1237
1238                        for(int i = 0; i<4; i++)
1239                            for(int j=0; j<4; j++)
1240                                mat[i*4+j] =
1241                                    surface_mats[counter][j][i];
1242
1243                        glPushMatrix();
1244
1245                        glMultMatrixd(mat);
1246
1247                        surfaces[counter++]->draw();
1248
1249                        glPopMatrix();
1250
1251                        free(mat);
1252
1253                        done = counter >= surfaces.size();
1254                    }
1255
1256                    if(done) {
1257                        changeState(RENDERING_SUB);
1258                    }
1259
1260                    break;
1261                                        }
1262
1263                case RENDERING_SUB:
1264                    {
1265                        bool done = false;
1266
1267                        while(iter != subObjects.end()) {
1268                            done = (*iter)->drawRelative();
1269                            if(done) {
1270                                iter++;
1271                            } else {
1272   #ifdef _TIMING
1273                                endTimer();
1274   #endif
1275                                return false;
1276                            }
1277                            if(timer->isRenderTimeExceeded()) {
1278   #ifdef _TIMING
1279                                endTimer();
```

```
#endif
                                return false;
                            }
                        }

                        changeState(IDLE);

                        glPopMatrix(); //pop matrix from
                                       //rendering init

                        break;
                    }
            }
#ifdef _TIMING
            endTimer();
#endif

            return state == IDLE;
        }

        const float LoDSection::getCurrentSize() const {
            return size_stack[size_stack.size()-1];
        }

        const Mat4x4f LoDSection::getCurrentMatrix() const {
            return mat_stack[mat_stack.size()-1];
        }

        const RTLS_Material LoDSection::getCurrentMaterial()
                                                        const {
            return material_stack[material_stack.size()-1];
        }

        void LoDSection::addSubLoD(LoDSection* obj) {
            subObjects.push_back(obj);
        }

        void LoDSection::justDraw() {

            GLdouble* mat = (GLdouble*)
                                malloc(16*sizeof(GLdouble));

            for(int i = 0; i<4; i++)
                for(int j=0; j<4; j++)
                    mat[i*4+j] = mat_stack[0][j][i];

            glPushMatrix();

            glMultMatrixd(mat);

            if(barktex) {
                glTexEnvi(GL_TEXTURE_ENV,
                          GL_TEXTURE_ENV_MODE,
                          GL_MODULATE);
                glEnable(GL_TEXTURE_2D);
                glBindTexture(GL_TEXTURE_2D, barktex);
            }

            branchMesh.drawAll();

            if(barktex) { glDisable(GL_TEXTURE_2D); }
```

```
1341
1342            for(int i=0;i<polygons.size();i++) {
1343                polygons[i].draw();
1344            }
1345
1346            for(int k=0; k<surfaces.size(); k++) {
1347
1348                for(int i = 0; i<4; i++)
1349                    for(int j=0; j<4; j++)
1350                        mat[i*4+j] = surface_mats[k][j][i];
1351
1352                glPushMatrix();
1353
1354                glMultMatrixd(mat);
1355
1356                surfaces[k]->draw();
1357
1358                glPopMatrix();
1359            }
1360
1361            for(vector<LoDSection*>::iterator iter =
1362                                subObjects.begin();
1363                iter != subObjects.end(); iter++) {
1364                  (*iter)->justDraw();
1365            }
1366
1367            glPopMatrix();
1368
1369            free(mat);
1370        }
1371
1372        void LoDSection::drawBBox(int max_depth) {
1373            GLdouble* mat = (GLdouble*)
1374                                malloc(16*sizeof(GLdouble));
1375
1376            for(int i = 0; i<4; i++)
1377                for(int j=0; j<4; j++)
1378                    mat[i*4+j] = mat_stack[0][j][i];
1379
1380            glPushMatrix();
1381
1382            glMultMatrixd(mat);
1383
1384            glColor3f(0.0f,0.0f,0.0f);
1385            glLineWidth(1.0f);
1386
1387            glBegin(GL_LINES);
1388            glVertex3f(lower_left[0],
1389                       lower_left[1],
1390                       lower_left[2]);
1391            glVertex3f(upper_right[0],
1392                       lower_left[1],
1393                       lower_left[2]);
1394            glVertex3f(lower_left[0],
1395                       lower_left[1],
1396                       lower_left[2]);
1397            glVertex3f(lower_left[0],
1398                      upper_right[1],
1399                       lower_left[2]);
1400            glVertex3f(lower_left[0],
1401                       lower_left[1],
```

```
1402                    lower_left[2]);
1403            glVertex3f(lower_left[0],
1404                    lower_left[1],
1405                    upper_right[2]);
1406
1407            glVertex3f(upper_right[0],
1408                    upper_right[1],
1409                    upper_right[2]);
1410            glVertex3f(lower_left[0],
1411                    upper_right[1],
1412                    upper_right[2]);
1413            glVertex3f(upper_right[0],
1414                    upper_right[1],
1415                    upper_right[2]);
1416            glVertex3f(upper_right[0],
1417                    lower_left[1],
1418                    upper_right[2]);
1419            glVertex3f(upper_right[0],
1420                    upper_right[1],
1421                    upper_right[2]);
1422            glVertex3f(upper_right[0],
1423                    upper_right[1],
1424                    lower_left[2]);
1425
1426            glVertex3f(upper_right[0],
1427                    lower_left[1],
1428                    lower_left[2]);
1429            glVertex3f(upper_right[0],
1430                    lower_left[1],
1431                    upper_right[2]);
1432            glVertex3f(upper_right[0],
1433                    lower_left[1],
1434                    lower_left[2]);
1435            glVertex3f(upper_right[0],
1436                    upper_right[1],
1437                    lower_left[2]);
1438
1439            glVertex3f(lower_left[0],
1440                    upper_right[1],
1441                    lower_left[2]);
1442            glVertex3f(lower_left[0],
1443                    upper_right[1],
1444                    upper_right[2]);
1445            glVertex3f(lower_left[0],
1446                    upper_right[1],
1447                    lower_left[2]);
1448            glVertex3f(upper_right[0],
1449                    upper_right[1],
1450                    lower_left[2]);
1451
1452            glVertex3f(lower_left[0],
1453                    lower_left[1],
1454                    upper_right[2]);
1455            glVertex3f(lower_left[0],
1456                    upper_right[1],
1457                    upper_right[2]);
1458            glVertex3f(lower_left[0],
1459                    lower_left[1],
1460                    upper_right[2]);
1461            glVertex3f(upper_right[0],
1462                    lower_left[1],
```

```
1463                    upper_right[2]);
1464
1465            glEnd();
1466
1467            if(my_lvl < max_depth) {
1468                for(vector<LoDSection*>::iterator iter =
1469                            subObjects.begin();
1470                    iter != subObjects.end(); iter++) {
1471                        (*iter)->drawBBox(max_depth);
1472                }
1473            }
1474
1475            glPopMatrix();
1476
1477            free(mat);
1478        }
1479
1480        void LoDSection::printState() const {
1481            switch(state) {
1482                case IDLE:
1483                    cout << "IDLE" << endl;
1484                    break;
1485                case GETTING_BBOX_INIT:
1486                    cout << "GETTING_BBOX_INIT" << endl;
1487                    break;
1488                case GETTING_BBOX_BRANCHES:
1489                    cout << "GETTING_BBOX_BRANCHES" << endl;
1490                    break;
1491                case GETTING_BBOX_POLYGONS:
1492                    cout << "GETTING_BBOX_POLYGONS" << endl;
1493                    break;
1494                case GETTING_BBOX_SURFACES:
1495                    cout << "GETTING_BBOX_SURFACES" << endl;
1496                    break;
1497                case GETTING_BBOX_SUB_INIT:
1498                    cout << "GETTING_BBOX_SUB_INIT" << endl;
1499                    break;
1500                case GETTING_BBOX_SUB:
1501                    cout << "GETTING_BBOX_SUB" << endl;
1502                    break;
1503                case BBOX_DONE:
1504                    cout << "BBOX_DONE" << endl;
1505                    break;
1506                case RENDERING_INIT:
1507                    cout << "RENDERING_INIT" << endl;
1508                    break;
1509                case RENDERING_BRANCHES:
1510                    cout << "RENDERING_BRANCHES" << endl;
1511                    break;
1512                case RENDERING_POLYGONS:
1513                    cout << "RENDERING_POLYGONS" << endl;
1514                    break;
1515                case RENDERING_SURFACES:
1516                    cout << "RENDERING_SURFACES" << endl;
1517                    break;
1518                case RENDERING_SUB:
1519                    cout << "RENDERING_SUB" << endl;
1520                    break;
1521                case READ_TEXTURE:
1522                    cout << "READ_TEXTURE" << endl;
1523                    break;
```

```
1524                case EDIT_TEXTURE:
1525                    cout << "EDIT_TEXTURE" << endl;
1526                    break;
1527                case CREATE_TEXTURE:
1528                    cout << "CREATE_TEXTURE" << endl;
1529                    break;
1530            }
1531        }
1532
1533        int LoDSection::getNumberOfPolygons() {
1534            int res = branchMesh.getNumberOfPolygons();
1535            res += polygons.size();
1536            for(int i=0; i<surfaces.size(); i++) {
1537                Surface* surf = surfaces[i];
1538                res += surf->getNumberOfPolygons();
1539            }
1540            return res;
1541        }
1542
1543        void LoDSection::startTimer() {
1544    #ifdef _TIMING
1545            QueryPerformanceCounter(&start_clk);
1546    #endif
1547        }
1548
1549        void LoDSection::endTimer() {
1550    #ifdef _TIMING
1551            QueryPerformanceCounter(&end_clk);
1552
1553            timing[state].QuadPart +=
1554                    (end_clk.QuadPart - start_clk.QuadPart);
1555    #endif
1556        }
1557
1558        LARGE_INTEGER* LoDSection::getTimings() {
1559            return timing;
1560        }
1561
1562        char* LoDSection::getStateLabel(int state) {
1563            switch(state) {
1564                case IDLE:
1565                    return "IDLE";
1566                case GETTING_BBOX_INIT:
1567                    return "GETTING BBOX INIT";
1568                case GETTING_BBOX_BRANCHES:
1569                    return "GETTING BBOX BRANCHES";
1570                case GETTING_BBOX_POLYGONS:
1571                    return "GETTING BBOX POLYGONS";
1572                case GETTING_BBOX_SURFACES:
1573                    return "GETTING BBOX SURFACES";
1574                case GETTING_BBOX_SUB_INIT:
1575                    return "GETTING BBOX SUB INIT";
1576                case GETTING_BBOX_SUB:
1577                    return "GETTING BBOX SUB";
1578                case BBOX_DONE:
1579                    return "BBOX DONE";
1580                case RENDERING_INIT:
1581                    return "RENDERING INIT";
1582                case RENDERING_BRANCHES:
1583                    return "RENDERING BRANCHES";
1584                case RENDERING_POLYGONS:
```

```
1585                    return "RENDERING POLYGONS";
1586                case RENDERING_SURFACES:
1587                    return "RENDERING SURFACES";
1588                case RENDERING_SUB:
1589                    return "RENDERING SUB";
1590                case READ_TEXTURE:
1591                    return "READ TEXTURE";
1592                case EDIT_TEXTURE:
1593                    return "EDIT TEXTURE";
1594                case CREATE_TEXTURE:
1595                    return "CREATE TEXTURE";
1596            }
1597            return "UNKNOWN STATE";
1598        }
1599    }
```

8 Polygon.h

```
1   #ifndef     _POLYGON_H_
2   #define     _POLYGON_H_
3
4   #include <vector>
5   #include <GL/glew.h>
6
7   #include "CGLA/Vec3f.h"
8   #include "CGLA/Vec4f.h"
9
10  #include "Common.h"
11  #include "RTLS_Material.h"
12
13  namespace RealTimeLSystem {
14
15      class Polygon {
16      public:
17          Polygon(RTLS::RTLS_Material _material);
18          ~Polygon(void);
19
20          void addPoint(CGLA::Vec3f point);
21          void addPoint(CGLA::Vec4f point);
22
23          void getBBox(CGLA::Vec3f& p0, CGLA::Vec3f& p1);
24
25          void draw()     const;
26      private:
27          RTLS::RTLS_Material material;
28          std::vector<CGLA::Vec3f> points;
29          CGLA::Vec3f     bb0, bb1;
30      };
31  }
32  #endif
```

9 Polygon.cpp

```
1   #include "Polygon.h"
2
3   using namespace CGLA;
4   using namespace std;
5
6   namespace RealTimeLSystem {
7
8       Polygon::Polygon(RTLS_Material _material) :
9                                           material(_material) {
10
11      }
12
13      Polygon::~Polygon(void) {
14          points.clear();
15      }
16
17      void Polygon::addPoint(Vec3f point) {
18          if(points.size() == 0) {
19              bb0 = point;
20              bb1 = point;
21          } else {
22
23              for(int j=0; j<3; j++) {
24                  if(point[j] < bb0[j]) {
25                      bb0[j] = point[j];
26                  }
27                  if(point[j] > bb1[j]) {
28                      bb1[j] = point[j];
29                  }
30              }
31          }
32          points.push_back(point);
33      }
34
35      void Polygon::addPoint(Vec4f point) {
36          addPoint(Vec3f(point[0]/point[3],
37                         point[1]/point[3],
38                         point[2]/point[3]));
39      }
40
41      void Polygon::draw() const {
42          Vec3f normal, pre_normal = Vec3f(0,0,0);
43          int size = points.size();
44
45          glPushAttrib(GL_ALL_ATTRIB_BITS);
46
47          glMaterialfv(GL_FRONT, GL_AMBIENT,
48                       material.getAmbientColor().get());
49          glMaterialfv(GL_FRONT, GL_DIFFUSE,
50                       material.getDiffuseColor().get());
51          glMaterialfv(GL_FRONT, GL_SPECULAR,
52                       material.getSpecularColor().get());
53          glMaterialf(GL_FRONT, GL_SHININESS,
54                      material.getShininess());
55
56          glBegin(GL_POLYGON);
57          for(int i=0; i<size; i++) {
58              normal = cross(points[(i < size-1 ? i+1 : 0)] –
59                             points[i],
```

```
60                                  points[(i > 0 ? i-1 : size-1)] –
61                                    points[i]);
62                  if(normal.length() < 0.00001) {
63                      if(pre_normal.length() == 0) {
64                          int j=2;
65                          while(normal.length() < 0.00001 &&
66                                j < size) {
67                              normal = cross(points[(i+j)%
68                                             (size-1)] –
69                                             points[i],
70                                             points[(i > 0 ?
71                                                    i-1 : size-1)]
72                                             - points[i]);
73                              j++;
74                          }
75                          if(normal.length() == 0) {
76                              normal = Vec3f(1,0,0);
77                          }
78                          normal = normalize(normal);
79                          pre_normal = normal;
80                      } else {
81                          normal = pre_normal;
82                      }
83                  } else {
84                      normal = normalize(normal);
85                      pre_normal = normal;
86                  }
87                  glNormal3fv(normal.get());
88                  glVertex3fv(points[i].get());
89              }
90              glEnd();
91
92              glPopAttrib();
93          }
94
95          void Polygon::getBBox(Vec3f& p0, Vec3f& p1) {
96
97              p0 = bb0;
98              p1 = bb1;
99          }
100     }
```

10 Rule.cpp

```
1   #ifndef     _RULE_H_
2   #define     _RULE_H_
3
4   #include <vector>
5
6   #include "Common.h"
7   #include "Element.h"
8
9   namespace RealTimeLSystem {
10
11      class Rule {
12      public:
13          virtual bool apply(Element* element_in,
14                     std::vector<Element*> *lsystem_out) = 0;
15      };
16  }
17
18  #endif
```

11 RTLS_Material.h

```
1   #ifndef _RTLS_MATERIAL_H_
2   #define _RTLS_MATERIAL_H_
3
4   #include "CGLA/Vec4f.h"
5
6   #include "Common.h"
7
8   namespace RealTimeLSystem {
9
10      class RTLS_Material {
11      public:
12
13          RTLS_Material();
14          RTLS_Material(CGLA::Vec4f _specular,
15                        CGLA::Vec4f _diffuse,
16                        CGLA::Vec4f _ambient,
17                        float _shininess);
18          ~RTLS_Material(void);
19
20          RTLS_Material(const   RTLS_Material& _mat);
21
22          void setMaterial(const RTLS_Material& _mat);
23
24          void setDiffuseColor(const CGLA::Vec4f& _diffuse)
25                                  { diffuse = _diffuse; }
26          void setSpecularColor(const CGLA::Vec4f& _specular)
27                                  { specular = _specular; }
28          void setAmbientColor(const CGLA::Vec4f& _ambient)
29                                  { ambient = _ambient; }
30          void setShininess(const float& _shininess)
31                                  { shininess = _shininess; }
32
33          void increaseDiffuseColor(
34                                   const CGLA::Vec4f& _diffuse) {
35               diffuse *= _diffuse; }
36          void increaseSpecularColor(
37                                  const CGLA::Vec4f& _specular) {
38               specular *= _specular; }
39          void increaseAmbientColor(
40                                  const CGLA::Vec4f& _ambient) {
41               ambient *= _ambient; }
42          void increaseShininess(const float& _shininess) {
43               shininess *= _shininess; }
44
45          const CGLA::Vec4f getDiffuseColor() const {
46                return diffuse; }
47
48          const CGLA::Vec4f getSpecularColor() const {
49                return specular; }
50
51          const CGLA::Vec4f getAmbientColor()  const {
52                return ambient; }
53
54          const float getShininess() const {
55                return shininess; }
56
57      private:
58          CGLA::Vec4f diffuse, specular, ambient;
59          float shininess;
```

```
60        };
61
62    }
63    #endif
```

12 RTLS_Material.cpp

```
1   #include "RTLS_Material.h"
2
3   using namespace CGLA;
4
5   namespace RealTimeLSystem {
6
7       RTLS_Material::RTLS_Material() :
8   specular(Vec4f(0,0,0,0)), diffuse(Vec4f(0,0,0,0)),
9   ambient(Vec4f(0,0,0,0)), shininess(100.0f) {
10
11      }
12
13      RTLS_Material::RTLS_Material(CGLA::Vec4f _specular,
14  CGLA::Vec4f _diffuse, CGLA::Vec4f _ambient, float
15  _shininess) : specular(_specular), diffuse(_diffuse),
16  ambient(_ambient), shininess(_shininess) {
17
18      }
19
20      RTLS_Material::~RTLS_Material(void) {
21
22      }
23
24      RTLS_Material::RTLS_Material(const RTLS_Material& _mat)
25  {
26          specular = _mat.getSpecularColor();
27          diffuse = _mat.getDiffuseColor();
28          ambient = _mat.getAmbientColor();
29          shininess = _mat.getShininess();
30      }
31
32      void RTLS_Material::setMaterial(const RTLS_Material&
33  _mat) {
34          specular = _mat.getSpecularColor();
35          diffuse = _mat.getDiffuseColor();
36          ambient = _mat.getAmbientColor();
37          shininess = _mat.getShininess();
38      }
39  }
```

13 Surface.h

```
1   #ifndef     _SURFACE_H_
2   #define     _SURFACE_H_
3
4   #include "CGLA/Vec3f.h"
5
6   #include "X3DObject/X3DObject.h"
7
8   #include "Common.h"
9
10  namespace RealTimeLSystem {
11
12      class Surface {
13      public:
14          virtual ~Surface(void) = 0;
15          virtual void draw()   = 0;
16          virtual void getBBox(CGLA::Vec3f& lower_left,
17                               CGLA::Vec3f& upper_right) = 0;
18          virtual int getNumberOfPolygons() =     0;
19      };
20
21      class X3DSurface : public Surface {
22      public:
23          X3DSurface(X3DObject* x3dobj):x3d_obj(x3dobj) { }
24          void draw();
25          void getBBox(CGLA::Vec3f& lower_left,
26                       CGLA::Vec3f& upper_right);
27          int getNumberOfPolygons();
28      private:
29          X3DObject* x3d_obj;
30      };
31  }
32  #endif
```

14 Surface.cpp

```
1   #include "Surface.h"
2
3   using namespace CGLA;
4
5   namespace RealTimeLSystem {
6
7       Surface::~Surface(void) { }
8
9       void X3DSurface::draw() {
10
11          x3d_obj->draw();
12      }
13
14      void X3DSurface::getBBox(Vec3f& lower_left,
15                               Vec3f& upper_right) {
16
17          x3d_obj->get_bbox(lower_left, upper_right);
18      }
19
20      int X3DSurface::getNumberOfPolygons() {
21
22          return x3d_obj->getNumberOfTriangles();
23      }
24  }
```

15 Timer.h

```
1   #ifndef     _TIMER_H_
2   #define     _TIMER_H_
3
4   #include <windows.h>
5
6   #include "Common.h"
7
8   namespace RealTimeLSystem {
9       class Timer   {
10
11      public:
12          Timer(void);
13          ~Timer(void);
14
15          void startTimer(DWORD limit);
16          bool isRenderTimeExceeded();
17
18          LONGLONG getFrequency();
19
20      private:
21          LARGE_INTEGER start_clk, now;
22          DWORD assigned_clks;
23      };
24  }
25
26  #endif
```

16 Timer.cpp

```
1   #include "Timer.h"
2
3   namespace RealTimeLSystem {
4
5       Timer::Timer(void) { }
6
7       Timer::~Timer(void) { }
8
9       void Timer::startTimer(DWORD limit) {
10          QueryPerformanceCounter(&start_clk);
11          assigned_clks = limit;
12      }
13
14      bool Timer::isRenderTimeExceeded() {
15          QueryPerformanceCounter(&now);
16          if(now.QuadPart - start_clk.QuadPart >
17             assigned_clks) {
18              return true;
19          }
20          return false;
21      }
22
23      LONGLONG Timer::getFrequency() {
24          LARGE_INTEGER freq;
25          QueryPerformanceFrequency(&freq);
26
27          return freq.QuadPart;
28
29      }
30  }
```

17 Tree.h

```
#ifndef     _TREE_H_
#define     _TREE_H_

#include <GL/glew.h>

#include "CGLA/Mat4x4f.h"
#include "CGLA/Vec4f.h"
#include "CGLA/Vec3f.h"

#include "Common.h"
#include "Timer.h"
#include "LoDSection.h"

namespace RealTimeLSystem {

    class Element;
    class Rule;

    class Tree {
    public:

        Tree(CGLA::Vec3f dollarVec, CGLA::Vec4f light_pos);
        ~Tree(void);

        void F(float l);
        void f(float l);
        void Plus(float l);
        void Minus(float l);
        void And(float l);
        void Hat(float l);
        void Backslash(float l);
        void Slash(float l);
        void Pipe();
        void DollarSign();
        void Push();
        void Pop();
        void PushPolygon();
        void PopPolygon();
        void G(float l);
        void Dot();
        void Uturn();
        void ApplySurface(RTLS::Surface*  surf);
        void Size(float w, bool     absolute);
        void Color(CGLA::Vec3f color, bool absolute);
        void Percent();
        void PushLoD();
        void PopLoD();

        void addRule(RTLS::Rule*    rule);
        void pushBackInitElement(RTLS::Element* e);

        void applyRules();

        void draw();

        void update(CGLA::Vec3f looking_from,
                    DWORD render_clks);

        void drawWithoutTexture();
```

```
60
61          void drawBBox();
62
63          void setPosition(const CGLA::Vec3f&     pos);
64
65          const CGLA::Vec3f getPosition()   const;
66
67          void rotate(CGLA::Axis axis, float angle);
68
69          void scale(CGLA::Vec3f scale);
70
71          void setTexture(GLuint tex);
72
73          void setMaterial(RTLS_Material&   mat);
74
75          void setLightPosition(CGLA::Vec4f& light_pos);
76
77          void printWord();
78
79          int getNumberOfPolygons() const;
80
81          void printTiming();
82
83          int getNumberOfSectionsDrawn();
84
85      private:
86
87          enum STATE {
88              NEW,
89              INIT,
90              IDLE,
91              CREATING_LESS_DETAILED,
92              CREATING_MORE_DETAILED,
93              SWAPPING
94          };
95
96          STATE state;
97
98          void printState();
99
100         inline int decent_to_depth_linear(int distance) {
101                         // y= ax+b; a=1-b, b=-(D-5)/(5-1);
102                         // lod = [1,5],1: highest detail ,
103                         // 5 lowest
104             return Common::round(6-distance –
105                   (float(section_depth - 5)/4.0f) *
106                   (1-(6-distance)));
107         }
108
109         inline int decent_to_depth_polynominal(
110                                             int distance) {
111              // y=ax^2 + b, a = (1-D)/24 b = (25*D - 1)/24,
112             return Common::round(float((1 –
113                     section_depth) * distance *distance +
114                     25 * section_depth - 1) / 24.0f);
115         }
116
117         inline int calculate_distance_linear(float units) {
118             return (int) min(5.0f, ceil(units / 5.0f));
119         }
120
```

```
121         inline int calculate_distance_x_squared(
122                                                 float units) {
123             return (int) min(5, max(1,
124                       ceil(log(units/5.0f)/log(2.0f)+1)));
125         }
126
127         void detectVendor()   {
128             const GLubyte* vendor = glGetString(GL_VENDOR);
129             const GLubyte ati[4] = "ATI";
130             for(int i=0; i<3; i++) {
131                 if(vendor[i] == '\0' ||
132                    vendor[i] != ati[i]) {
133                     isVendorATI = false;
134                     return;
135                 }
136             }
137             isVendorATI = true;
138         }
139
140         bool isVendorATI;
141
142         void createTexture(int max_depth,
143                            int next_lod_if_finished);
144
145         RTLS::Timer* timer;
146
147         RenderTexture* tex;
148
149         unsigned char* tmp_texture;
150
151         int counter;
152
153         bool up2date;
154
155         int section_depth, section_depth_counter;
156
157         int current_distance;
158
159         CGLA::Vec3f     position;
160         CGLA::Vec3f     scaleVec;
161
162         CGLA::Mat4x4f rotationMatrix;
163
164         GLdouble* glRotationMatrix;
165
166         CGLA::Vec3f     dollarVector;
167         CGLA::Vec4f     light_position;
168
169         RTLS::LoDSection* currentSection;
170         RTLS::LoDSection* rootSection;
171
172         void executeWord();
173
174         std::vector<Rule*> rules;
175
176         std::vector<Element*> word1, word2;
177
178         bool use_primary;
179
180         bool pruning;
181         int prun_counter;
```

```
182
183             std::vector<RTLS::LoDSection*> lodSections;
184             std::vector<RTLS::LoDSection*> lodSection_stack;
185         };
186     }
187     #endif
```

18 Tree.cpp

```
#include "Tree.h"
#include "Rule.h"

using namespace std;
using namespace CGLA;

namespace RealTimeLSystem {

    Tree::Tree(Vec3f dollarVec, Vec4f light_pos) :
               dollarVector(dollarVec),
               light_position(light_pos), state(NEW) {
        use_primary = true;
        up2date = false;
        position = Vec3f(0,0,0);
        rotationMatrix = identity_Mat4x4f();
        glRotationMatrix = (GLdouble*)
                           malloc(16*sizeof(GLdouble));
        for(int i = 0; i<4; i++)
            for(int j=0; j<4; j++)
                glRotationMatrix[i*4+j] =
                                rotationMatrix[j][i];

        scaleVec = Vec3f(1,1,1);

        detectVendor();

        glPushAttrib(GL_ALL_ATTRIB_BITS);

        tex = new RenderTexture(TEXTURE_SIZE,
                                TEXTURE_SIZE);

        tex->Initialize();

        if(isVendorATI) {
            tex->BeginCapture();
            glPixelStorei(GL_PACK_SWAP_BYTES, TRUE);
            tex->EndCapture();
        }

        glPopAttrib();

        timer = new Timer();

        tmp_texture = new unsigned
                      char[TEXTURE_SIZE*TEXTURE_SIZE*4];

    }

    Tree::~Tree(void) {
        rules.clear();
        lodSection_stack.clear();
        lodSections.clear();
        word1.clear();
        word2.clear();
        free(glRotationMatrix);
        delete tex;
        delete tmp_texture;
    }

```

```
60      void Tree::F(float l) {
61          if(!pruning) {
62              currentSection->F(l);
63          }
64      }
65
66      void Tree::f(float l) {
67          if(!pruning) {
68              currentSection->f(l);
69          }
70      }
71
72      void Tree::Push() {
73          if(pruning) {
74              prun_counter++;
75          } else {
76              currentSection->Push();
77          }
78      }
79
80      void Tree::Pop() {
81          if(pruning) {
82              prun_counter--;
83              if(prun_counter == 0) {
84                  pruning = false;
85
86                  currentSection->Pop();
87              }
88          } else {
89              currentSection->Pop();
90          }
91      }
92
93      void Tree::Size(float w, bool absolute) {
94          if(!pruning) {
95              currentSection->Size(w, absolute);
96          }
97      }
98
99      void Tree::And(float l) {
100         if(!pruning) {
101             currentSection->And(l);
102         }
103     }
104
105     void Tree::Hat(float l) {
106         if(!pruning) {
107             currentSection->Hat(l);
108         }
109     }
110
111     void Tree::Uturn() {
112         if(!pruning) {
113             currentSection->Uturn();
114         }
115     }
116
117     void Tree::Minus(float l) {
118         if(!pruning) {
119             currentSection->Minus(l);
120         }
```

```
121         }
122
123         void Tree::Plus(float l) {
124             if(!pruning) {
125                 currentSection->Plus(l);
126             }
127         }
128
129         void Tree::Slash(float l) {
130             if(!pruning) {
131                 currentSection->Slash(l);
132             }
133         }
134
135         void Tree::Backslash(float l) {
136             if(!pruning) {
137                 currentSection->Backslash(l);
138             }
139         }
140
141         void Tree::PushPolygon() {
142             if(pruning) {
143                 prun_counter++;
144             } else {
145                 currentSection->PushPolygon();
146             }
147         }
148
149         void Tree::PopPolygon() {
150             if(pruning) {
151                 prun_counter--;
152                 if(prun_counter == 0) {
153                     pruning = false;
154
155                     currentSection->PopPolygon();
156                 }
157             } else {
158                 currentSection->PopPolygon();
159             }
160         }
161
162         void Tree::Dot() {
163             if(!pruning) {
164                 currentSection->Dot();
165             }
166         }
167
168         void Tree::G(float l) {
169             if(!pruning) {
170                 currentSection->G(l);
171             }
172         }
173
174         void Tree::Percent() {
175             if(!pruning) {
176                 pruning = true;
177                 prun_counter = 1;
178             }
179         }
180
181         void Tree::Color(Vec3f color, bool absolute) {
```

```
182             if(!pruning) {
183                 currentSection->Color(color, absolute);
184             }
185         }
186
187         void Tree::DollarSign() {
188             if(!pruning) {
189                 currentSection->DollarSign();
190             }
191         }
192
193         void Tree::ApplySurface(Surface* surf) {
194             if(!pruning) {
195                 currentSection->ApplySurface(surf);
196             }
197         }
198
199         void Tree::PushLoD() {
200             if(pruning) {
201                 prun_counter++;
202             } else {
203                 section_depth_counter++;
204                 if(section_depth_counter > section_depth) {
205                     section_depth = section_depth_counter;
206                 }
207                 if(currentSection->isAligning()) {
208                     currentSection->forceEndAligning();
209                 }
210                 float size  = currentSection->getCurrentSize();
211                 Mat4x4f mat = currentSection->
212                                             getCurrentMatrix();
213
214                 LoDSection* newLoD = new
215                              LoDSection(currentSection,
216                                         section_depth_counter,
217                                         tmp_texture, timer);
218
219                 currentSection->addSubLoD(newLoD);
220
221                 newLoD->setBark(currentSection->getBark());
222
223                 currentSection = newLoD;
224
225                 lodSections.push_back(currentSection);
226                 lodSection_stack.push_back(currentSection);
227             }
228         }
229
230         void Tree::PopLoD() {
231             if(pruning) {
232                 prun_counter--;
233                 if(prun_counter == 0) {
234                     pruning = false;
235
236                     section_depth_counter--;
237                     lodSection_stack.pop_back();
238                     currentSection = lodSection_stack[
239                                     lodSection_stack.size()-1];
240                 }
241             } else {
242                 section_depth_counter--;
```

```
243                 lodSection_stack.pop_back();
244                 currentSection = lodSection_stack[
245                                         lodSection_stack.size()-1];
246             }
247         }
248
249         void Tree::setTexture(GLuint tex) {
250             if(!pruning) {
251                 currentSection->setBark(tex);
252             }
253         }
254
255         void Tree::setMaterial(RTLS_Material& mat) {
256             if(!pruning) {
257                 currentSection->setMaterial(mat);
258             }
259         }
260
261         void Tree::addRule(Rule* rule) {
262             rules.push_back(rule);
263         }
264
265         void Tree::pushBackInitElement(Element* e) {
266             word1.push_back(e);
267         }
268
269         void Tree::applyRules() {
270             vector<Element*> *elements_in;
271             vector<Element*> *elements_out;
272
273             up2date = false;
274
275             if(use_primary) {
276                 elements_in  = &word1;
277                 elements_out = &word2;
278             } else {
279                 elements_out = &word1;
280                 elements_in  = &word2;
281             }
282
283             for(int i=0;i < elements_in->size();i++) { // Loop
284                                           //through all elements
285                 bool no_rules=true;
286                 for(int j=0;j<rules.size();j++) { // Loop
287                                             //through all rules
288                     if(rules[j]->
289                        apply((*elements_in)[i],elements_out)) {
290                         no_rules = false;
291                         break;
292                     }
293                 }
294                 if (no_rules) // If no rules exist for this
295                               //element it should be transfered
296                               //to the new list 'as is'
297                     elements_out->push_back((*elements_in)[i]);
298             }
299
300             elements_in->clear();
301
302             use_primary = !use_primary; // Switch the current
303                                         // list for the next
```

```
304                                         // iteration
305       }
306
307       void Tree::update(Vec3f looking_from,
308                         DWORD render_clks) {
309
310           timer->startTimer(render_clks);
311
312   #ifdef _DEBUG
313           printState();
314   #endif
315
316           switch(state) {
317               case NEW:
318                   current_distance =
319                        calculate_distance_x_squared(
320                         (position - looking_from).length());
321
322                   cout << "Starting at distance: " <<
323                            current_distance << endl;
324
325                   state = INIT;
326                   executeWord();
327                   counter = 0;
328                   break;
329               case INIT:
330                   timer->startTimer((DWORD) 10000000000);
331                   createTexture(decent_to_depth_linear(
332                         current_distance), current_distance);
333                   break;
334               case IDLE:
335                   {
336                       int tmp = calculate_distance_x_squared(
337                          (position - looking_from).length());
338
339                       if(current_distance > tmp) {
340                           //closing in, prepare more detailed
341                           //level
342                           state = CREATING_MORE_DETAILED;
343                           cout << "CREATING_MORE_DETAILED
344                                   decent to: "
345                                <<
346                                   decent_to_depth_linear(
347                                   current_distance-1)
348                                << endl;
349                           counter = 0;
350                       } else if(current_distance < tmp) {
351                           //moving away, prepare less
352                           //detailed level
353                           state = CREATING_LESS_DETAILED;
354                           cout << "CREATING_LESS_DETAILED
355                                    decent to: "
356                                <<
357                                    decent_to_depth_linear(
358                                      current_distance+1)
359                                << endl;
360                           counter = 0;
361                       }
362                       break;
363                   }
```

```
364                 case CREATING_LESS_DETAILED:
365
366                     createTexture(decent_to_depth_linear(
367                       current_distance+1), current_distance+1);
368                     break;
369                 case CREATING_MORE_DETAILED:
370
371                     createTexture(decent_to_depth_linear(
372                       current_distance-1), current_distance-1);
373                     break;
374                 case SWAPPING:
375                     int depth =
376                       decent_to_depth_linear(current_distance);
377                     for(int i=0; i<lodSections.size(); i++) {
378                         if(lodSections[i]->getLvl() <= depth) {
379                             lodSections[i]->swapTexture();
380                         }
381                     }
382                     state = IDLE;
383                     break;
384             }
385         }
386
387         void Tree::draw() {
388
389             if(state == NEW || state == INIT) return; //cant
390                                 //draw anything is these states
391
392             glPushAttrib(GL_ALL_ATTRIB_BITS);
393             glDisable(GL_CULL_FACE);
394             glDisable(GL_NORMALIZE);
395             glMatrixMode(GL_MODELVIEW);
396             glDisable(GL_LIGHTING);
397             glPushMatrix();
398
399             glEnable(GL_ALPHA_TEST);
400             glAlphaFunc(GL_GREATER, 0);
401
402             glTranslatef(position[0],
403                          position[1],
404                          position[2]);
405
406             glScalef(scaleVec[0], scaleVec[1], scaleVec[2]);
407
408             glMultMatrixd(glRotationMatrix);
409
410             if(rootSection) {
411                 rootSection->
412                   drawTexturified(
413                     decent_to_depth_linear(current_distance));
414             }
415
416             glDisable(GL_ALPHA_TEST);
417
418             glPopMatrix();
419             glPopAttrib();
420         }
421
422         void Tree::createTexture(int max_depth,
423                                  int next_distance_if_finished){
424
```

```
425             if(!up2date) {
426                 executeWord();
427             }
428
429             glPushAttrib(GL_ALL_ATTRIB_BITS);
430             glDisable(GL_CULL_FACE);
431
432             bool done;
433
434             while(counter < lodSections.size()) {
435                 if(lodSections[counter]->getLvl()<=max_depth){
436                     tex->BeginCapture();
437                     done = lodSections[counter]->
438                                         createTexture(max_depth);
439                     tex->EndCapture();
440                     if(done) {
441                         counter++;
442                     } else {
443                         glPopAttrib();
444                         return;
445                     }
446                 } else {
447                     counter++;
448                 }
449
450                 if(timer->isRenderTimeExceeded()) {
451                     glPopAttrib();
452                     return;
453                 }
454             }
455
456             glPopAttrib();
457
458             if(counter >= lodSections.size()) {
459                 state = SWAPPING;
460                 current_distance = next_distance_if_finished;
461                 cout << "CURRENT LOD IS NOW: "
462                      << current_distance << endl;
463             }
464         }
465
466         void Tree::executeWord() {
467
468             cout << "Excetuting word" << endl;
469
470             vector<Element*> *elements_in;
471
472             lodSection_stack.clear(); //should allready  be
473                                       //empty
474             lodSections.clear();      //create new lod objects
475             currentSection = new LoDSection(NULL, 1,
476                                             tmp_texture, timer);
477             currentSection->setDollarVector(dollarVector);
478             currentSection->setLightPosition(light_position);
479             rootSection = currentSection;
480
481             lodSections.push_back(currentSection);
482             lodSection_stack.push_back(currentSection);
483
484             section_depth = 1;
485             section_depth_counter = 1;
```

```
486
487             pruning = false;
488
489             if(use_primary) {
490                 elements_in = &word1;
491             } else {
492                 elements_in = &word2;
493             }
494
495             for(int i=0;i<elements_in->size();i++) {
496                 (*elements_in)[i]->execute(this);
497             }
498             up2date = true;
499
500             cout << "LoD depth " << section_depth
501                  << " LoDSections: " << lodSections.size()
502                  << endl;
503         }
504
505         void Tree::drawWithoutTexture() {
506
507             if(!up2date) {
508                 executeWord();
509             }
510
511             glPushAttrib(GL_ALL_ATTRIB_BITS);
512             glPushMatrix();
513
514             glTranslatef(position[0],
515                          position[1],
516                          position[2]);
517
518             glScalef(scaleVec[0], scaleVec[1], scaleVec[2]);
519
520             glMultMatrixd(glRotationMatrix);
521
522             rootSection->justDraw();
523
524             glPopMatrix();
525             glPopAttrib();
526         }
527
528         void Tree::drawBBox() {
529
530             if(!up2date) {
531                 executeWord();
532             }
533
534             glPushMatrix();
535
536             glTranslatef(position[0],
537                          position[1],
538                          position[2]);
539
540             glScalef(scaleVec[0], scaleVec[1], scaleVec[2]);
541
542             glMultMatrixd(glRotationMatrix);
543
544             rootSection->
545             drawBBox(decent_to_depth_linear(current_distance));
546
```

```
547             glPopMatrix();
548         }
549
550         void Tree::setPosition(const CGLA::Vec3f &pos) {
551             position = pos;
552         }
553
554         const CGLA::Vec3f Tree::getPosition() const {
555             return position;
556         }
557
558         void Tree::rotate(const CGLA::Axis axis, float angle) {
559             rotationMatrix = rotationMatrix *
560                                 rotation_Mat4x4f(axis, angle);
561             for(int i = 0; i<4; i++)
562                 for(int j=0; j<4; j++)
563                     glRotationMatrix[i*4+j] =
564                                         rotationMatrix[j][i];
565         }
566
567         void Tree::scale(CGLA::Vec3f scale) {
568             scaleVec = scale;
569         }
570
571         void Tree::printWord() {
572             vector<Element*> *elements_in;
573             if(use_primary) {
574                 elements_in = &word1;
575             } else {
576                 elements_in = &word2;
577             }
578
579             for(int i=0;i<elements_in->size();i++) {
580                 (*elements_in)[i]->print();
581             }
582         }
583
584         void Tree::printState() {
585             switch(state) {
586                 case NEW:
587                     cout << "NEW" << endl;
588                     break;
589                 case INIT:
590                     cout << "INIT" << endl;
591                     break;
592                 case IDLE:
593                     cout << "IDLE" << endl;
594                     break;
595                 case CREATING_LESS_DETAILED:
596                     cout << "CREATING_LESS_DETAILED" << endl;
597                     break;
598                 case CREATING_MORE_DETAILED:
599                     cout << "CREATING_MORE_DETAILED" << endl;
600                     break;
601                 case SWAPPING:
602                     cout << "SWAPPING" << endl;
603                     break;
604             }
605         }
606
607         int Tree::getNumberOfPolygons() const {
```

```
608             int res = 0;
609             LoDSection* lods;
610 
611             for(int i=0; i<lodSections.size(); i++) {
612                 lods = lodSections[i];
613                 res += lods->getNumberOfPolygons();
614             }
615             return res;
616         }
617 
618         void Tree::printTiming() {
619             LARGE_INTEGER* res = new LARGE_INTEGER[16];
620             for(int j=0; j<16; j++) {
621                 res[j].QuadPart = 0;
622             }
623 
624             LARGE_INTEGER* tmp;
625             for(int i=0; i<lodSections.size(); i++) {
626                 if(lodSections[i]->getLvl() <=
627                     decent_to_depth_linear(current_distance)) {
628                     tmp = lodSections[i]->getTimings();
629                     for(int j=0; j<16; j++) {
630                         res[j].QuadPart += tmp[j].QuadPart;
631                     }
632                 }
633             }
634 
635             cout << "CPU frequency: " << timer->getFrequency()
636                  << endl;
637             for(int j=0; j<16; j++) {
638                 cout << rootSection->getStateLabel(j) << ": \t"
639                      << res[j].QuadPart << endl;
640             }
641         }
642 
643         int Tree::getNumberOfSectionsDrawn() {
644             int res = 0;
645             for(int i=0; i<lodSections.size(); i++) {
646                 if(lodSections[i]->getLvl() <=
647                     decent_to_depth_linear(current_distance)) {
648                     res++;
649                 }
650             }
651             return res;
652         }
653     }
```

19 Basic.h

```
#ifndef     _BASIC_H_
#define     _BASIC_H_

#include "Common.h"
#include "Rule.h"
#include "Element.h"
#include "Surface.h"

namespace Basic   {

    class RuleA1 : public RTLS::Rule {
    public:
        bool apply(RTLS::Element* element_in,
                std::vector<RTLS::Element*> *lsystem_out);
    };

    class RuleA2 : public RTLS::Rule {
    public:
        RuleA2(RTLS::Surface* _leaf,
               RTLS::Surface* _flower) : leaf(_leaf),
                                         flower(_flower) {}
        bool apply(RTLS::Element* element_in,
                std::vector<RTLS::Element*> *lsystem_out);
    private:
        RTLS::Surface *leaf, *flower;
    };

    class RuleA3 : public RTLS::Rule {
    public:
        RuleA3(RTLS::Surface* _leaf,
               RTLS::Surface* _flower) : leaf(_leaf),
                                         flower(_flower) {}
        bool apply(RTLS::Element* element_in,
                std::vector<RTLS::Element*> *lsystem_out);
    private:
        RTLS::Surface *leaf,  *flower;
    };

    class RuleA4 : public RTLS::Rule {
    public:
        RuleA4(RTLS::Surface* _leaf,
               RTLS::Surface* _flower) : leaf(_leaf),
                                         flower(_flower) {}
        bool apply(RTLS::Element* element_in,
                std::vector<RTLS::Element*> *lsystem_out);
    private:
        RTLS::Surface *leaf,  *flower;
    };

    class RuleA5 : public RTLS::Rule {
    public:
        RuleA5(RTLS::Surface* _leaf,
               RTLS::Surface* _flower) : leaf(_leaf),
                                         flower(_flower) {}
        bool apply(RTLS::Element* element_in,
                std::vector<RTLS::Element*> *lsystem_out);
    private:
        RTLS::Surface *leaf,  *flower;
    };
```

```
60
61      class RuleA6 : public RTLS::Rule {
62      public:
63          RuleA6(RTLS::Surface* _leaf,
64                 RTLS::Surface* _flower) : leaf(_leaf),
65                                           flower(_flower) {}
66          bool apply(RTLS::Element* element_in,
67                     std::vector<RTLS::Element*> *lsystem_out);
68      private:
69          RTLS::Surface *leaf,  *flower;
70      };
71
72      class A : public RTLS::Element {
73      public:
74          float l;
75          float w;
76          A(float   _l,   float _w) : l(_l), w(_w) {};
77          void execute(RTLS::Tree*    tree) {     };
78          void print();
79      };
80
81      class B : public RTLS::Element {
82      public:
83          float l;
84          float w;
85          B(float   _l,   float _w) : l(_l), w(_w) {};
86          void execute(RTLS::Tree*    tree) {     };
87          void print();
88      };
89
90      class C : public RTLS::Element {
91      public:
92          float l;
93          float w;
94          C(float   _l,   float _w) : l(_l), w(_w) {};
95          void execute(RTLS::Tree*    tree) {     };
96          void print();
97      };
98
99      class D : public RTLS::Element {
100     public:
101         float l;
102         float w;
103         D(float   _l,   float _w) : l(_l), w(_w) {};
104         void execute(RTLS::Tree*    tree) {     };
105         void print();
106     };
107
108     class E : public RTLS::Element {
109     public:
110         float l;
111         float w;
112         E(float   _l,   float _w) : l(_l), w(_w) {};
113         void execute(RTLS::Tree*    tree) {     };
114         void print();
115     };
116
117     class H : public RTLS::Element {
118     public:
119         float l;
120         float w;
```

```
121             H(float   _l,   float _w) : l(_l), w(_w) {};
122             void execute(RTLS::Tree*    tree) {     };
123             void print();
124         };
125     }
126     #endif
```

20 Basic.cpp

```
#include "Basic.h"

using namespace std;
using namespace RealTimeLSystem;

namespace Basic {

    const float r1 = 0.9f;
    const float r2 = 0.7f;
    const float a0 = 20;
    const float a2 = -30;
    const float d = 137.5f;
    const float wr = 0.707f;

    const float af = 0.0f;

    void A::print () {
        cout << "A(" << l << "," << w << ")";
    }
    void B::print () {
        cout << "B";
    }
    void C::print () {
        cout << "C";
    }

    void D::print () {
        cout << "D";
    }

    void E::print () {
        cout << "E";
    }

    void H::print () {
        cout << "H";
    }

    bool RuleA1::apply(Element* element_in,
                       vector<Element*> *lsystem_out) {
        if (A *a=dynamic_cast<A*>(element_in)) {
            lsystem_out->push_back(new Size(a->w));
            lsystem_out->push_back(new F(a->l));

            if(rand()/float(RAND_MAX) > 0.05f) {

                lsystem_out->push_back(new PushLoD());

                lsystem_out->push_back(new And(a0));
                lsystem_out->push_back(new Slash(af));

                lsystem_out->push_back(new B(a->l*r2,
                                             a->w*wr));

                lsystem_out->push_back(new PopLoD());
            }

            lsystem_out->push_back(new Slash(d));
            lsystem_out->push_back(new A(a->l*r1,a->w*wr));
```

```
60              return true;
61          }
62          return false;
63      }
64
65      bool RuleA2::apply(Element* element_in,
66                         vector<Element*> *lsystem_out) {
67          if (B *b=dynamic_cast<B*>(element_in)) {
68
69              lsystem_out->push_back(new Size(b->w));
70              lsystem_out->push_back(new F(b->l));
71
72              lsystem_out->push_back(new Push());
73              lsystem_out->push_back(new DollarSign());
74              lsystem_out->push_back(new Plus(a2));
75
76              lsystem_out->push_back(new
77                                     ApplySurface(flower));
78
79              lsystem_out->push_back(new Slash(2*a2));
80
81              lsystem_out->push_back(new ApplySurface(leaf));
82              lsystem_out->push_back(new Slash(2*a2));
83              lsystem_out->push_back(new ApplySurface(leaf));
84              lsystem_out->push_back(new Slash(2*a2));
85              lsystem_out->push_back(new ApplySurface(leaf));
86              lsystem_out->push_back(new Slash(2*a2));
87              lsystem_out->push_back(new ApplySurface(leaf));
88              lsystem_out->push_back(new Pop());
89
90              if(rand()/float(RAND_MAX) > 0.10f) {
91
92                  lsystem_out->push_back(new PushLoD());
93                  lsystem_out->push_back(new Minus(a2));
94                  lsystem_out->push_back(new Slash(af));
95                  lsystem_out->push_back(new DollarSign());
96                  lsystem_out->push_back(new C(b->l*r2,
97                                               b->w*wr));
98                  lsystem_out->push_back(new PopLoD());
99              }
100
101             lsystem_out->push_back(new C(b->l*r1,b->w*wr));
102
103             return true;
104         }
105         return false;
106     }
107
108     bool RuleA3::apply(Element* element_in,
109                        vector<Element*> *lsystem_out) {
110         if (C *c=dynamic_cast<C*>(element_in)) {
111
112             lsystem_out->push_back(new Size(c->w));
113             lsystem_out->push_back(new F(c->l));
114
115             lsystem_out->push_back(new Push());
116             lsystem_out->push_back(new DollarSign());
117             lsystem_out->push_back(new Minus(a2));
118
119             lsystem_out->push_back(new ApplySurface(leaf));
120             lsystem_out->push_back(new Minus(2*a2));
```

```
121                 lsystem_out->push_back(new ApplySurface(leaf));
122                 lsystem_out->push_back(new Minus(2*a2));
123                 lsystem_out->push_back(new ApplySurface(leaf));
124                 lsystem_out->push_back(new Minus(2*a2));
125                 lsystem_out->push_back(new ApplySurface(leaf));
126                 lsystem_out->push_back(new Pop());
127
128                 if(rand()/float(RAND_MAX) > 0.10f) {
129                     lsystem_out->push_back(new PushLoD());
130                     lsystem_out->push_back(new Plus(a2));
131                     lsystem_out->push_back(new Slash(af));
132                     lsystem_out->push_back(new DollarSign());
133
134                     lsystem_out->push_back(new D(c->l*r2,
135                                                  c->w*wr));
136
137                     lsystem_out->push_back(new PopLoD());
138                 }
139
140                 lsystem_out->push_back(new D(c->l*r1,c->w*wr));
141
142                 return true;
143             }
144             return false;
145         }
146
147         bool RuleA4::apply(Element* element_in,
148                            vector<Element*> *lsystem_out) {
149             if (D *d=dynamic_cast<D*>(element_in)) {
150
151                 lsystem_out->push_back(new Size(d->w));
152                 lsystem_out->push_back(new F(d->l));
153
154                 lsystem_out->push_back(new Push());
155                 lsystem_out->push_back(new DollarSign());
156                 lsystem_out->push_back(new Plus(a2));
157
158                 lsystem_out->push_back(new
159                                        ApplySurface(flower));
160                 lsystem_out->push_back(new Slash(2*a2));
161
162                 lsystem_out->push_back(new ApplySurface(leaf));
163                 lsystem_out->push_back(new Slash(2*a2));
164                 lsystem_out->push_back(new ApplySurface(leaf));
165                 lsystem_out->push_back(new Slash(2*a2));
166                 lsystem_out->push_back(new ApplySurface(leaf));
167                 lsystem_out->push_back(new Slash(2*a2));
168                 lsystem_out->push_back(new ApplySurface(leaf));
169                 lsystem_out->push_back(new Pop());
170
171                 if(rand()/float(RAND_MAX) > 0.15f) {
172                     lsystem_out->push_back(new PushLoD());
173                     lsystem_out->push_back(new Minus(a2));
174                     lsystem_out->push_back(new Slash(af));
175                     lsystem_out->push_back(new DollarSign());
176
177                     lsystem_out->push_back(new E(d->l*r2,
178                                                  d->w*wr));
179
180                     lsystem_out->push_back(new PopLoD());
181                 }
```

```
182
183             lsystem_out->push_back(new E(d->l*r1,d->w*wr));
184
185             return true;
186         }
187         return false;
188     }
189
190     bool RuleA5::apply(Element* element_in,
191                        vector<Element*> *lsystem_out) {
192         if (E *e=dynamic_cast<E*>(element_in)) {
193
194             lsystem_out->push_back(new Size(e->w));
195             lsystem_out->push_back(new F(e->l));
196
197             lsystem_out->push_back(new Push());
198             lsystem_out->push_back(new DollarSign());
199             lsystem_out->push_back(new Minus(a2));
200
201             lsystem_out->push_back(new
202                                    ApplySurface(flower));
203             lsystem_out->push_back(new Minus(2*a2));
204
205             lsystem_out->push_back(new ApplySurface(leaf));
206             lsystem_out->push_back(new Minus(2*a2));
207             lsystem_out->push_back(new ApplySurface(leaf));
208             lsystem_out->push_back(new Minus(2*a2));
209             lsystem_out->push_back(new ApplySurface(leaf));
210             lsystem_out->push_back(new Minus(2*a2));
211             lsystem_out->push_back(new ApplySurface(leaf));
212             lsystem_out->push_back(new Pop());
213
214             lsystem_out->push_back(new Push());
215             lsystem_out->push_back(new Plus(a2));
216             lsystem_out->push_back(new DollarSign());
217
218             lsystem_out->push_back(new H(e->l*r2,
219                                          e->w*wr));
220
221             lsystem_out->push_back(new Pop());
222
223             lsystem_out->push_back(new H(e->l*r1,e->w*wr));
224
225             return true;
226         }
227         return false;
228     }
229
230     bool RuleA6::apply(Element* element_in,
231                        vector<Element*> *lsystem_out) {
232         if (H *h=dynamic_cast<H*>(element_in)) {
233             lsystem_out->push_back(new Size(h->w));
234             lsystem_out->push_back(new F(h->l));
235
236             lsystem_out->push_back(new Push());
237             lsystem_out->push_back(new DollarSign());
238             lsystem_out->push_back(new Plus(a2));
239
240             lsystem_out->push_back(new
241                                    ApplySurface(flower));
242             lsystem_out->push_back(new Slash(2*a2));
```

```
243
244                 lsystem_out->push_back(new ApplySurface(leaf));
245                 lsystem_out->push_back(new Slash(2*a2));
246                 lsystem_out->push_back(new ApplySurface(leaf));
247                 lsystem_out->push_back(new Slash(2*a2));
248                 lsystem_out->push_back(new ApplySurface(leaf));
249                 lsystem_out->push_back(new Slash(2*a2));
250                 lsystem_out->push_back(new ApplySurface(leaf));
251                 lsystem_out->push_back(new Pop());
252
253                 lsystem_out->push_back(new Push());
254                 lsystem_out->push_back(new Minus(a2));
255                 lsystem_out->push_back(new DollarSign());
256
257                 lsystem_out->push_back(new E(h->l*r2,
258                                              h->w*wr));
259
260                 lsystem_out->push_back(new Pop());
261
262                 lsystem_out->push_back(new E(h->l*r1,h->w*wr));
263
264                 return true;
265             }
266             return false;
267         }
268     }
```

21 Bush.h

```
1   #ifndef     _BUSH_H_
2   #define     _BUSH_H_
3
4   #include "Common.h"
5   #include "Rule.h"
6   #include "Element.h"
7
8   namespace Bush {
9
10      const float   a =   22.5f;
11
12      class A1 : public RTLS::Element {
13      public:
14          A1() {};
15          void execute(RTLS::Tree*    tree) {     };
16          void print() { std::cout << "A1"; };
17      };
18
19      class A2 : public RTLS::Element {
20      public:
21          A2() {};
22          void execute(RTLS::Tree*    tree) {     };
23          void print() { std::cout << "A2"; };
24      };
25
26      class A3 : public RTLS::Element {
27      public:
28          A3() {};
29          void execute(RTLS::Tree*    tree) {     };
30          void print() { std::cout << "A3"; };
31      };
32
33      class A4 : public RTLS::Element {
34      public:
35          A4() {};
36          void execute(RTLS::Tree*    tree) {     };
37          void print() { std::cout << "A4"; };
38      };
39
40      class A5 : public RTLS::Element {
41      public:
42          A5() {};
43          void execute(RTLS::Tree*    tree) {     };
44          void print() { std::cout << "A5"; };
45      };
46
47      class S : public RTLS::Element {
48      public:
49          S() {};
50          void execute(RTLS::Tree*    tree) {     };
51          void print() { std::cout << "S"; };
52      };
53
54      class L : public RTLS::Element {
55      public:
56          L() {};
57          void execute(RTLS::Tree*    tree) {     };
58          void print() { std::cout << "L"; };
59      };
```

```
60
61        class Bush1   : public RTLS::Rule {
62        public:
63            bool apply(RTLS::Element* element_in,
64                       std::vector<RTLS::Element*> *lsystem_out);
65        };
66        class Bush2   : public RTLS::Rule {
67        public:
68            bool apply(RTLS::Element* element_in,
69                       std::vector<RTLS::Element*> *lsystem_out);
70        };
71        class Bush3   : public RTLS::Rule {
72        public:
73            bool apply(RTLS::Element* element_in,
74                       std::vector<RTLS::Element*> *lsystem_out);
75        };
76        class Bush4   : public RTLS::Rule {
77        public:
78            bool apply(RTLS::Element* element_in,
79                       std::vector<RTLS::Element*> *lsystem_out);
80        };
81    }
82    #endif
```

22 Bush.cpp

```
#include "Bush.h"

using namespace std;
using namespace RealTimeLSystem;

namespace Bush {

    bool Bush1::apply(Element* element_in,
                      vector<Element*> *lsystem_out) {
        bool a1 = false;
        bool a2 = false;
        bool a3 = false;
        bool a4 = false;
        bool a5 = false;

        if(A1 *e=dynamic_cast<A1*>(element_in)) {
            a1 = true;
        } else if(A2 *e=dynamic_cast<A2*>(element_in)) {
            a2 = true;
        } else if(A3 *e=dynamic_cast<A3*>(element_in)) {
            a3 = true;
        } else if(A4 *e=dynamic_cast<A4*>(element_in)) {
            a4 = true;
        } else if(A5 *e=dynamic_cast<A5*>(element_in)) {
            a5 = true;
        }

        if (a1 || a2 || a3 || a4 || a5 ) {
            if(a1 || a2 || a3 || a4 ) {
                lsystem_out->push_back(new PushLoD());
            } else {
                lsystem_out->push_back(new Push());
            }
            lsystem_out->push_back(new And(a));
            lsystem_out->push_back(new F());
            lsystem_out->push_back(new L());
            lsystem_out->push_back(new Size());

            if(a1) {
                lsystem_out->push_back(new A2());
            } else if(a2) {
                lsystem_out->push_back(new A3());
            } else if(a3) {
                lsystem_out->push_back(new A4());
            } else {
                lsystem_out->push_back(new A5());
            }

            if(a1 || a2 || a3 || a4) {
                lsystem_out->push_back(new PopLoD());
            } else {
                lsystem_out->push_back(new Pop());
            }
            lsystem_out->push_back(new Slash(a));
            lsystem_out->push_back(new Slash(a));
            lsystem_out->push_back(new Slash(a));
            lsystem_out->push_back(new Slash(a));
            lsystem_out->push_back(new Slash(a));
            lsystem_out->push_back(new Color());
```

```
60
61              if(a1 || a2 || a3 || a4) {
62                  lsystem_out->push_back(new PushLoD());
63              } else {
64                  lsystem_out->push_back(new Push());
65              }
66
67              lsystem_out->push_back(new And(a));
68              lsystem_out->push_back(new F());
69              lsystem_out->push_back(new L());
70              lsystem_out->push_back(new Size());
71
72              if(a1) {
73                  lsystem_out->push_back(new A2());
74              } else if(a2) {
75                  lsystem_out->push_back(new A3());
76              } else if(a3) {
77                  lsystem_out->push_back(new A4());
78              } else {
79                  lsystem_out->push_back(new A5());
80              }
81
82              if(a1 || a2 || a3 || a4) {
83                  lsystem_out->push_back(new PopLoD());
84              } else {
85                  lsystem_out->push_back(new Pop());
86              }
87              lsystem_out->push_back(new Slash(a));
88              lsystem_out->push_back(new Slash(a));
89              lsystem_out->push_back(new Slash(a));
90              lsystem_out->push_back(new Slash(a));
91              lsystem_out->push_back(new Slash(a));
92              lsystem_out->push_back(new Slash(a));
93              lsystem_out->push_back(new Slash(a));
94              lsystem_out->push_back(new Color());
95
96              if(a1 || a2 || a3 || a4) {
97                  lsystem_out->push_back(new PushLoD());
98              } else {
99                  lsystem_out->push_back(new Push());
100             }
101
102             lsystem_out->push_back(new And(a));
103             lsystem_out->push_back(new F());
104             lsystem_out->push_back(new L());
105             lsystem_out->push_back(new Size());
106
107             if(a1) {
108                 lsystem_out->push_back(new A2());
109             } else if(a2) {
110                 lsystem_out->push_back(new A3());
111             } else if(a3) {
112                 lsystem_out->push_back(new A4());
113             } else {
114                 lsystem_out->push_back(new A5());
115             }
116
117             if(a1 || a2 || a3 || a4) {
118                 lsystem_out->push_back(new PopLoD());
119             } else {
120                 lsystem_out->push_back(new Pop());
```

```
121             }
122
123             return true;
124         }
125         return false;
126     }
127
128     bool Bush2::apply(Element* element_in,
129                       vector<Element*> *lsystem_out) {
130         if (F *e=dynamic_cast<F*>(element_in)) {
131             lsystem_out->push_back(new S());
132             lsystem_out->push_back(new Slash(a));
133             lsystem_out->push_back(new Slash(a));
134             lsystem_out->push_back(new Slash(a));
135             lsystem_out->push_back(new Slash(a));
136             lsystem_out->push_back(new Slash(a));
137             lsystem_out->push_back(new F());
138             return true;
139         }
140         return false;
141     }
142
143     bool Bush3::apply(Element* element_in,
144                       vector<Element*> *lsystem_out) {
145         if (S *e=dynamic_cast<S*>(element_in)) {
146             lsystem_out->push_back(new F());
147             lsystem_out->push_back(new L());
148             return true;
149         }
150         return false;
151     }
152
153     bool Bush4::apply(Element* element_in,
154                       vector<Element*> *lsystem_out) {
155         if (L *e=dynamic_cast<L*>(element_in)) {
156             lsystem_out->push_back(new Push());
157             lsystem_out->push_back(new Color());
158             lsystem_out->push_back(new Color());
159             lsystem_out->push_back(new Color());
160             lsystem_out->push_back(new Hat(a));
161             lsystem_out->push_back(new Hat(a));
162             lsystem_out->push_back(new PushPolygon());
163             lsystem_out->push_back(new Plus(a));
164             lsystem_out->push_back(new f());
165             lsystem_out->push_back(new Minus(a));
166             lsystem_out->push_back(new f());
167             lsystem_out->push_back(new Minus(a));
168             lsystem_out->push_back(new f());
169             lsystem_out->push_back(new Plus(a));
170             lsystem_out->push_back(new Uturn());
171             lsystem_out->push_back(new Plus(a));
172             lsystem_out->push_back(new f());
173             lsystem_out->push_back(new Minus(a));
174             lsystem_out->push_back(new f());
175             lsystem_out->push_back(new Minus(a));
176             lsystem_out->push_back(new f());
177             lsystem_out->push_back(new PopPolygon());
178             lsystem_out->push_back(new Pop());
179             return true;
180         }
181         return false;
```

```
182         }
183     }
```

23 Capsella.h

```
1   #ifndef _CAPSELLA_H_
2   #define _CAPSELLA_H_
3
4   #include "CGLA/Vec4f.h"
5
6   #include "Common.h"
7   #include "Rule.h"
8   #include "Element.h"
9
10  namespace Capsella {
11
12      class a : public RTLS::Element {
13      public:
14          float t;
15          a(float   _t)   : t(_t)     {};
16          void execute(RTLS::Tree*    tree) {     };
17          void print() { std::cout << "a(" << t << ")"; };
18      };
19      class A : public RTLS::Element {
20      public:
21          A() {};
22          void execute(RTLS::Tree*    tree) {     };
23          void print() { std::cout << "A"; };
24      };
25      class I : public RTLS::Element {
26      public:
27          float t;
28          I(float   _t)   : t(_t)     {};
29          void execute(RTLS::Tree*    tree) {     };
30          void print() { std::cout << "I(" << t << ")"; };
31      };
32      class u : public RTLS::Element {
33      public:
34          float t;
35          u(float   _t)   : t(_t)     {};
36          void execute(RTLS::Tree*    tree) {     };
37          void print() { std::cout << "u(" << t << ")"; };
38      };
39      class L : public RTLS::Element {
40      public:
41          L() {};
42          void execute(RTLS::Tree*    tree) {     };
43          void print() { std::cout << "L"; };
44      };
45      class K : public RTLS::Element {
46      public:
47          K() {};
48          void execute(RTLS::Tree*    tree) {     };
49          void print() { std::cout << "K"; };
50      };
51      class X : public RTLS::Element {
52      public:
53          float t;
54          X(float   _t)   : t(_t)     {};
55          void execute(RTLS::Tree*    tree) {     };
56          void print() { std::cout << "X(" << t << ")"; };
57      };
58      class Capsella1     : public RTLS::Rule {
59      public:
```

```
60          bool apply(RTLS::Element* element_in,
61  std::vector<RTLS::Element*>   *lsystem_out);
62      };
63      class Capsella2     : public RTLS::Rule {
64      public:
65          bool apply(RTLS::Element* element_in,
66  std::vector<RTLS::Element*>   *lsystem_out);
67      };
68      class Capsella3     : public RTLS::Rule {
69      public:
70          bool apply(RTLS::Element* element_in,
71  std::vector<RTLS::Element*>   *lsystem_out);
72      };
73      class Capsella4     : public RTLS::Rule {
74      public:
75          bool apply(RTLS::Element* element_in,
76  std::vector<RTLS::Element*>   *lsystem_out);
77      };
78      class Capsella5     : public RTLS::Rule {
79      public:
80          bool apply(RTLS::Element* element_in,
81  std::vector<RTLS::Element*>   *lsystem_out);
82      };
83      class Capsella6     : public RTLS::Rule {
84      public:
85          bool apply(RTLS::Element* element_in,
86  std::vector<RTLS::Element*>   *lsystem_out);
87      };
88      class Capsella7     : public RTLS::Rule {
89      public:
90          bool apply(RTLS::Element* element_in,
91  std::vector<RTLS::Element*>   *lsystem_out);
92      };
93      class Capsella8     : public RTLS::Rule {
94      public:
95          bool apply(RTLS::Element* element_in,
96  std::vector<RTLS::Element*>   *lsystem_out);
97      };
98      class Capsella9     : public RTLS::Rule {
99      public:
100         bool apply(RTLS::Element* element_in,
101 std::vector<RTLS::Element*>   *lsystem_out);
102     };
103     class Capsella10 : public RTLS::Rule  {
104     public:
105         bool apply(RTLS::Element* element_in,
106 std::vector<RTLS::Element*>   *lsystem_out);
107     };
108     class Capsella11 : public RTLS::Rule  {
109     public:
110         bool apply(RTLS::Element* element_in,
111 std::vector<RTLS::Element*>   *lsystem_out);
112     };
113 }
114 #endif
```

24 Capsella.cpp

```
#include "Capsella.h"

using namespace std;
using namespace CGLA;
using namespace RealTimeLSystem;

namespace Capsella {

    const float r1 = 0.9f;
    const float r2 = 0.6f;
    const float a0 = 70;
    const float a1 = 9;
    const float a2 = 18;
    const float a3 = 50;

    const float af = 25;

    const float i0= 10;
    const float i1 = 5;
    const float i2 = 8;
    const float i3 = 7;
    const float i4 = 2;

    const float d0 = 137.5f;
    const float d1 = 90.0f;
    const float wr = 0.707f;

    bool Capsella1::apply(Element* element_in,
                          vector<Element*> *lsystem_out) {
        if (a *e=dynamic_cast<a*>(element_in)) {
            if(e->t > 0) {
                lsystem_out->push_back(new Push());
                lsystem_out->push_back(new And(a0));
                lsystem_out->push_back(new L());
                lsystem_out->push_back(new Pop());
                lsystem_out->push_back(new Slash(d0));
                lsystem_out->push_back(new I(i0));
                lsystem_out->push_back(new a(e->t-1));
                return true;
            }
        }
        return false;
    }
    bool Capsella2::apply(Element* element_in,
                          vector<Element*> *lsystem_out) {
        if (a *e=dynamic_cast<a*>(element_in)) {
            if(e->t == 0) {
                lsystem_out->push_back(new Push());
                lsystem_out->push_back(new And(a0));
                lsystem_out->push_back(new L());
                lsystem_out->push_back(new Pop());
                lsystem_out->push_back(new Slash(d0));
                lsystem_out->push_back(new I(i0));
                lsystem_out->push_back(new A());
                return true;
            }
        }
        return false;
    }
```

```
60      bool Capsella3::apply(Element* element_in,
61                            vector<Element*> *lsystem_out) {
62          if (A *e=dynamic_cast<A*>(element_in)) {
63              lsystem_out->push_back(new PushLoD());
64
65              lsystem_out->push_back(new And(a2));
66              lsystem_out->push_back(new u(4));
67              lsystem_out->push_back(new F());
68              lsystem_out->push_back(new F());
69              lsystem_out->push_back(new I(i0));
70              lsystem_out->push_back(new I(i1));
71              lsystem_out->push_back(new PushLoD());
72              lsystem_out->push_back(new X(i1));
73              lsystem_out->push_back(new K());
74              lsystem_out->push_back(new K());
75              lsystem_out->push_back(new K());
76              lsystem_out->push_back(new K());
77              lsystem_out->push_back(new PopLoD());
78              lsystem_out->push_back(new PopLoD());
79              lsystem_out->push_back(new Slash(d0));
80              lsystem_out->push_back(new I(i2));
81              lsystem_out->push_back(new A());
82              return true;
83          }
84          return false;
85      }
86      bool Capsella4::apply(Element* element_in,
87                            vector<Element*> *lsystem_out) {
88          if (I *e=dynamic_cast<I*>(element_in)) {
89              if(e->t > 0) {
90                  lsystem_out->push_back(new F());
91                  lsystem_out->push_back(new I(e->t-1));
92                  return true;
93              }
94          }
95          return false;
96      }
97      bool Capsella5::apply(Element* element_in,
98                            vector<Element*> *lsystem_out) {
99          if (I *e=dynamic_cast<I*>(element_in)) {
100             if(e->t == 0) {
101                 lsystem_out->push_back(new F());
102                 return true;
103             }
104         }
105         return false;
106     }
107     bool Capsella6::apply(Element* element_in,
108                           vector<Element*> *lsystem_out) {
109         if (u *e=dynamic_cast<u*>(element_in)) {
110             if(e->t > 0) {
111                 lsystem_out->push_back(new And(a1));
112                 lsystem_out->push_back(new u(e->t-1));
113                 return true;
114             }
115         }
116         return false;
117     }
118     bool Capsella7::apply(Element* element_in,
119                           vector<Element*> *lsystem_out) {
120         if (u *e=dynamic_cast<u*>(element_in)) {
```

```
121             if(e->t == 0) {
122                 lsystem_out->push_back(new And(a1));
123                 return true;
124             }
125         }
126         return false;
127     }
128     bool Capsella8::apply(Element* element_in,
129                           vector<Element*> *lsystem_out) {
130         if (L *e=dynamic_cast<L*>(element_in)) {
131 
132             lsystem_out->push_back(new PushLoD());
133             lsystem_out->push_back(new PushPolygon());
134             lsystem_out->push_back(new Dot());
135             lsystem_out->push_back(new Minus(a2));
136             lsystem_out->push_back(new F());
137             lsystem_out->push_back(new I(i3));
138             lsystem_out->push_back(new Plus(a2));
139             lsystem_out->push_back(new F());
140             lsystem_out->push_back(new I(i3));
141             lsystem_out->push_back(new Plus(a2));
142             lsystem_out->push_back(new F());
143             lsystem_out->push_back(new I(i3));
144 
145             lsystem_out->push_back(new Minus(a2));
146             lsystem_out->push_back(new Uturn());
147             lsystem_out->push_back(new Minus(a2));
148             lsystem_out->push_back(new F());
149             lsystem_out->push_back(new I(i3));
150             lsystem_out->push_back(new Plus(a2));
151             lsystem_out->push_back(new F());
152             lsystem_out->push_back(new I(i3));
153             lsystem_out->push_back(new Plus(a2));
154             lsystem_out->push_back(new F());
155             lsystem_out->push_back(new I(i3));
156 
157             lsystem_out->push_back(new PopPolygon());
158             lsystem_out->push_back(new PopLoD());
159 
160             return true;
161         }
162         return false;
163     }
164     bool Capsella9::apply(Element* element_in,
165                           vector<Element*> *lsystem_out) {
166         if (K *e=dynamic_cast<K*>(element_in)) {
167 
168             RTLS_Material flower_mat =
169                     RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
170                                   Vec4f(0.8, 0.8, 0.2, 1.),
171                                   Vec4f(0.8, 0.8, 0.2, 1.),
172                                   1.);
173 
174             lsystem_out->push_back(new Push());
175             lsystem_out->push_back(new
176                                  Material(flower_mat));
177             lsystem_out->push_back(new And(a2));
178             lsystem_out->push_back(new PushPolygon());
179             lsystem_out->push_back(new Slash(af));
180             lsystem_out->push_back(new Dot());
181             lsystem_out->push_back(new Plus(a2));
```

```
182                 lsystem_out->push_back(new F());
183                 lsystem_out->push_back(new I(i4));
184                 lsystem_out->push_back(new Minus(a2));
185                 lsystem_out->push_back(new Minus(a2));
186                 lsystem_out->push_back(new F());
187                 lsystem_out->push_back(new I(i4));
188                 lsystem_out->push_back(new PopPolygon());
189                 lsystem_out->push_back(new Pop());
190
191                 lsystem_out->push_back(new Push());
192                 lsystem_out->push_back(new
193                                        Material(flower_mat));
194                 lsystem_out->push_back(new And(a2));
195                 lsystem_out->push_back(new PushPolygon());
196                 lsystem_out->push_back(new Slash(af));
197                 lsystem_out->push_back(new Dot());
198                 lsystem_out->push_back(new Minus(a2));
199                 lsystem_out->push_back(new F());
200                 lsystem_out->push_back(new I(i4));
201                 lsystem_out->push_back(new Plus(a2));
202                 lsystem_out->push_back(new Plus(a2));
203                 lsystem_out->push_back(new F());
204                 lsystem_out->push_back(new I(i4));
205                 lsystem_out->push_back(new PopPolygon());
206                 lsystem_out->push_back(new Pop());
207
208                 lsystem_out->push_back(new Slash(d1));
209
210                 return true;
211             }
212             return false;
213         }
214         bool Capsella10::apply(Element* element_in,
215                                vector<Element*> *lsystem_out) {
216             if (X *e=dynamic_cast<X*>(element_in)) {
217                 if(e->t > 0) {
218                     lsystem_out->push_back(new X(e->t - 1));
219                     return true;
220                 }
221             }
222             return false;
223         }
224         bool Capsella11::apply(Element* element_in,
225                                vector<Element*> *lsystem_out) {
226             if (X *e=dynamic_cast<X*>(element_in)) {
227                 if(e->t == 0) {
228
229                     RTLS_Material dead_flower_mat =
230                         RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
231                                       Vec4f(0.3, 0.3, 0.1, 1.),
232                                       Vec4f(0.3, 0.3, 0.1, 1.),
233                                       1.);
234
235                     lsystem_out->push_back(new Hat(a3));
236                     lsystem_out->push_back(new Push());
237                     lsystem_out->push_back(new
238                                           Material(dead_flower_mat));
239
240                     lsystem_out->push_back(new Push());
241                     lsystem_out->push_back(new Plus(a2));
242                     lsystem_out->push_back(new f());
```

```
243                     lsystem_out->push_back(new f());
244                     lsystem_out->push_back(new f());
245                     lsystem_out->push_back(new f());
246                     lsystem_out->push_back(new Minus(a2));
247                     lsystem_out->push_back(new Minus(a2));
248                     lsystem_out->push_back(new Push());
249                     lsystem_out->push_back(new f());
250                     lsystem_out->push_back(new f());
251                     lsystem_out->push_back(new f());
252                     lsystem_out->push_back(new Push());
253                     lsystem_out->push_back(new Minus(a2));
254                     lsystem_out->push_back(new Minus(a2));
255                     lsystem_out->push_back(new f());
256                     lsystem_out->push_back(new PushPolygon());
257                     lsystem_out->push_back(new Dot());
258                     lsystem_out->push_back(new Pop());
259                     lsystem_out->push_back(new Dot());
260                     lsystem_out->push_back(new Pop());
261                     lsystem_out->push_back(new Dot());
262                     lsystem_out->push_back(new Pop());
263                     lsystem_out->push_back(new Dot());
264                     lsystem_out->push_back(new Minus(a2));
265                     lsystem_out->push_back(new Minus(a2));
266                     lsystem_out->push_back(new f());
267                     lsystem_out->push_back(new f());
268                     lsystem_out->push_back(new f());
269                     lsystem_out->push_back(new f());
270                     lsystem_out->push_back(new Dot());
271                     lsystem_out->push_back(new Plus(a2));
272                     lsystem_out->push_back(new Plus(a2));
273                     lsystem_out->push_back(new f());
274                     lsystem_out->push_back(new f());
275                     lsystem_out->push_back(new f());
276                     lsystem_out->push_back(new Dot());
277                     lsystem_out->push_back(new Plus(a2));
278                     lsystem_out->push_back(new Plus(a2));
279                     lsystem_out->push_back(new f());
280                     lsystem_out->push_back(new Dot());
281                     lsystem_out->push_back(new PopPolygon());
282                     lsystem_out->push_back(new Pop());
283                     lsystem_out->push_back(new Percent());
284                     return true;
285                 }
286             }
287             return false;
288         }
289     }
```

25 Palm.h

```
1   #ifndef     _PALM_H_
2   #define     _PALM_H_
3
4   #include "Common.h"
5   #include "Rule.h"
6   #include "Element.h"
7   #include "Surface.h"
8
9   namespace Palm {
10
11      class PalmRule1     : public RTLS::Rule {
12      public:
13          PalmRule1(RTLS::Surface* _leaf) : Rule(),
14                                            leaf(_leaf) {}
15          bool apply(RTLS::Element* element_in,
16                     std::vector<RTLS::Element*> *lsystem_out);
17      private:
18          RTLS::Surface *leaf;
19
20          inline float degree2radian(float degree) {
21              return degree * M_PI / 180.0f;
22          }
23      };
24
25      class PalmRule2     : public RTLS::Rule {
26      public:
27          PalmRule2(RTLS::Surface* _leaf) : Rule(),
28                                            leaf(_leaf) {}
29          bool apply(RTLS::Element* element_in,
30                     std::vector<RTLS::Element*> *lsystem_out);
31      private:
32          RTLS::Surface *leaf;
33
34          inline float degree2radian(float degree) {
35              return degree * M_PI / 180.0f;
36          }
37      };
38
39      class PalmRule3     : public RTLS::Rule {
40      public:
41          PalmRule3()     : Rule() {}
42          bool apply(RTLS::Element* element_in,
43                     std::vector<RTLS::Element*> *lsystem_out);
44      };
45
46      class PalmRule4     : public RTLS::Rule {
47      public:
48          PalmRule4(RTLS::Material branch) : Rule(),
49                                             branchMat(branch) {}
50          bool apply(RTLS::Element* element_in,
51                     std::vector<RTLS::Element*> *lsystem_out);
52      private:
53          RTLS::Material branchMat;
54      };
55
56      class M : public RTLS::Element {
57      public:
58          M() : Element() {};
59          void execute(RTLS::Tree*    tree) {     };
```

```
60          void print();
61      };
62
63      class L : public RTLS::Element {
64      public:
65          L(float s, float a) : Element(), size(s),
66                                angle(a) {};
67          void execute(RTLS::Tree*    tree) {     };
68          void print();
69          float size, angle;
70      };
71
72      class H : public RTLS::Element {
73      public:
74          H(int b, float s) : Element(), size(s), branches(b)
75        {};
76          void execute(RTLS::Tree* tree) { };
77          void print();
78          float size;
79          int branches;
80      };
81
82      class A : public RTLS::Element {
83      public:
84          A(int h, int l, float s) : Element(), size(s),
85                                     height(h), leafs(l) {};
86          void execute(RTLS::Tree* tree) {  };
87          void print();
88          float size;
89          int height, leafs;
90      };
91  }
92  #endif
```

26 Palm..cpp

```
#include "Palm.h"

using namespace std;
using namespace RealTimeLSystem;

namespace Palm {

    const float a0 =  15.00f;
    const float a1 =  30.00f;
    const float a2 =  35.00f;
    const float a3 =   1.00f;
    const float a4 = 137.50f;
    const float r0 =   1.10f;
    const float r1 =   1.20f;
    const float r2 =   0.95f;
    const float s0 =   0.90f;
    const float e0 =   0.05f;
    const float e1 =   0.10f;
    const float l0 =   0.50f;

    void M::print () {
        cout << "M";
    }

    void L::print () {
        cout << "L(" << size << ")";
    }

    void H::print () {
        cout << "H(" << size << ")";
    }

    void A::print () {
        cout << "A(" << size << ")";
    }

    bool PalmRule1::apply(Element* element_in,
                          vector<Element*> *lsystem_out) {
        if (M *m=dynamic_cast<M*>(element_in)) {
            float gamma = atan(l0*2.0f/(e1-e0));
            lsystem_out->push_back(new Push());
            lsystem_out->push_back(new Plus(a1));
            lsystem_out->push_back(new f(e1 / sin(gamma +
                     degree2radian(90-a1)) * sin(gamma)));
            lsystem_out->push_back(new ApplySurface(leaf));
            lsystem_out->push_back(new Pop());
            lsystem_out->push_back(new Push());
            lsystem_out->push_back(new Minus(a1));
            lsystem_out->push_back(new f(e1 / sin(gamma +
                     degree2radian(90-a1)) * sin(gamma)));
            lsystem_out->push_back(new ApplySurface(leaf));
            lsystem_out->push_back(new Pop());

            lsystem_out->push_back(new Size(e0));
            lsystem_out->push_back(new F(l0));
            lsystem_out->push_back(new Size(0));
            lsystem_out->push_back(new F(0));

            lsystem_out->push_back(new Push());
```

```
60              lsystem_out->push_back(new Plus(a0));
61              lsystem_out->push_back(new ApplySurface(leaf));
62              lsystem_out->push_back(new Minus(2*a0));
63              lsystem_out->push_back(new ApplySurface(leaf));
64              lsystem_out->push_back(new Pop());
65
66              lsystem_out->push_back(new ApplySurface(leaf));
67
68              return true;
69          }
70          return false;
71      }
72
73      bool PalmRule2::apply(Element* element_in,
74                            vector<Element*> *lsystem_out) {
75          if (L *a=dynamic_cast<L*>(element_in)) {
76              float gamma = atan(l0*2.0f/(a->size*(1-r0)));
77              float x = sin(gamma + degree2radian(90-a2)) *
78                        sin(gamma);
79
80              lsystem_out->push_back(new L(a->size*r0,
81                                           a->angle*r1));
82              lsystem_out->push_back(new And(a->angle*r1));
83
84              lsystem_out->push_back(new Push());
85              lsystem_out->push_back(new Plus(a2));
86              lsystem_out->push_back(new f(0.5*a->size / x));
87              lsystem_out->push_back(new ApplySurface(leaf));
88              lsystem_out->push_back(new Pop());
89              lsystem_out->push_back(new Push());
90              lsystem_out->push_back(new Minus(a2));
91              lsystem_out->push_back(new f(0.5*a->size / x));
92              lsystem_out->push_back(new ApplySurface(leaf));
93              lsystem_out->push_back(new Pop());
94
95              lsystem_out->push_back(new Size(a->size*r0));
96              lsystem_out->push_back(new F(l0));
97
98              return true;
99          }
100         return false;
101     }
102
103     bool PalmRule3::apply(Element* element_in,
104                           vector<Element*> *lsystem_out) {
105         if (H *h=dynamic_cast<H*>(element_in)) {
106
107             lsystem_out->push_back(new Slash(a4));
108
109             lsystem_out->push_back(new PushLoD());
110
111             lsystem_out->push_back(new Push());
112             lsystem_out->push_back(new L(e0, a3));
113             lsystem_out->push_back(new M());
114             lsystem_out->push_back(new Pop());
115
116             if(h->branches-1 > 0) {
117
118                 lsystem_out->push_back(new Slash(a4));
119                 lsystem_out->push_back(new Push());
120                 lsystem_out->push_back(new L(e0, a3));
```

```
121                     lsystem_out->push_back(new M());
122                     lsystem_out->push_back(new Pop());
123
124                 }
125
126                 lsystem_out->push_back(new PopLoD());
127
128                 if(h->branches-1 > 0) {
129
130                     lsystem_out->push_back(new Slash(a4));
131
132                 }
133
134                 if(h->branches-2 > 0) {
135
136                     lsystem_out->push_back(new H(h->branches-2,
137                                                  h->size));
138                 }
139
140                 return true;
141             }
142             return false;
143         }
144
145         bool PalmRule4::apply(Element* element_in,
146                               vector<Element*> *lsystem_out) {
147             if (A *a=dynamic_cast<A*>(element_in)) {
148                 if(a->height > 0) {
149                     lsystem_out->push_back(
150                                         new Size(a->size*r2*s0));
151                     lsystem_out->push_back(new F(0));
152                     lsystem_out->push_back(
153                                            new Size(a->size*r2));
154                     lsystem_out->push_back(new F(1));
155                     lsystem_out->push_back(new A(a->height-1,
156                                                  a->leafs,
157                                                  a->size*r2));
158                 } else {
159                     lsystem_out->push_back(new PushLoD());
160                     lsystem_out->push_back(
161                                         new Material(branchMat));
162                     lsystem_out->push_back(
163                                 new Palm::H(a->leafs, a->size*r2));
164                     lsystem_out->push_back(new PopLoD());
165                 }
166                 return true;
167             }
168             return false;
169         }
170     }
```

27 Kildekode til test applikationer

Kildekoden til de 2 test applikationer følger her:

27.1 Viewer

Det program bruges til at vise en enkel model

27.1.1 Main.cpp

```
1   #include <iostream>
2
3   #include <GL/glew.h>
4   #include <GL/glut.h>
5
6   #include <IL/il.h>
7   #include <IL/ilu.h>
8
9   #include "Graphics/TrackBall.h"
10  #include "Graphics/DisplayText.h"
11
12  #include "Common/CommonDefs.h"
13
14  #include "CGLA/Mat4x4f.h"
15  #include "CGLA/Vec3f.h"
16  #include "CGLA/Vec4f.h"
17  #include "CGLA/Vec2i.h"
18
19  #include "X3DObject/X3DObject.h"
20
21  #include "../RTLsystem/Tree.h"
22  #include "../RTLsystem/Rule.h"
23  #include "../RTLsystem/RTLS_Material.h"
24
25  #include "../RTLsystem/Capsella.h"
26  #include "../RTLsystem/Basic.h"
27
28  #include "../RTLsystem/Surface.h"
29
30  #include "../RTLsystem/Bush.h"
31
32  #include "../RTLsystem/Palm.h"
33
34  using namespace std;
35  using namespace CGLA;
36  using namespace Graphics;
37  using namespace RTLS;
38
39  TrackBall *ball;
40  DisplayText displaytext;
41
42  GLuint tex;
43
44  int old_state=GLUT_UP;
45
46  bool paused = true;
47
48  Tree *tree1;
49
50  int draw_type = 0;
51
52  X3DObject *x3d_leaf, *x3d_flower;
53  X3DSurface *leaf_surface, *flower_surface;
```

```
54
55  int WINX=512, WINY=512;
56
57  GLfloat light_position[] = { 1.0, 0.0, 0.0, 0.0 };
58  GLfloat light_ambient[]  = { 0.2, 0.2, 0.2, 1.0 };
59  GLfloat light_diffuse[]  = { 0.3, 0.3, 0.3, 1.0 };
60
61  Vec3f pos = Vec3f(1,0,0);
62  bool forward = true;
63
64  bool drawAxis = false;
65
66  DWORD CLK_PER_FRAME;
67  LARGE_INTEGER last_time;
68  LARGE_INTEGER start_clk;
69  LARGE_INTEGER after_draw_clk;
70  DWORD diff;
71
72  int update = 0;
73
74  void axis();
75  GLuint loadTexture(char* filename);
76
77  void initGL(void) {
78
79        glMatrixMode(GL_PROJECTION);
80        glLoadIdentity();
81      glViewport(0,0,WINX,WINY);
82
83      gluPerspective(45.0, 1, 1.5, 1000);
84
85      glMatrixMode(GL_MODELVIEW);
86        glClearColor(1.0,1.0,1.0,1.0);
87        glShadeModel(GL_SMOOTH);
88        glEnable(GL_DEPTH_TEST);
89      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
90      glShadeModel(GL_SMOOTH);
91  }
92
93  void display() {
94
95        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
96
97
98      glMatrixMode(GL_MODELVIEW);
99
100       glLoadIdentity();
101
102       ball->do_spin();
103       ball->set_gl_modelview();
104
105     if(drawAxis) {
106         axis();
107     }
108
109       switch(draw_type) {
110         case 4:
111             tree1->drawBBox();
112         case 0:
113                   tree1->draw();
114                   break;
```

```
115           case 3:
116               tree1->drawBBox();
117               case 1:
118               glLightfv(GL_LIGHT0, GL_POSITION,
119                         light_position);
120               glEnable(GL_LIGHTING);
121               glEnable(GL_LIGHT0);
122                     tree1->drawWithoutTexture();
123               glDisable(GL_LIGHT0);
124               glDisable(GL_LIGHTING);
125                     break;
126               case 2:
127                     tree1->drawBBox();
128                     break;
129         }
130
131         displaytext.draw();
132
133       glutSwapBuffers();
134   }
135
136   void mouse(int button, int state, int x, int y) {
137         if (old_state==GLUT_UP && state==GLUT_DOWN) {
138               if (button==GLUT_LEFT_BUTTON)
139                   ball->grab_ball(ROTATE_ACTION,Vec2i(x,y));
140               else if (button==GLUT_MIDDLE_BUTTON)
141                     ball->grab_ball(ZOOM_ACTION,Vec2i(x,y));
142               else if (button==GLUT_RIGHT_BUTTON)
143                     ball->grab_ball(PAN_ACTION,Vec2i(x,y));
144         }
145         if (old_state==GLUT_DOWN && state==GLUT_UP)
146               ball->release_ball();
147         old_state=state;
148   }
149
150   void motion(int x, int y) {
151         if (old_state==GLUT_DOWN)
152               ball->roll_ball(Vec2i(x,y));
153   }
154
155   void reshape( int width, int height ) {
156       glViewport( 0, 0, width, height );
157         ball->set_screen_window(width,height);
158         ball->set_screen_centre(Vec2i(width/2,height/2));
159         glutPostRedisplay();
160   }
161
162   void keyboard(unsigned char key, int x, int y) {
163     switch(key) {
164       case 'Q':
165       case 'q':
166       case '\033': exit(0); break;
167       case 't':
168       case 'T':
169           draw_type = 0; break;
170         case 'g':
171       case 'G':
172           draw_type = 1; break;
173         case 'b':
174         case 'B':
175               draw_type = 2; break;
```

```
176         case 'h':
177           case 'H':
178                 draw_type = 3; break;
179         case 'y':
180           case 'Y':
181                 draw_type = 4; break;
182           case 'p':
183           case 'P':
184                 paused = !paused; break;
185           case 'a':
186           case 'A':
187             pos[2] -= 5; cout << "Distance to tree: "
188                               << pos.length() << endl; break;
189           case 'z':
190           case 'Z':
191                 pos[2] += 5; cout << "Distance to tree: "
192                               << pos.length() << endl; break;
193         case 'x':
194         case 'X':
195             drawAxis = !drawAxis; break;
196         case 'v':
197         case 'V':
198             tree1->printTiming();
199             break;
200         case 'm':
201         case 'M':
202             cout << " Number of polygons in model: "
203                  << tree1->getNumberOfPolygons() << endl;
204             break;
205         case 'N':
206         case 'n':
207             cout << "Number of sections drawn: "
208                  << tree1->getNumberOfSectionsDrawn() << endl;
209       }
210         if( key >= '0' && key <= '9') {
211             ball->reset();
212             if( key == '9') {
213                 ball->set_eye_dist(80);
214                 cout << "Eye dist: " << 80 << endl;
215             } else {
216                 ball->set_eye_dist((key - '0') * 5);
217                 cout << "Eye dist: " << (key - '0') * 5
218                      << endl;
219             }
220         }
221     }
222
223     void animate() {
224           QueryPerformanceCounter(&start_clk);
225
226         if((start_clk.QuadPart - last_time.QuadPart) >
227             CLK_PER_FRAME) {
228             if(!paused) {
229                     if(forward) {
230                           pos[2] -= 0.01;
231                           if(pos[2] <= 2.5f) {
232                                 forward = false;
233                           }
234                     } else {
235                           pos[2] += 0.01;
236                           if(pos[2] >= 22.5f) {
```

```
237                                 forward = true;
238                             }
239                         }
240                     }
241                 glutPostRedisplay();
242                 QueryPerformanceCounter(&last_time);
243             }
244             QueryPerformanceCounter(&after_draw_clk);
245
246             diff = CLK_PER_FRAME - (after_draw_clk.QuadPart –
247                                     start_clk.QuadPart);
248
249             if(diff < 0) {
250                 cout << "WARNING no time for update" << endl;
251                 return;
252             }
253
254             tree1->update(pos, diff);
255
256             QueryPerformanceCounter(&start_clk);
257             DWORD took = start_clk.QuadPart -
258                          after_draw_clk.QuadPart;
259         }
260
261         void init_il() {
262               // Initialize image loading library (DevIL)
263               ilInit();
264               iluInit();
265
266               // Enable conversion from palette to RGB
267               ilEnable(IL_CONV_PAL);
268         }
269
270         int main(int argc, char* argv[]) {
271
272               glutInit(&argc,argv);
273               glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE);
274               glutInitWindowSize(WINX,WINY);
275               glutCreateWindow("RealTimeLSystem");
276               glutShowWindow();
277
278             glutDisplayFunc(display);
279               glutKeyboardFunc(keyboard);
280               glutIdleFunc(animate);
281               glutMouseFunc(mouse);
282               glutReshapeFunc(reshape);
283               glutMotionFunc(motion);
284
285               initGL();
286
287             init_il();
288
289             tex = loadTexture(argv[1]);
290
291               int err = glewInit();
292               if (GLEW_OK != err) {
293                     cout << "GLEW Error: "
294                          << glewGetErrorString(err) << endl;
295                     return -1;
296               }
297
```

```
298         RTLS_Material tree_mat =
299               RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
300                             Vec4f(0.7, 0.5, 0.5, 1.),
301                             Vec4f(0.0, 0.0, 0.0, 1.), 10.);
302
303         tree1 = new Tree(Vec3f(1.0f,0.0f,0.0f),
304                          Vec4f(1,1,0,0));
305
306         x3d_leaf = new X3DObject("gfx/leaf2");
307         x3d_leaf->set_scale(0.2f);
308
309         leaf_surface = new X3DSurface(x3d_leaf);
310
311         x3d_flower = new X3DObject("gfx/rose");
312         x3d_flower->set_scale(0.1f);
313
314         flower_surface = new X3DSurface(x3d_flower);
315
316           tree1->addRule(new Basic::RuleA1());
317         tree1->addRule(new Basic::RuleA2(leaf_surface,
318                                          flower_surface));
319         tree1->addRule(new Basic::RuleA3(leaf_surface,
320                                          flower_surface));
321         tree1->addRule(new Basic::RuleA4(leaf_surface,
322                                          flower_surface));
323         tree1->addRule(new Basic::RuleA5(leaf_surface,
324                                          flower_surface));
325         tree1->addRule(new Basic::RuleA6(leaf_surface,
326                                          flower_surface));
327
328         tree1->pushBackInitElement(new
329                                    RTLS::Material(tree_mat));
330         tree1->pushBackInitElement(new RTLS::Texture(tex));
331         tree1->pushBackInitElement(new Color(0.8f,0.5f,0.5f));
332         tree1->pushBackInitElement(new Basic::A(1,0.5));
333
334         for(int i=0; i<10; i++) {
335             tree1->applyRules();
336         }
337
338         tree1->scale(Vec3f(0.5, 0.5, 0.5));
339
340         tree1->rotate(CGLA::ZAXIS, -M_PI/2.0f);
341
342         tree1->rotate(CGLA::XAXIS, 140*M_PI/180.0f);
343
344         ball = new TrackBall(Vec3f(0,1.5,0), 10, 10,
345                              WINX, WINY);
346
347           displaytext.addFramerate();
348
349         LARGE_INTEGER frequency;
350
351         QueryPerformanceFrequency(&frequency);
352
353         CLK_PER_FRAME = frequency.QuadPart / 60; //60 Hz
354
355         QueryPerformanceCounter(&last_time);
356
357         glutMainLoop();
358
```

```
359         return 0;
360   }
361   void axis (void) {
362         /*Plot part of axis and an auxiliary line*/
363         GLfloat v0[] = {0., 0., 0.};
364         GLfloat vx[]= {5., 0., 0.};
365         GLfloat vy[]= {0, 5., 0.};
366         GLfloat vz[]= {0., 0., 5.};
367
368         glPushAttrib(GL_CURRENT_BIT);
369         glColor3f(1., 0., 0.);
370         glBegin(GL_LINES);
371         glVertex3fv(v0);
372         glVertex3fv(vx);
373         for(int i=1; i<6; i++) {
374               glVertex3f(i,0.1,0);
375               glVertex3f(i,-0.1,0);
376         }
377         glEnd();
378
379         glColor3f(0., 1., 0.);
380         glBegin(GL_LINES);
381         glVertex3fv(v0);
382         glVertex3fv(vy);
383         for(int i=1; i<6; i++) {
384               glVertex3f(0.1,i,0);
385               glVertex3f(-0.1,i,0);
386         }
387         glEnd();
388
389         glColor3f(0., 0., 1.);
390         glBegin (GL_LINES);
391         glVertex3fv (v0);
392         glVertex3fv (vz);
393         for(int i=1; i<6; i++) {
394               glVertex3f(0.1,0,i);
395               glVertex3f(-0.1,0,i);
396         }
397         glEnd ();
398
399         glPopAttrib();
400   }
401
402   GLuint loadTexture(char* filename) {
403
404         ILuint  ImgId;
405         ilGenImages(1, &ImgId);
406         ilBindImage(ImgId);
407
408         if(!ilLoadImage(filename)) {
409               cout << "could not load " << filename << endl;
410               exit(0);
411         }
412         ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
413
414         int width = ilGetInteger(IL_IMAGE_WIDTH);
415         int height = ilGetInteger(IL_IMAGE_HEIGHT);
416
417         const unsigned char* image = ilGetData();
418
419         GLuint tex;
```

```
420
421         glGenTextures(1, &tex);
422         glBindTexture(GL_TEXTURE_2D, tex);
423
424         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
425                         GL_LINEAR);
426         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
427                         GL_LINEAR);
428
429       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
430                       GL_CLAMP_TO_EDGE);
431         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
432                         GL_CLAMP_TO_EDGE);
433
434         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
435                      height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
436                      image);
437
438       return tex;
439   }
```

27.2 TerrainRoamer

Applikation hvor der er sat alle modellerne ind I et terræn

27.2.1 Terrain.h

```
#ifndef __TERRAIN_H__
#define __TERRAIN_H__
#include <iostream>
#include <gl/glew.h>
#include <GL/glut.h>
#include <IL/il.h>
#include <IL/ilu.h>

#include "../RTLsystem/Tree.h"

#include "CGLA/Quaternion.h"
#include "CGLA/Vec3f.h"
#include "CGLA/Vec2f.h"

#define HEIGHT_MOD 0.05
#define WATER_HEIGHT 25

class Terrain {

public:
      Terrain(void);
      ~Terrain(void);

      void loadHeightMap(char*);
      void loadTexture(char*);
      void loadNoiseTexture(char*);
      void loadOceanTexture(char*);

      void createDisplayList();

      void draw();

      void addTree(RealTimeLSystem::Tree* tree);

      float getHeight(int, int);
      float getHeight(int);

    void update(CGLA::Vec3f looking_from, DWORD time);

      // Id of texture map
      GLuint my_tex, my_noiseTex, my_oceanTex;

      int hm_x, hm_y, hm_bpp;
      unsigned char* heightMap;

      int noise_x, noise_y;
      int ocean_x, ocean_y;
private:

    bool isNew;
    std::vector<RealTimeLSystem::Tree*>::iterator
          tree_iter;

      GLuint displList;
      std::vector<RealTimeLSystem::Tree*> trees;
};
#endif
```

27.2.2 Terrain.cpp

```
1   #include "Terrain.h"
2
3   using namespace std;
4   using namespace CGLA;
5
6   Terrain::Terrain() {
7       isNew = true;
8   }
9
10  Terrain::~Terrain(void) {
11        glDeleteLists(dispList, 1);
12  }
13
14  void Terrain::loadHeightMap(char* filename) {
15        glPushAttrib(GL_ALL_ATTRIB_BITS);
16      // Create and bind an image.
17        ILuint ImgId;
18        ilGenImages(1, &ImgId);
19        ilBindImage(ImgId);
20
21        // Load the image - abort program if it fails.
22        if(!ilLoadImage(filename)) {
23              cout << "could not load " << filename << endl;
24              exit(0);
25        }
26        ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE);
27
28        // Get dimensions of image.
29        hm_x = ilGetInteger(IL_IMAGE_WIDTH);
30        hm_y = ilGetInteger(IL_IMAGE_HEIGHT);
31        hm_bpp = ilGetInteger(IL_IMAGE_BYTES_PER_PIXEL);
32
33  //Set a pointer to point to the first byte of the image.
34        heightMap = ilGetData();
35      glPopAttrib();
36  }
37
38  void Terrain::loadNoiseTexture(char* filename) {
39        glPushAttrib(GL_ALL_ATTRIB_BITS);
40        // Create and bind an image.
41        ILuint  ImgId;
42        ilGenImages(1, &ImgId);
43        ilBindImage(ImgId);
44
45        // Load the image - abort program if it fails.
46        if(!ilLoadImage(filename)) {
47              cout << "could not load " << filename << endl;
48              exit(0);
49        }
50        ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE);
51
52        // Get dimensions of image.
53        noise_x = ilGetInteger(IL_IMAGE_WIDTH);
54        noise_y = ilGetInteger(IL_IMAGE_HEIGHT);
55  //Set a pointer to point to the first byte of the image.
56        const unsigned char* image = ilGetData();
57
58        // --------------------------------------
59        // Use the image as an OpenGL texture
```

```
60          // --------------------------------------
61
62          //Create and bind a texture name
63          glGenTextures(1, &my_noiseTex);
64          glBindTexture(GL_TEXTURE_2D, my_noiseTex);
65
66          // Setup the texture interpolation
67          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
68                          GL_LINEAR);
69          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
70
71          GL_LINEAR_MIPMAP_LINEAR);
72
73          // Specify how we deal with wrapping.
74          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
75                          GL_REPEAT);
76          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
77                          GL_REPEAT);
78
79          // Build a pyramid of texture images.
80          gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
81
82          noise_x, noise_y,
83
84          GL_RGB, GL_UNSIGNED_BYTE,
85
86          image);
87
88      glPopAttrib();
89  }
90
91  void Terrain::loadOceanTexture(char* filename) {
92          glPushAttrib(GL_ALL_ATTRIB_BITS);
93          // Create and bind an image.
94          ILuint  ImgId;
95          ilGenImages(1, &ImgId);
96          ilBindImage(ImgId);
97
98          // Load the image - abort program if it fails.
99          if(!ilLoadImage(filename)) {
100               cout << "could not load " << filename << endl;
101               exit(0);
102         }
103         ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE);
104
105         // Get dimensions of image.
106         ocean_x = ilGetInteger(IL_IMAGE_WIDTH);
107         ocean_y = ilGetInteger(IL_IMAGE_HEIGHT);
108
109 //Set a pointer to point to the first byte of the image.
110         const unsigned char* image = ilGetData();
111
112         // --------------------------------------
113         // Use the image as an OpenGL texture
114         // --------------------------------------
115
116         //Create and bind a texture name
117         glGenTextures(1, &my_oceanTex);
118         glBindTexture(GL_TEXTURE_2D, my_oceanTex);
119
120         // Setup the texture interpolation
```

```
121         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
122                         GL_LINEAR);
123         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
124
125         GL_LINEAR_MIPMAP_LINEAR);
126
127         // Specify how we deal with wrapping.
128         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
129                         GL_REPEAT);
130         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
131                         GL_REPEAT);
132
133         // Build a pyramid of texture images.
134         gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
135
136         ocean_x, ocean_y,
137
138         GL_RGB, GL_UNSIGNED_BYTE,
139
140         image);
141       glPopAttrib();
142   }
143
144   void Terrain::loadTexture(char* filename) {
145         glPushAttrib(GL_ALL_ATTRIB_BITS);
146       // Create and bind an image.
147         ILuint  ImgId;
148         ilGenImages(1, &ImgId);
149         ilBindImage(ImgId);
150
151         // Load the image - abort program if it fails.
152         if(!ilLoadImage(filename)) {
153               cout << "could not load " << filename << endl;
154               exit(0);
155         }
156         ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE);
157
158         // Get dimensions of image.
159         int size_x = ilGetInteger(IL_IMAGE_WIDTH);
160         int size_y = ilGetInteger(IL_IMAGE_HEIGHT);
161         int bytes_per_pixel =
162                     ilGetInteger(IL_IMAGE_BYTES_PER_PIXEL);
163
164   //Set a pointer to point to the first byte of the image.
165         const unsigned char* image = ilGetData();
166
167         // --------------------------------------
168         // Use the image as an OpenGL texture
169         // --------------------------------------
170
171         //Create and bind a texture name
172         glGenTextures(1, &my_tex);
173         glBindTexture(GL_TEXTURE_2D, my_tex);
174
175         // Setup the texture interpolation
176         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
177                         GL_LINEAR);
178         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
179                         GL_LINEAR_MIPMAP_LINEAR);
180
181         // Specify how we deal with wrapping.
```

```
182         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
183                         GL_CLAMP_TO_EDGE);
184         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
185                         GL_CLAMP_TO_EDGE);
186
187         // Build a pyramid of texture images.
188         gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
189
190         size_x, size_y,
191
192         GL_RGB, GL_UNSIGNED_BYTE,
193
194         image);
195       glPopAttrib();
196   }
197
198   void Terrain::createDisplayList() {
199
200       /* Multi texturing */
201         glActiveTextureARB(GL_TEXTURE2_ARB);
202         glBindTexture(GL_TEXTURE_2D, my_oceanTex);
203         glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
204                   GL_MODULATE);
205         glDisable(GL_TEXTURE_2D);
206
207         glActiveTextureARB(GL_TEXTURE1_ARB);
208         glBindTexture(GL_TEXTURE_2D, my_noiseTex);
209         glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
210                   GL_COMBINE_EXT);
211         glDisable(GL_TEXTURE_2D);
212
213         glActiveTextureARB(GL_TEXTURE0_ARB);
214         glBindTexture(GL_TEXTURE_2D, my_tex);
215         glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
216                   GL_MODULATE);
217         glDisable(GL_TEXTURE_2D);
218
219         dispList = glGenLists(1);
220         glNewList(dispList, GL_COMPILE);
221
222         glActiveTextureARB(GL_TEXTURE1_ARB);
223         glEnable(GL_TEXTURE_2D);
224         glActiveTextureARB(GL_TEXTURE0_ARB);
225         glEnable(GL_TEXTURE_2D);
226
227         for(int y=0; y<hm_y-1; y++) {
228               glBegin(GL_TRIANGLE_STRIP);
229               for(int x=0; x<hm_x; x++) {
230                     glNormal3f((getHeight(x-1,y+1)-
231                      getHeight(x+1,y+1))/2., (getHeight(x,y)-
232                      getHeight(x,y+2))/2., 1);
233                     glMultiTexCoord2fARB(GL_TEXTURE0_ARB,
234                             x/(float)hm_x,(y+1)/(float)hm_y);
235                     glMultiTexCoord2fARB(GL_TEXTURE1_ARB,
236                       64*x/(float)noise_x,64*(y+1)/(float)noise_y);
237                     glVertex3f(x,y+1,getHeight(x,y+1));
238
239                     glNormal3f((getHeight(x-1,y)-
240                      getHeight(x+1,y))/2., (getHeight(x,y-1)-
241                      getHeight(x,y+1))/2., 1);
242                     glMultiTexCoord2fARB(GL_TEXTURE0_ARB,
```

```
243                                   x/(float)hm_x,y/(float)hm_y);
244                         glMultiTexCoord2fARB(GL_TEXTURE1_ARB,
245                         64*x/(float)noise_x,64*y/(float)noise_y);
246                         glVertex3f(x,y,getHeight(x,y));
247                   }
248                   glEnd();
249             }
250
251           //Disable textures
252             glDisable(GL_TEXTURE_2D);
253             glActiveTextureARB(GL_TEXTURE1_ARB);
254             glDisable(GL_TEXTURE_2D);
255
256           //water
257             glActiveTextureARB(GL_TEXTURE2_ARB);
258             glEnable(GL_TEXTURE_2D);
259
260             glEnable(GL_BLEND);
261             glDepthMask(GL_FALSE);
262
263             glBlendFunc(GL_SRC_ALPHA, GL_ONE);
264
265             glBegin(GL_QUADS);
266                   glNormal3f(0,0,1);
267
268                   glMultiTexCoord2fARB(GL_TEXTURE2_ARB, 0., 0.);
269                   glVertex3f(0,0,HEIGHT_MOD*WATER_HEIGHT);
270
271                   glMultiTexCoord2fARB(GL_TEXTURE2_ARB, 64., 0.);
272                   glVertex3f(hm_x,0,HEIGHT_MOD*WATER_HEIGHT);
273
274                   glMultiTexCoord2fARB(GL_TEXTURE2_ARB, 64, 64);
275                   glVertex3f(hm_x,hm_y,HEIGHT_MOD*WATER_HEIGHT);
276
277                   glMultiTexCoord2fARB(GL_TEXTURE2_ARB, 0., 64.);
278                   glVertex3f(0,hm_y,HEIGHT_MOD*WATER_HEIGHT);
279
280             glEnd();
281
282             glDepthMask(GL_TRUE);
283             glDisable(GL_BLEND);
284
285             glDisable(GL_TEXTURE_2D);
286
287             glEndList();
288
289       }
290
291       void Terrain::draw() {
292             glPushMatrix();
293             glPushAttrib(GL_ALL_ATTRIB_BITS);
294
295           glDisable(GL_LIGHTING);
296
297           for(std::vector<RealTimeLSystem::Tree*>::iterator iter
298                   = trees.begin(); iter != trees.end(); iter++) {
299               (*iter)->draw();
300           }
301
302           glEnable(GL_LIGHTING);
303
```

```
304         glPopAttrib();
305           glPopMatrix();
306
307         glPushMatrix();
308           glPushAttrib(GL_ALL_ATTRIB_BITS);
309           glCallList(dispList);
310         glPopAttrib();
311           glPopMatrix();
312     }
313
314     void Terrain::update(Vec3f looking_from, DWORD time) {
315
316         LARGE_INTEGER start_clk;
317         LARGE_INTEGER after_update_clk;
318
319         QueryPerformanceCounter(&start_clk);
320
321         if(isNew && trees.size() > 0) {
322             tree_iter = trees.begin();
323             isNew = false;
324         } else if(trees.size() == 0) {
325             return;
326         }
327
328         QueryPerformanceCounter(&after_update_clk);
329         int t = 0;
330         while(after_update_clk.QuadPart -  start_clk.QuadPart <
331               time && t < trees.size()) {
332
333             if(tree_iter == trees.end()) {
334                 tree_iter = trees.begin();
335             }
336
337             (*tree_iter)->update(looking_from, time –
338                                  (after_update_clk.QuadPart -
339                                  start_clk.QuadPart));
340
341             tree_iter++;
342             t++;
343
344             QueryPerformanceCounter(&after_update_clk);
345         }
346     }
347
348     float Terrain::getHeight(int x, int y) {
349           if(x<0 || x>=hm_x)
350                 return 0;
351           if(y<0 || y>=hm_y)
352                 return 0;
353           return HEIGHT_MOD*int(heightMap[(x+y*hm_x)*hm_bpp]);
354     }
355
356     float Terrain::getHeight(int x) {
357           return HEIGHT_MOD*int(heightMap[x*hm_bpp]);
358     }
359
360     void Terrain::addTree(RealTimeLSystem::Tree* tree) {
361           trees.push_back(tree);
362     }
```

27.2.3 Navigator.h

```
1   #pragma once
2   #include <iostream>
3   #include <GL/glew.h>
4
5   #include "CGLA/Vec2f.h"
6   #include "CGLA/Vec3f.h"
7
8   #include "Terrain.h"
9
10  #define SPEED 0.2
11  #define ROT_SPEED 0.05
12
13  class Navigator {
14
15  public:
16        Navigator(Terrain*, float, float);
17
18        void left(bool pressed);
19        void right(bool pressed);
20        void up(bool pressed);
21        void down(bool pressed);
22        void shift(bool pressed);
23        void ctrl(bool pressed);
24
25        void lookAt();
26        void setPerspective();
27
28        void draw(bool); //draw the person
29
30        void camMode(int);//1:1st person,2: 3rd person,3: god
31        void godMode(int); //1: pan, 2: rotate, 3: zoom;
32
33        void setWindowSize(float x, float y) {
34                                          WINX = x; WINY = y;};
35
36        void move(int, int);
37        void mousePressed(int x, int y) { mouseDown = true;
38                                    last_x = x; last_y = y;};
39        void mouseReleased() { mouseDown = false; };
40        void advance();
41        void print();
42
43        float getZ() const;
44        float getZ(const CGLA::Vec2f&) const;
45        int zoom;
46
47      CGLA::Vec2f pos, delta, cam_delta;
48  private:
49        Terrain* terrain;
50        CGLA::Vec3f god_pos, god_delta;
51        bool down_key, left_key, right_key, up_key,
52             shift_key, ctrl_key;
53        int cam_mode, god_mode;
54        int last_x, last_y; //last mouse position
55        bool mouseDown;
56        float WINX, WINY;
57  };
```

27.2.4 Navigator.cpp

```
#include "Navigator.h"

using namespace std;
using namespace CGLA;

Navigator::Navigator(Terrain* t, float winx, float winy) {
      terrain = t;
      pos = Vec2f(127.,127.);
      delta = Vec2f(1.,0.);
      cam_delta = Vec2f(1.,0.);
      god_pos = Vec3f(127.,127.,10);
      god_delta = Vec3f(1.,0.,0.);
      cam_mode = 1;
      god_mode = 1;
      zoom = 53;
      WINX = winx;
      WINY = winy;
    down_key = false;
    left_key = false;
    right_key = false;
    up_key = false;
    shift_key = false;
    ctrl_key = false;
}

float Navigator::getZ() const {
      float fractionx = pos[0] - (int)pos[0];
      float fractiony = pos[1] - (int)pos[1];

      return (fractionx * (fractiony * terrain->
        getHeight((int)floor(pos[0]),(int)ceil(pos[1])) +
      (1-fractiony) * terrain->
        getHeight((int)floor(pos[0]),(int)floor(pos[1]))) +
      (1-fractionx) * (fractiony * terrain->
        getHeight((int)ceil(pos[0]),(int)ceil(pos[1])) +
      (1-fractiony) * terrain->
        getHeight((int)ceil(pos[0]),(int)floor(pos[1]))));
}

float Navigator::getZ(const Vec2f& vec_pos) const {
      float fractionx = vec_pos[0] - (int)vec_pos[0];
      float fractiony = vec_pos[1] - (int)vec_pos[1];

      return (fractionx * (fractiony * terrain->
  getHeight((int)floor(vec_pos[0]),(int)ceil(vec_pos[1])) +
(1-fractiony) * terrain->
  getHeight((int)floor(vec_pos[0]),(int)floor(vec_pos[1])))
      +
(1-fractionx) * (fractiony * terrain->
  getHeight((int)ceil(vec_pos[0]),(int)ceil(vec_pos[1])) +
(1-fractiony) * terrain->
 getHeight((int)ceil(vec_pos[0]),(int)floor(vec_pos[1]))));
}

void Navigator::ctrl(bool pressed) {
      ctrl_key = pressed;
}
void Navigator::shift(bool pressed) {
      shift_key = pressed;
```

```
60  }
61  void Navigator::up(bool pressed) {
62        up_key = pressed;
63  }
64  void Navigator::down(bool pressed) {
65        down_key = pressed;
66  }
67  void Navigator::left(bool pressed) {
68        left_key = pressed;
69  }
70  void Navigator::right(bool pressed) {
71        right_key = pressed;
72  }
73
74  void Navigator::move(int x, int y) {  //mouse
75        if(cam_mode == 3) { //only in gode mode
76              double v;
77              int dx = x - last_x;
78              int dy = y - last_y;
79              switch(god_mode) {
80              case 1://pan
81                    god_pos[0] += 0.5*dx*god_delta[1];
82                    god_pos[1] -= 0.5*dx*god_delta[0];
83
84                    god_pos[2] -= 0.5*dy;
85                    break;
86              case 2://rotate
87                    god_delta -= 0.01*dx*
88                     Vec3f(-1*god_delta[1], god_delta[0], 0);
89                    god_delta[2] -= 0.01*dy;
90                    god_delta = normalize(god_delta);
91                    break;
92              case 3://zoom
93                    zoom += dy;
94                    if(zoom > 90)
95                          zoom = 90;
96                    if(zoom < 1)
97                          zoom = 1;
98                    break;
99              }
100             last_x = x;
101             last_y = y;
102       }
103 }
104
105 void Navigator::lookAt() {
106       switch(cam_mode) {
107             case 3: //god is watching!
108
109       gluLookAt(god_pos[0],god_pos[1],god_pos[2],
110
111       god_pos[0]+god_delta[0],god_pos[1]+god_delta[1],
112               god_pos[2]+god_delta[2], 0,0,1);
113                   break;
114             case 2: //3rd person
115                   gluLookAt(pos[0]-2*cam_delta[0],
116                         pos[1]-2*cam_delta[1],getZ()+0.5,
117                         pos[0],pos[1],getZ(), 0,0,1);
118                   break;
119             case 1: //1st person
120       // gluLookAt is used to specify viewing parameters.
```

```
121         gluLookAt(pos[0],pos[1],getZ()+3, // position of eye.
122         pos[0]+delta[0],pos[1]+delta[1],getZ()+3,  // centre
123                                       //(the point we look at)
124         0,0,1); // the general direction of up.
125                     break;
126         }
127   }
128
129   void Navigator::setPerspective() {
130         if(cam_mode==3)
131               gluPerspective(zoom,float(WINX)/WINY,.1,256);
132         else
133               gluPerspective(53,float(WINX)/WINY,.1,256);
134   }
135
136   void Navigator::advance() {
137         if(cam_mode != 3) {
138               Vec2f tmp_pos = pos;
139               Vec2f tmp_delta = delta;
140               if(up_key)
141                     tmp_pos += (shift_key?2:1)*SPEED*delta;
142               if(down_key)
143                     tmp_pos -= (shift_key?2:1)*SPEED*delta;
144               if(ctrl_key) { //strafe
145                     if(left_key)
146                           tmp_pos +=
147                      (shift_key?2:1)*SPEED*orthogonal(delta);
148                     if(right_key)
149                           tmp_pos -=
150                      (shift_key?2:1)*SPEED*orthogonal(delta);
151                     } else { //rotation
152                     if(left_key) {
153                           tmp_delta +=
154                       ROT_SPEED*orthogonal(delta);
155                           tmp_delta = normalize(tmp_delta);
156                     }
157                     if(right_key) {
158                           tmp_delta -=
159                      ROT_SPEED*orthogonal(delta);
160                           tmp_delta = normalize(tmp_delta);
161                     }
162               }
163               pos = tmp_pos;
164               cam_delta = delta;
165               delta = tmp_delta;
166         } else { //god mode
167               if(up_key)
168                     god_pos +=
169               (shift_key?2:1)*5*SPEED*Vec3f(god_delta[0],
170                                             god_delta[1], 0);
171               if(down_key)
172                     god_pos -=
173               (shift_key?2:1)*5*SPEED*Vec3f(god_delta[0],
174                                             god_delta[1], 0);
175               if(ctrl_key) { //strafe
176                     if(left_key)
177                           god_pos +=
178                (shift_key?2:1)*5*SPEED*Vec3f(-1*god_delta[1],
179                                             god_delta[0], 0);
180                     if(right_key)
181                           god_pos -=
```

```
182                (shift_key?2:1)*5*SPEED*Vec3f(-1*god_delta[1],
183                                              god_delta[0], 0);
184                      } else { //rotation
185                      if(left_key) {
186                            god_delta += ROT_SPEED*Vec3f(-
187                             1*god_delta[1], god_delta[0], 0);
188                            god_delta = normalize(god_delta);
189                      }
190                      if(right_key) {
191                            god_delta -= ROT_SPEED*Vec3f(-
192                             1*god_delta[1], god_delta[0], 0);
193                            god_delta = normalize(god_delta);
194                      }
195                }
196          }
197    }
198
199    void Navigator::camMode(int mode) {
200          cam_mode = mode;
201    }
202
203    void Navigator::godMode(int mode) {
204          god_mode = mode;
205    }
206
207    void Navigator::print() {
208          float fractionx = pos[0] - (int)pos[0];
209        float fractiony = pos[1] - (int)pos[1];
210
211          cout << "Positions: avatar:" << pos << " god: " <<
212          god_pos << endl;
213          cout << "Deltas: avatar" << delta << " god: " <<
214          god_delta << endl;
215          cout << "Height in the 4 points: " << endl;
216          cout << "C: " << terrain->
217          getHeight((int)floor(pos[0]),(int)ceil(pos[1]))
218          << " A: " << terrain->
219          getHeight((int)floor(pos[0]),(int)floor(pos[1]))
220          << endl;
221          cout << "D: " << terrain->
222          getHeight((int)ceil(pos[0]),(int)ceil(pos[1]))
223          << " B: " << terrain->
224          getHeight((int)ceil(pos[0]),(int)floor(pos[1]))
225          << endl;
226          cout << "Fractions: " << fractionx << ", " <<
227                   fractiony << endl;
228
229          float height = (fractionx * (fractiony * terrain>
230            getHeight((int)floor(pos[0]),(int)ceil(pos[1])) +
231          (1-fractiony) * terrain->
232            getHeight((int)floor(pos[0]),(int)floor(pos[1]))) +
233          (1-fractionx) * (fractiony * terrain->
234            getHeight((int)ceil(pos[0]),(int)ceil(pos[1])) +
235          (1-fractiony) * terrain->
236            getHeight((int)ceil(pos[0]),(int)floor(pos[1]))));
237          cout << "Height: " << height << endl;
238
239          cout << endl;
240    }
```

27.2.5 Main.cpp

```
1   #include <iostream>
2
3   #include <GL/glew.h>
4   #include <GL/glut.h>
5
6   #include <IL/il.h>
7   #include <IL/ilu.h>
8
9   #include "Graphics/DisplayText.h"
10
11  #include "Common/CommonDefs.h"
12
13  #include "X3DObject/X3DObject.h"
14
15  #include "Terrain.h"
16  #include "Navigator.h"
17
18  #include "../RTLsystem/Tree.h"
19  #include "../RTLsystem/Rule.h"
20
21  #include "../RTLsystem/Capsella.h"
22  #include "../RTLsystem/Basic.h"
23  #include "../RTLsystem/Palm.h"
24  #include "../RTLsystem/Bush.h"
25
26  #include "../RTLsystem/Surface.h"
27
28  #include "../RTLsystem/RTLS_Material.h"
29
30  using namespace std;
31  using namespace CGLA;
32  using namespace RealTimeLSystem;
33
34  DisplayText disp;
35
36  int main_window;
37
38  int WINX = 512, storedWINX = 512;
39  int WINY = 512, storedWINY = 512;
40
41  X3DObject *x3d_leaf, *x3d_flower, *x3d_palm_leaf;
42  X3DSurface *leaf_surface, *flower_surface,
43  *palm_leaf_surface;
44
45  GLuint bark_tex;
46
47  GLfloat light_position[] = { 0., 0., 1., 0. };
48  bool selection_mode = false;
49  int selected_obj = 0;
50
51  Terrain* terrain = new Terrain();
52  Navigator* navigator = new Navigator(terrain, WINX, WINY);
53
54  #define TREES 50
55  #define BUSHES 50
56  #define PALMS 20
57  #define CAPSELLAS 50
58
59  std::vector<Tree*> trees;
```

```
60
61  Tree *tree;
62
63  DWORD CLK_PER_FRAME;
64  LARGE_INTEGER last_time;
65  LARGE_INTEGER start_clk;
66  LARGE_INTEGER after_draw_clk;
67  DWORD diff;
68
69  GLuint loadTexture(char* filename);
70
71  // ----------------------------------------
72  // Function that loads an image from disk and
73  // then creates a texture from the image
74  // ----------------------------------------
75  void init_il() {
76        // Initialize image loading library (DevIL)
77        ilInit();
78        iluInit();
79
80        // Enable conversion from palette to RGB
81        ilEnable(IL_CONV_PAL);
82  }
83
84  void init_gl(int w, int h) {
85        glEnable(GL_CULL_FACE);
86        glCullFace(GL_BACK);
87        glEnable(GL_LIGHTING);
88        glEnable(GL_LIGHT0);
89
90        //glEnable(GL_NORMALIZE);
91
92        // We switch to projection matrix model
93        glMatrixMode(GL_PROJECTION);
94
95        // Clear projection matrix.
96        glLoadIdentity();
97
98        // Specify a perspective projection
99        gluPerspective(navigator->zoom,float(w)/h,.1,256);
100
101       // Go back to modelview mode.
102       glMatrixMode(GL_MODELVIEW);
103 }
104
105 void initTrees() {
106       float x,y,z;
107
108     bark_tex = loadTexture("gfx/bark7.jpg");
109
110     x3d_leaf = new X3DObject("gfx/leaf2");
111     x3d_leaf->set_scale(0.1f);
112
113     leaf_surface = new X3DSurface(x3d_leaf);
114
115     x3d_flower = new X3DObject("gfx/rose");
116     x3d_flower->set_scale(0.1f);
117
118     flower_surface = new X3DSurface(x3d_flower);
119
120     RTLS_Material tree_mat =
```

```
121             RTLS_Material(Vec4f(0.9, 0.7, 0.7, 1.),
122                           Vec4f(0.7, 0.5, 0.5, 1.),
123                           Vec4f(0.0, 0.0, 0.0, 1.), 10.);
124
125         for(int i=0; i<TREES; i++) {
126
127               do {
128                     x = rand() / float(RAND_MAX) * 255.0f;
129                     y = rand() / float(RAND_MAX) * 255.0f;
130                     z = navigator->getZ(Vec2f(x,y));
131
132               } while(z < HEIGHT_MOD*WATER_HEIGHT || x > 90
133                      && y < 120);
134
135
136           tree = new Tree(Vec3f(1.0f, 0.0f, 0.0f),
137                           Vec4f(1,0,0,0));
138
139           tree->addRule(new Basic::RuleA1());
140           tree->addRule(new Basic::RuleA2(leaf_surface,
141                                           flower_surface));
142           tree->addRule(new Basic::RuleA3(leaf_surface,
143                                           flower_surface));
144           tree->addRule(new Basic::RuleA4(leaf_surface,
145                                           flower_surface));
146           tree->addRule(new Basic::RuleA5(leaf_surface,
147                                           flower_surface));
148           tree->addRule(new Basic::RuleA6(leaf_surface,
149                                           flower_surface));
150
151           tree->pushBackInitElement(new
152                       RealTimeLSystem::Material(tree_mat));
153           tree->pushBackInitElement(new
154                        RealTimeLSystem::Texture(bark_tex));
155             tree->pushBackInitElement(new
156                                       Color(0.8f,0.5f,0.5f));
157             tree->pushBackInitElement(new Basic::A(1,0.5));
158
159             for(int i=0; i<10; i++) {
160               tree->applyRules();
161           }
162
163           tree->setPosition(Vec3f(x,y,z));
164           tree->rotate(CGLA::YAXIS, M_PI/2.0f);
165
166           trees.push_back(tree);
167
168           terrain->addTree(tree);
169         }
170
171       RTLS_Material palm_trunk_mat =
172                 RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
173                               Vec4f(0.5, 0.4, 0.3, 1.),
174                               Vec4f(0.6, 0.5, 0.4, 1.), 10.);
175
176       RTLS_Material palm_branch_mat =
177                 RTLS_Material(Vec4f(0.0, 0.1, 0.0, 1.),
178                               Vec4f(0.0, 0.5, 0.0, 1.),
179                               Vec4f(0.0, 0.5, 0.0, 1.), 10.);
180
181       x3d_palm_leaf = new X3DObject("gfx/palm_leaf");
```

```
182         palm_leaf_surface = new X3DSurface(x3d_palm_leaf);
183
184         for(int i=0; i<PALMS; i++) {
185             do {
186                 x = 160.0f + rand() / float(RAND_MAX) * 50.0f;
187                 y = 50.0f + rand() / float(RAND_MAX) * 30.0f;
188                 z = navigator->getZ(Vec2f(x,y));
189
190             } while(z < HEIGHT_MOD*WATER_HEIGHT);
191
192             tree = new Tree(Vec3f(1.0f,0.0f,0.0f),
193                             Vec4f(1,0,0,0));
194
195             tree->addRule(new
196                           Palm::PalmRule1(palm_leaf_surface));
197             tree->addRule(new
198                           Palm::PalmRule2(palm_leaf_surface));
199             tree->addRule(new Palm::PalmRule3());
200             tree->addRule(new
201                           Palm::PalmRule4(palm_branch_mat));
202
203             tree->pushBackInitElement(new
204                     RealTimeLSystem::Material(palm_trunk_mat));
205             tree->pushBackInitElement(new Palm::A(15, 20,1));
206
207             for(int i=0; i<35; i++) {
208                 tree->applyRules();
209             }
210
211             tree->setPosition(Vec3f(x,y,z));
212             tree->rotate(CGLA::YAXIS, M_PI/2.0f);
213             tree->scale(Vec3f(.4, .4, .4));
214
215             trees.push_back(tree);
216
217              terrain->addTree(tree);
218         }
219
220         /* Capsella */
221
222         RTLS_Material capsella_mat =
223                  RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
224                                Vec4f(0.0, 0.8, 0.0, 1.),
225                                Vec4f(0.0, 0.8, 0.0, 1.), 1.);
226
227         for(int i=0; i<CAPSELLAS; i++) {
228
229             do {
230                 x = rand() / float(RAND_MAX) * 255.0f;
231                 y = rand() / float(RAND_MAX) * 255.0f;
232                 z = navigator->getZ(Vec2f(x,y));
233
234             } while(z < HEIGHT_MOD*WATER_HEIGHT ||
235                     x > 90 && y < 120);
236
237             tree = new Tree(Vec3f(1.0f,0.0f,0.0f),
238                             Vec4f(1,0,0,0));
239
240             tree->addRule(new Capsella::Capsella1());
241             tree->addRule(new Capsella::Capsella2());
242             tree->addRule(new Capsella::Capsella3());
```

```
243             tree->addRule(new Capsella::Capsella4());
244             tree->addRule(new Capsella::Capsella5());
245             tree->addRule(new Capsella::Capsella6());
246             tree->addRule(new Capsella::Capsella7());
247             tree->addRule(new Capsella::Capsella8());
248             tree->addRule(new Capsella::Capsella9());
249             tree->addRule(new Capsella::Capsella10());
250             tree->addRule(new Capsella::Capsella11());
251
252             tree->pushBackInitElement(new Size(1));
253             tree->pushBackInitElement(new
254                    RealTimeLSystem::Material(capsella_mat));
255             tree->pushBackInitElement(new Capsella::I(9));
256             tree->pushBackInitElement(new Capsella::a(5));
257
258             for(int i=0; i<16; i++) {
259                 tree->applyRules();
260             }
261
262             tree->setPosition(Vec3f(x,y,z));
263                 tree->rotate(CGLA::YAXIS, M_PI/2.0f);
264             tree->scale(Vec3f(.005, .005, .005));
265
266                 trees.push_back(tree);
267
268                 terrain->addTree(tree);
269         }
270
271         /* bush */
272
273         RTLS_Material bush_mat =
274                   RTLS_Material(Vec4f(0.0, 0.0, 0.0, 1.),
275                                 Vec4f(0.0, 0.6, 0.0, 1.),
276                                 Vec4f(0.0, 0.4, 0.0, 1.), 10.);
277
278         for(int i=0; i<BUSHES; i++) {
279
280             do {
281                 x = rand() / float(RAND_MAX) * 255.0f;
282                 y = rand() / float(RAND_MAX) * 255.0f;
283                 z = navigator->getZ(Vec2f(x,y));
284
285             } while(z < HEIGHT_MOD*WATER_HEIGHT ||
286                     x > 90 && y < 120);
287
288             tree = new Tree(Vec3f(1.0f,0.0f,0.0f),
289                             Vec4f(1,0,0,0));
290
291             tree->addRule(new Bush::Bush1());
292             tree->addRule(new Bush::Bush2());
293             tree->addRule(new Bush::Bush3());
294             tree->addRule(new Bush::Bush4());
295
296
297
298             tree->pushBackInitElement(new
299                           RealTimeLSystem::Material(bush_mat));
300             tree->pushBackInitElement(new Size(1));
301             tree->pushBackInitElement(new Bush::A1());
302
303             for(int i=0; i<7; i++) {
```

```
304             tree->applyRules();
305         }
306
307         tree->setPosition(Vec3f(x,y,z));
308             tree->rotate(CGLA::YAXIS, M_PI/2.0f);
309         tree->scale(Vec3f(.05, .05, .05));
310
311             trees.push_back(tree);
312
313             terrain->addTree(tree);
314     }
315 }
316
317 void draw_scene() {
318       glLoadIdentity();
319
320       navigator->lookAt();
321
322       glLightfv(GL_LIGHT0, GL_POSITION, light_position);
323
324       terrain->draw();
325 }
326
327 void display() {
328
329       /* ZOOOOOOM */
330       // We switch to projection matrix model
331       glMatrixMode(GL_PROJECTION);
332
333       // Clear projection matrix.
334       glLoadIdentity();
335
336       // Specify a perspective projection
337       navigator->setPerspective(); // set zoom
338
339       // Go back to modelview mode.
340       glMatrixMode(GL_MODELVIEW);
341
342
343       // Clear the screen
344       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
345
346       // Now draw scene w. pick = false
347       draw_scene();
348
349       // Draw the frame rate counter.
350
351       glPushAttrib(GL_ALL_ATTRIB_BITS);
352       disp.draw();
353       glPopAttrib();
354
355       // Swap buffers so that we can see what was
356       // just drawn.
357       glutSwapBuffers();
358 }
359
360 void reshape(int w, int h) {
361       init_gl(w,h);
362       glViewport(0,0,w,h);
363       WINX = w;
364       WINY = h;
```

```
365         navigator->setWindowSize(w,h);
366         glutPostRedisplay();
367   }
368
369   void animate() {
370         QueryPerformanceCounter(&start_clk);
371
372       if((start_clk.QuadPart - last_time.QuadPart) >
373          CLK_PER_FRAME) {
374
375           navigator->advance();
376
377             if ( glutGetWindow() != main_window )
378                   glutSetWindow(main_window);
379
380             glutPostRedisplay();
381
382           QueryPerformanceCounter(&last_time);
383       }
384       QueryPerformanceCounter(&after_draw_clk);
385
386       diff = CLK_PER_FRAME - (after_draw_clk.QuadPart –
387                               start_clk.QuadPart);
388
389       if(diff < 0) {
390           cout << "WARNING no time for update" << endl;
391           return;
392       }
393
394       terrain->update(Vec3f(navigator->pos[0],
395                             navigator->pos[1],
396                             navigator->getZ()), diff);
397
398       QueryPerformanceCounter(&start_clk);
399   }
400
401   void visible(int vis) {
402         // If the window becomes visible we
403         // switch on the idle function.
404         if (vis == GLUT_VISIBLE)
405               glutIdleFunc(animate);
406         else
407               glutIdleFunc(0);
408   }
409
410   // ----------------------------------------
411   // Special keyboard function. Gets special
412   // characters from keyboard. (e.g. F1,
413   // arrow keys, etc.)
414   // ----------------------------------------
415
416   void spec_keyfun(int key, int, int) {
417         navigator->
418                   shift(GLUT_ACTIVE_SHIFT&glutGetModifiers());
419         navigator->ctrl(GLUT_ACTIVE_CTRL&glutGetModifiers());
420
421         switch(key) {
422               case GLUT_KEY_LEFT:
423                     navigator->left(true);
424                     break;
425               case GLUT_KEY_RIGHT:
```

```
426                     navigator->right(true);
427                     break;
428               case GLUT_KEY_UP:
429                     navigator->up(true);
430                     break;
431               case GLUT_KEY_DOWN:
432                     navigator->down(true);
433                     break;
434               }
435   }
436
437   // ---------------------------------------
438   // Special keyboard function for release
439   // Gets special characters from keyboard.
440   // (e.g. F1, arrow keys, etc.)
441   // ---------------------------------------
442
443
444   void spec_keyfun_up(int key, int, int) {
445         navigator->
446                  shift(GLUT_ACTIVE_SHIFT&glutGetModifiers());
447         navigator->ctrl(GLUT_ACTIVE_CTRL&glutGetModifiers());
448
449         switch(key) {
450               case GLUT_KEY_LEFT:
451                     navigator->left(false);
452                     break;
453               case GLUT_KEY_RIGHT:
454                     navigator->right(false);
455                     break;
456               case GLUT_KEY_UP:
457                     navigator->up(false);
458                     break;
459               case GLUT_KEY_DOWN:
460                     navigator->down(false);
461                     break;
462               }
463   }
464
465   // ---------------------------------------
466   // Keyboard function. Gets
467   // characters from keyboard.
468   // ---------------------------------------
469
470   void keyfun(unsigned char key, int, int) {
471         switch(key) {
472               case 'f':
473                     glutPositionWindow(10, 30);
474                     glutReshapeWindow(storedWINX,storedWINY);
475                     break;
476               case 'F':
477                     storedWINX = WINX;
478                     storedWINY = WINY;
479                     glutFullScreen();
480                     break;
481               case 'q':
482               case 'Q':
483               case 27:
484                     exit(0);
485               case 'p':
486                     navigator->print();
```

```
487                     break;
488             case 'L':
489                     glEnable(GL_LIGHTING);
490                     break;
491             case 'l':
492                     glDisable(GL_LIGHTING);
493                     break;
494       }
495   }
496
497   void mouse(int button, int state, int x, int y) {
498       Vec2i pos(x,y);
499
500       if (state==GLUT_DOWN && button==GLUT_LEFT_BUTTON) {
501             navigator->mousePressed(x,y);
502       }
503
504       if(state==GLUT_UP && button==GLUT_LEFT_BUTTON) {
505             navigator->mouseReleased();
506       }
507
508       glutPostRedisplay();
509   }
510
511   void move(int x, int y) {
512       navigator->move(x,y);
513   }
514
515   void camMode(int value) {
516       navigator->camMode(value);
517   }
518
519   void godModes(int value) {
520       navigator->godMode(value);
521   }
522
523   void initMenu() {
524
525       int god_modes = glutCreateMenu(godModes);
526       glutAddMenuEntry("Pan",1);
527       glutAddMenuEntry("Rotate",2);
528       glutAddMenuEntry("Zoom",3);
529
530       int cam_menu = glutCreateMenu(camMode);
531       glutAddMenuEntry("1st person",1);
532       glutAddMenuEntry("God mode",3);
533       glutAddSubMenu("God modes", god_modes);
534
535       glutCreateMenu(NULL);
536
537
538       glutAddSubMenu("Cam mode", cam_menu);
539
540       glutAttachMenu(GLUT_RIGHT_BUTTON);
541   }
542
543   int main(int argc, char* argv[]) {
544
545       // Initialize glut
546       glutInit(&argc,argv);
547       glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE);
```

```
548         glutInitWindowSize(WINX,WINY);
549         main_window = glutCreateWindow("Terrain Roamer");
550
551
552         // functions as needed.
553         glutDisplayFunc(display);
554         glutVisibilityFunc(visible);
555         glutKeyboardFunc(keyfun);
556         glutSpecialFunc(spec_keyfun);
557         glutSpecialUpFunc(spec_keyfun_up);
558         glutIgnoreKeyRepeat(true);
559         glutReshapeFunc(reshape);
560         glutMouseFunc(mouse);
561         glutMotionFunc(move);
562
563         // Set colour for background clearing
564         glClearColor(.5,.7,1.,1);
565
566         // Initialize OpenGL
567         init_gl(WINX,WINY);
568
569         // Init OpenGL extensions
570         int err = glewInit();
571         if (GLEW_OK != err) {
572               cout << "GLEW Error: "
573                    << glewGetErrorString(err) << endl;
574               return -1;
575         }
576
577         init_il();
578
579         initMenu();
580
581         if(argv[1])
582               terrain->loadHeightMap(argv[1]);
583         else
584               cout << "Please specify heightmap, texture,
585                        noise & ocean: TerrainRoamerEx6
586                       <heightMap> <bw-texture> <noise>
587                       <ocean>" << endl;
588
589         if(argv[2])
590               terrain->loadTexture(argv[2]);
591         else
592               cout << "Please specify heightmap, texture,
593                        noise & ocean: TerrainRoamerEx6
594                        <heightMap> <bw-texture> <noise>
595                        <ocean>" << endl;
596
597         if(argv[3])
598               terrain->loadNoiseTexture(argv[3]);
599         else
600               cout << "Please specify heightmap, texture,
601                        noise & ocean: TerrainRoamerEx6
602                        <heightMap> <bw-texture> <noise>
603                        <ocean>" << endl;
604
605         if(argv[4])
606               terrain->loadOceanTexture(argv[4]);
607         else
608               cout << "Please specify heightmap, texture,
```

```
609                         noise & ocean: TerrainRoamerEx6
610                         <heightMap> <bw-texture> <noise>
611                         <ocean>" << endl;
612
613         glEnable(GL_DEPTH_TEST);
614
615         terrain->createDisplayList();
616
617
618         disp.addFramerate();
619
620         initTrees();
621
622         LARGE_INTEGER frequency;
623
624         QueryPerformanceFrequency(&frequency);
625
626         CLK_PER_FRAME = frequency.QuadPart / 60;
627
628         QueryPerformanceCounter(&last_time);
629
630         // Pass control to GLUT.
631         glutMainLoop();
632         return 0;
633
634   }
635
636   GLuint loadTexture(char* filename) {
637
638         // Create and bind an image.
639         ILuint  ImgId;
640         ilGenImages(1, &ImgId);
641         ilBindImage(ImgId);
642
643         // Load the image - abort program if it fails.
644         if(!ilLoadImage(filename)) {
645               cout << "could not load " << filename << endl;
646               exit(0);
647         }
648         ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
649
650         // Get dimensions of image.
651         int width = ilGetInteger(IL_IMAGE_WIDTH);
652         int height = ilGetInteger(IL_IMAGE_HEIGHT);
653
654         const unsigned char* image = ilGetData();
655
656         // --------------------------------------
657         // Use the image as an OpenGL texture
658         // --------------------------------------
659
660       glPushAttrib(GL_ALL_ATTRIB_BITS);
661       GLuint tex;
662
663         //Create and bind a texture name
664         glGenTextures(1, &tex);
665         glBindTexture(GL_TEXTURE_2D, tex);
666
667         // Setup the texture interpolation
668         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
669                         GL_LINEAR);
```

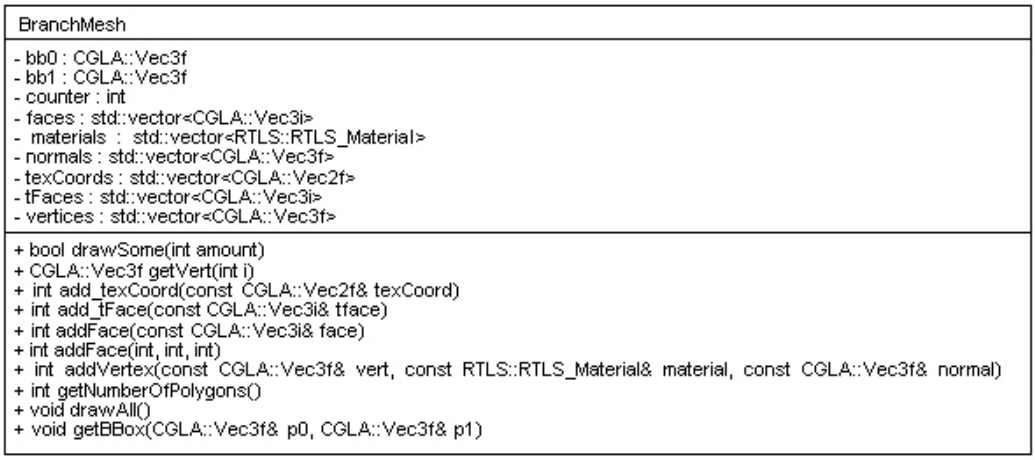
```
670         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
671                         GL_LINEAR);
672
673         // Specify how we deal with wrapping.
674         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
675                         GL_CLAMP);
676         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
677                         GL_CLAMP);
678
679         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
680                 height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
681
682       glPopAttrib();
683
684       return tex;
685   }
```

Appendiks C

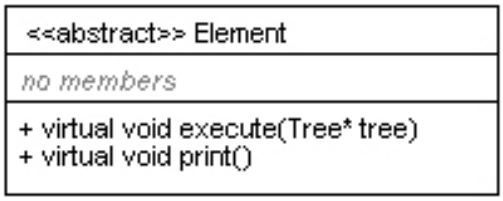
Realtidsrendering – Lindenmayers systemer

Klassediagrammer

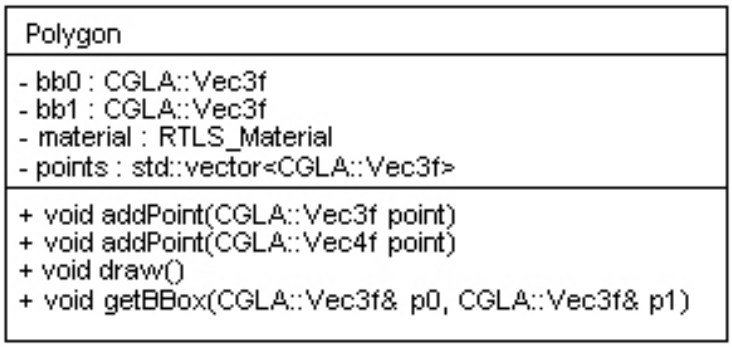
1 BranchMesh



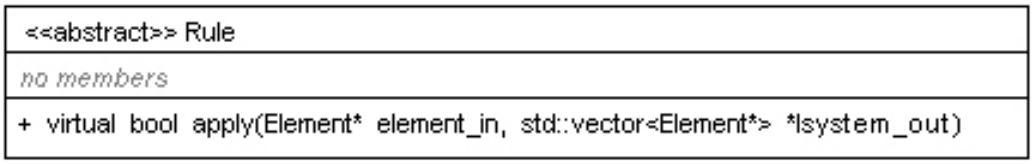
2 Element



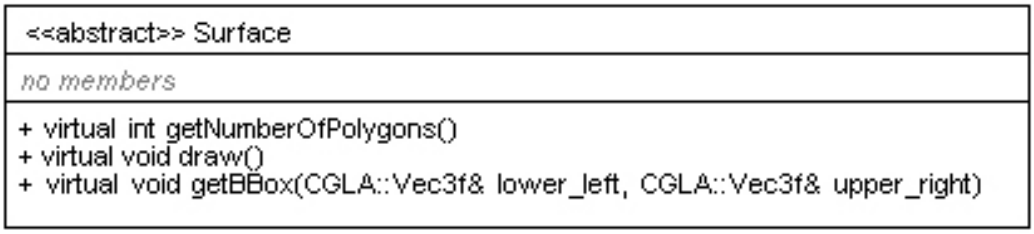
3 Polygon



4 Rule

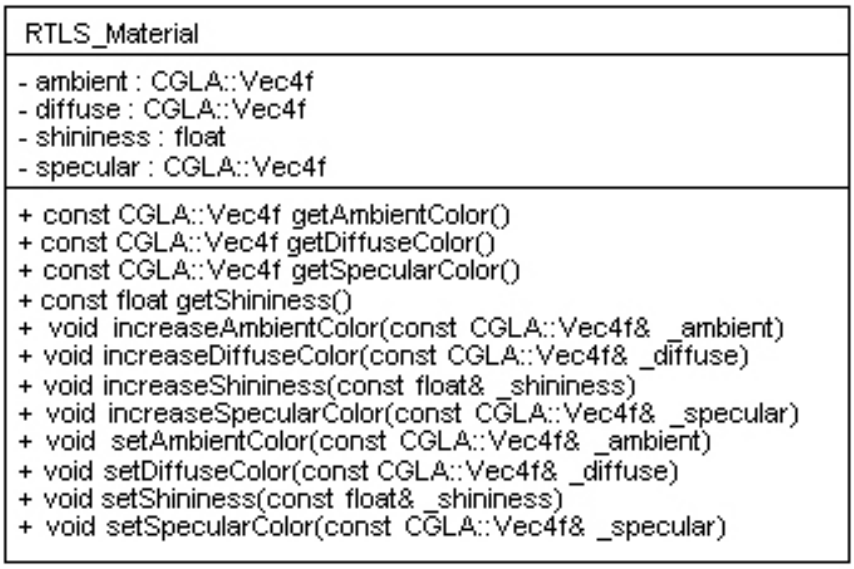


5 Surface

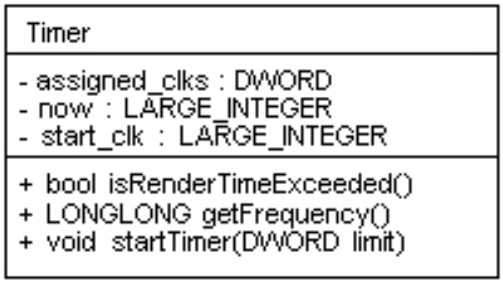


6 LoDSection

7 RTLS_Material



8 Timer



9 Tree


Tree
- lodSections : std::vector<RTLS::LoDSection*>
- rules : std::vector<Rule*>
- state : STATE
- tex : RenderTexture*
- int calculate_distance_linear(float units)
- int calculate_distance_x_squared(float units)
- int decent_to_depth_linear(int distance)
- int decent_to_depth_polynominal(int distance)
- void createTexture(int max_depth, int next_lod_if_finished)
- void detectVendor()
+ CGLA::Vec3f getPosition()
+ int getNumberOfPolygons()
+ int getNumberOfSectionsDrawn()
+ void addRule(RTLS::Rule* rule)
+ void And(float l)
+ void applyRules()
+ void ApplySurface(RTLS::Surface* surf)
+ void Backslash(float l)
+ void Color(CGLA::Vec3f color, bool absolute)
+ void DollarSign()
+ void Dot()
+ void draw()
+ void drawBBox()
+ void drawWithoutTexture()
+ void f(float l)
+ void F(float l)
+ void G(float l)
+ void Hat(float l)
+ void Minus(float l)
+ void Percent()
+ void Plus(float l)
+ void Pop()
+ void PopLoD()
+ void PopPolygon()
+ void printTiming()
+ void printWord()
+ void Push()
+ void pushBackInitElement(Element* e)
+ void PushLoD()
+ void PushPolygon()
+ void rotate(CGLA::Axis axis, float angle)
+ void scale(CGLA::Vec3f scale)
+ void setLightPosition(CGLA::Vec4f& light_pos)
+ void setMaterial(RTLS_Material& mat)
+ void setPosition(const CGLA::Vec3f& pos)
+ void setTexture(GLuint tex)
+ void Size(float w, bool absolute)
+ void Slash(float l)
+ void update(CGLA::Vec3f looking_from, DWORD render_clks)
+ void Uturn()

Appendiks D

Realtidsrendering – Lindenmayers systemer

Måledata

1 Hondas Træ

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,03	0,07	0,08	0,04	0,01
GETTING BBOX BRANCHES	0,03	0,03	0,02	0,01	0,00
GETTING BBOX POLYGONS	0,03	0,03	0,02	0,01	0,00
GETTING BBOX SURFACES	21,55	19,47	12,22	3,49	0,00
GETTING BBOX SUB INIT	0,00	0,01	0,01	0,00	0,00
GETTING BBOX SUB	0,00	1,61	8,85	17,69	37,25
BBOX DONE	0,04	0,05	0,05	0,02	0,00
RENDERING INIT	1,62	1,52	1,01	0,35	0,04
RENDERING BRANCHES	12,14	9,74	6,17	2,01	0,22
RENDERING POLYGONS	0,09	0,08	0,05	0,02	0,00
RENDERING SURFACES	86,45	77,08	50,44	14,76	0,00
RENDERING SUB	0,17	7,25	40,30	83,46	101,87
READ TEXTURE	751,13	697,10	470,10	157,25	17,39
EDIT TEXTURE	497,26	460,45	310,39	104,63	11,81
CREATE TEXTURE	17,39	17,10	37,88	5,08	0,72
Total	1387,92	1291,59	937,61	388,81	169,32

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 1

2 Palmen

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,05	0,05	0,00	0,01	0,00
GETTING BBOX BRANCHES	0,01	0,01	0,00	0,00	0,00
GETTING BBOX POLYGONS	0,01	0,01	0,00	0,00	0,00
GETTING BBOX SURFACES	1,50	1,50	0,01	0,01	0,00
GETTING BBOX SUB INIT	0,01	0,01	0,00	0,00	0,00
GETTING BBOX SUB	0,00	0,00	1,55	1,52	1,58
BBOX DONE	0,03	0,03	0,00	0,00	0,00
RENDERING INIT	0,40	0,39	0,07	0,08	0,04
RENDERING BRANCHES	7,02	7,05	0,67	0,67	0,72
RENDERING POLYGONS	0,03	0,03	0,00	0,00	0,00
RENDERING SURFACES	22,52	23,11	0,00	0,00	0,00
RENDERING SUB	0,05	0,05	29,86	30,67	31,30
READ TEXTURE	207,91	207,67	36,83	35,43	17,84
EDIT TEXTURE	146,68	145,68	24,03	24,11	12,10
CREATE TEXTURE	4,49	3,37	0,63	1,56	0,69
Total	390,70	388,95	93,66	94,06	64,29

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 1

3 Busken

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,39	0,12	0,04	0,01	0,00
GETTING BBOX BRANCHES	0,10	0,03	0,01	0,00	0,00
GETTING BBOX POLYGONS	0,70	0,12	0,05	0,02	0,00
GETTING BBOX SURFACES	0,46	0,13	0,06	0,02	0,00
GETTING BBOX SUB INIT	0,06	0,01	0,00	0,00	0,00
GETTING BBOX SUB	0,00	1,28	1,61	1,82	1,64
BBOX DONE	0,22	0,08	0,03	0,01	0,00
RENDERING INIT	4,18	1,52	0,47	0,14	0,04
RENDERING BRANCHES	38,59	6,57	2,69	1,01	0,07
RENDERING POLYGONS	57,48	6,73	3,12	1,07	0,00
RENDERING SURFACES	0,24	0,08	0,02	0,01	0,00
RENDERING SUB	0,49	81,53	93,06	97,22	100,11
READ TEXTURE	2092,89	718,66	228,57	69,16	17,20
EDIT TEXTURE	1409,43	469,52	152,67	46,61	11,96
CREATE TEXTURE	44,69	128,83	35,12	6,53	0,28
Total	3649,91	1415,20	517,51	223,64	131,32

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 1

4 Capsella

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,05	0,06	0,08	0,10	0,01
GETTING BBOX BRANCHES	0,02	0,02	0,02	0,02	0,00
GETTING BBOX POLYGONS	0,05	0,05	0,02	0,02	0,00
GETTING BBOX SURFACES	0,11	0,15	0,07	0,08	0,00
GETTING BBOX SUB INIT	0,01	0,01	0,01	0,01	0,00
GETTING BBOX SUB	0,00	0,00	0,16	0,16	0,33
BBOX DONE	0,04	0,05	0,06	0,06	0,00
RENDERING INIT	0,98	1,27	0,68	0,73	0,04
RENDERING BRANCHES	6,04	7,50	5,64	5,76	2,37
RENDERING POLYGONS	2,26	4,15	0,39	0,60	0,00
RENDERING SURFACES	0,05	0,07	0,03	0,04	0,00
RENDERING SUB	0,11	0,17	2,93	3,12	7,94
READ TEXTURE	474,89	517,94	298,60	318,47	17,59
EDIT TEXTURE	306,74	342,47	194,77	215,21	12,02
CREATE TEXTURE	42,73	34,49	24,87	18,23	1,46
Total	834,08	908,40	528,32	562,62	41,77

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 1

5 Hondas Træ, software

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,12	0,18	0,15	0,06	0,01
GETTING BBOX BRANCHES	0,11	0,11	0,08	0,03	0,00
GETTING BBOX POLYGONS	0,11	0,12	0,09	0,03	0,00
GETTING BBOX SURFACES	52,97	46,69	27,21	7,41	0,01
GETTING BBOX SUB INIT	0,01	0,04	0,04	0,01	0,00
GETTING BBOX SUB	0,00	6,59	26,32	59,05	103,43
BBOX DONE	0,20	0,22	0,17	0,06	0,01
RENDERING INIT	6,20	5,61	3,38	1,06	0,10
RENDERING BRANCHES	43,72	36,57	22,26	6,47	0,48
RENDERING POLYGONS	0,38	0,37	0,21	0,07	0,01
RENDERING SURFACES	168,17	148,00	87,58	23,77	0,01
RENDERING SUB	0,41	25,42	101,90	180,93	213,94
READ TEXTURE	2284,26	2032,82	1247,84	392,35	35,50
EDIT TEXTURE	2976,51	2648,92	1627,06	510,80	46,49
CREATE TEXTURE	17,73	12,56	7,73	2,38	0,22
Total	5550,90	4964,22	3152,03	1184,48	400,20

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 2

6 Hondas Træ, hardware

Tilstand \ LoD	1	2	3	4	5
GETTING BBOX INIT	0,12	0,18	0,15	0,06	0,01
GETTING BBOX BRANCHES	0,10	0,12	0,08	0,03	0,00
GETTING BBOX POLYGONS	0,11	0,12	0,09	0,03	0,00
GETTING BBOX SURFACES	53,05	46,29	27,27	7,26	0,01
GETTING BBOX SUB INIT	0,01	0,04	0,04	0,01	0,00
GETTING BBOX SUB	0,00	6,48	26,01	59,26	103,42
BBOX DONE	0,20	0,22	0,17	0,06	0,01
RENDERING INIT	6,22	5,46	3,39	1,02	0,09
RENDERING BRANCHES	41,78	40,20	24,32	7,14	0,54
RENDERING POLYGONS	0,38	0,33	0,21	0,07	0,01
RENDERING SURFACES	184,46	162,60	95,66	26,15	0,01
RENDERING SUB	0,53	27,43	109,19	195,84	229,77
READ TEXTURE	1,70	0,63	0,39	0,12	0,01
EDIT TEXTURE	2309,74	2056,59	1263,47	396,72	36,08
CREATE TEXTURE	2314,54	2060,18	1263,01	397,41	36,15
Total	4912,94	4406,87	2813,44	1091,18	406,10

Tiderne er angivet i millisekunder

Målingerne er udført på testmaskine 2

Appendiks E

Realtidsrendering – Lindenmayers systemer

Brugervejledning

1 Viewer

I denne applikation kan følgende taster benyttes:

q/Q/Esc	Afslutter applikationen.
t/T	Viser den teksturificerede model.
g/G	Viser polygon modellen.
b/B	Tegner alle bounding bokse.
y/Y	Viser den teksturificerede model sammen med alle bounding bokse.
h/H	Viser polygon modellen sammen med alle bounding bokse.
p/P	Toggler pause. Det er muligt at starte en simulering af at punktet modellen ses fra bevæger sig gennem alle afstandsniveauer. Nå applikationen startes så kører simulationen ikke.
a/A	Bevæger punktet, som simulere hvor modellen ses fra, -5 enheder i x-aksens retning.
z/Z	Bevæger punktet, som simulere hvor modellen ses fra, 5 enheder i x-aksens retning.
x/X	Viser et verdens koordinatsystem.
v/V	Printer tiderne for hvor længe algoritmen var om de forskellige tilstande.
n/N	Skriver antallet af sektioner, der benyttes til at repræsentere modellen i det afstands interval den er i på pågældende tidspunkt, ud i konsollen.
m/M	Skriver antal af polygoner der benyttes i polygon modellen ud i konsollen.
0-8	Nulstiller alle rotationer og translationer og sætter afstanden, som modellen ses fra, til 5 gange værdien af tasten.
9	Nulstiller alle rotationer og translationer og sætter afstanden, som modellen ses fra, til 80 enheder.

2 TerrainRoamer

Der er 2 muligheder for at bevæge sig rundt i terrænet. Den ene måde er i første persons synsvinkel. Man bevæger sig rundt med piletasterne og kan benytte ctrl-tasten til at bevæge sig sidelæns og shift-tasten til at bevæge sig hurtigere.
Den anden metode er i med et frit kamera. Der er 3 bevægelser man kan benytte, Pan, Rotate og Zoom. Man benytter musen til at lave de bevægelser ved at trykke venstre tast ned og trække musen.

Modellerne bliver opdateret efter placeringen af synsvinklen i 1. person.

Man skifter synsvinkel ved at trykke højre musetast ned og så kommer følgende menu op, hvor man kan vælge synsvinkel, samt bevægelserne i ”God mode”, som er det frie kamera.