VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

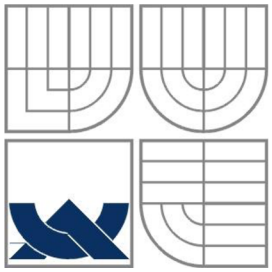
GENEROVÁNÍ REALISTICKÝCH MODELŮ STROMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

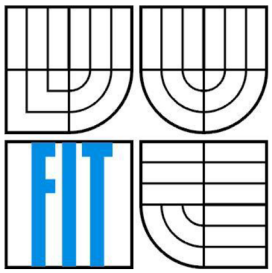
AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV LUŇÁK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GENEROVÁNÍ REALISTICKÝCH MODELŮ STROMŮ

GENERATION OF REALISTIC TREE MODELS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV LUŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2011

Abstrakt

Tato páce se zabývá generováním realistických modelů stromů. Implementovaný algoritmus má mnoho nastavitelných parametrů, které mu umožňují generovat velké množství druhů listnatých stromů. Generované modely jsou také poměrně realistické.

Abstract

This thesis deals with generation of realistic tree models. Implemented algorithm has many variable parameters, which allow generating large amount of species of broad-leaved trees. Generated models are also quite realistic looking.

Klíčová slova

Parametrické generování, stromy, modely, OpenSceneGraph

Keywords

Parametrical generation, trees, models, OpenSceneGraph

Citace

Luňák Miroslav: Generování realistických modelů stromů, diplomová práce, Brno, FIT VUT v Brně, 2011

Generování realistických modelů stromů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Lukáše Poloka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Luňák
25.5.2011

Obsah

1 Úvod

V dnešním moderním světě, kde jsou počítače využívané v každém odvětví průmyslu a služeb, jsou kladeny vysoké nároky na prezentaci a zobrazování dat. Trojrozměrná počítačová grafika je v tomto směru jedním z hlavních oborů. Jednou z výzev 3D grafiky je zobrazování realistických přírodních scenérií, které jsou pak používány ve většině interaktivních aplikací, jako jsou například počítačové hry, vojenské simulátory a také virtuální realita. A právě realistické zobrazení stromů a vegetace obecně přispívá k vyšší kvalitě a věrohodnosti.

Modely stromů je možné vytvářet mnoha způsoby. Například ručním vytvářením modelů. Tento přístup sice produkuje kvalitní modely, ale je velmi časově i finančně náročný. Další způsob je automatické procedurální generování modelů. To má oproti ručnímu vytváření mnoho výhod. Je podstatně rychlejší, a pokud je generující algoritmus kvalitní, tak i výsledné modely stromů v kvalitě nezaostávají za ručně vytvořenými modely. Zároveň umožňuje generovat velké množství druhů stromů pouhou změnou nastavení parametrů.

V první kapitole provedu přehled algoritmů pro generování stromů, na které jsem narazil při hledání vhodného algoritmu pro tuto práci. V dalších kapitolách se zaměřím na algoritmus pro generování realistických modelů stromů, který jsem si vybral a který vychází z dokumentu [1] v příloze Literatura. Bude v nich představeno fungování algoritmu a jeho následná implementace.

2 Algoritmy generování

2.1 Generování pomocí LSystémů

Pojem LSystém[2] pochází od Aristid Lindenmayera, který úpravou regulárních nebo bezkontextových přepisovacích gramatik pro účely modelováni vývoje jednoduchých mnohobuněčných organizmů, vytvořil teorii, která se později ukázala byt vhodná i pro simulaci růstu a vývoje i vysokých biologických forem jako jsou květiny, stromy nebo keře - tzv. Lindenmayerovi systémy (dále LSystémy). LSystém je tedy ve své podstatě gramatika.

Gramatikou nazýváme čtveřici: (N, ∑, P, S), kde: N je množina neterminálních symbolů, ∑ je množina terminálních symbolů, P je množina přepisovacích pravidel a S je startovací symbol gramatiky. Říkáme, že gramatika generuje nějaký jazyk. Jazyk je množina řetězců složených z terminálních symbolů. Generováním myslíme proces, kdy ze zadaného počátečního řetězce znaků (axiomů), aplikací přepisovacích pravidel z množiny P na neterminální symboly tohoto řetězce, generujeme řetězec nový tak dlouho, až ve výsledném řetězci jsou jen symboly terminální.

Toto ale LSystém modifikuje a místo toho se volí omezení počtu přepisování nějakou zvolenou konstantou. Přepisování některých symbolů řetězce je většinou pevně dané, ale může být také určeno na základě generátoru náhodných čísel (stochastické L-systémy). Každý symbol v řetězci má přiřazen jistý geometrický význam, například transformaci či generování/nakreslení objektu. V případě použití želví grafiky se jedná o příkazy pro posun a natočení želvy. Rozhodnutí, které z pravidel se použije, je deterministické, proto množina pravidel P nesmí obsahovat více pravidel se stejným znakem na levé straně. LSystému s takovýmto algoritmem výběru použitého pravidla se říká LSystém typu 0.

V předchozím odstavci jsem popsal, co je to LS typu 0. Tento LSystém používá deterministického rozhodování, které pravidlo použije. To ovšem znamená, že musí produkovat stále stejný řetězec znaků. Toto je bohužel pravda a to je jeho velká nevýhoda. Stromy vygenerované tímto druhem LS budou vypadat stále stejně.

Tento neduh řeší tzv. stochastický LSystém. Ten se od typu 0 liší tím, že má více pravidel, která mají na pravé straně stejný symbol. Při přepisování se poté z těchto pravidel vybírá nedeterministicky, např. na základě pravděpodobnosti, kterou každé pravidlo má u sebe uvedenou a s jehož četností se uplatňuje, nebo může být inicializováno pomocí generátoru náhodných čísel. Takovéto LSystémy už produkují pokaždé jiný řetězec. Jejich jazyk je rozmanitější. Jsou tedy pro generování přírodních objektů vhodnější. Mají ovšem jednu nevýhodu a to tu, že generované řetězce jsou pokaždé jiné a tudíž generované stromy také pokaždé vypadají jinak. Pokud bychom tedy chtěli vygenerovat stejný řetězec, museli bychom toto obejít. Řešení je ale v tomto případě

jednoduché. Stačí LSystém inicializovat stejným číslem a poté bude i výsledný vygenerovaný řetězec stejný.

2.1.1 Generování pomocí želvy

LSystém jako takový žádný strom vygenerovat nedokáže. Vygeneruje pouze dlouhý řetězec symbolů. Pro vlastní generování je třeba použít nástroj, který dané symboly vygenerovaného řetězce dokáže interpretovat jako pravidla pro růst stromu a jeho větvení a na základě těchto pravidel vygenerovat strukturu stromu.

Pro tento účel byla vytvořena metoda zvaná želva [2]. Želva dokáže vykonávat příkazy, které jsou jí dávány. Má svůj aktuální vnitřní stav, součástí kterého jsou atributy jako poloha, směr. Může jich být, a také jich je, typicky více. Ale tyto 2 jsou základní. Příkazy pro želvičku jsou např.: otoč se o 30 stupňů okolo *osy x*, nebo pohni se o jednotku vpřed a nakresli tak čáru (segment). Každý příkaz tedy změní vnitřní stav želvy. Želva také pro potřeby rozvětvování musí obsahovat stavový zásobník, ve kterém uchovává aktuální stav, pokud narazí na rozvětvení. Po dokončení generování větve se vrátí k uloženému stavu a pokračuje dál ve zpracování původní větve. A tímto způsobem dokáže vygenerovat celou strukturu stromu. Příklad stromu vygenerovaného pomocí LSystému lze vidět na obrázku č. 1.

Obrázek 1 - keř vygenerovaný pomocí LSystému [2]

Obrázek 2 - strom vygenerovaný pomocí LSystému

2.2 Generování parametrickými generátory

Parametrické generování představuje úplně odlišný přístup ke generování stromů. Struktura stromu se negeneruje pomocí pravidel nebo gramatik jako u LSystémů, ale vytváření struktury se řídí pomocí daných parametrů. Tento přístup má své výhody i nevýhody. Mezi výhody patří lepší kontrola nad výsledným tvarem stromu a snadnější vizualizace struktury stromu při zadávání parametrů. Nevýhodou naopak je, že pokud chceme detailně řídit generování stromu, je nutné mít generátor s velkým množstvím parametrů a pak delší dobu trvá, než vytvoříme konfiguraci správných hodnot tak, aby výsledný strom vypadal podle našich představ.

Ve své práci jsem si vybral právě tento typ generujícího algoritmu.

3 Generující algoritmus

V této části bude představen přístup při návrhu implementovaného algoritmu pro generování realistických modelů stromů. Algoritmus pracuje na bázi parametrického generování struktury stromu. Při generování se algoritmus nesnaží napodobit biologický růst stromu, ale pomocí nastavených parametrů generuje pouze jeho výslednou strukturu. Jednotlivé parametry a jejich vliv na výslednou strukturu stromu budou popsány dále v této kapitole.

Stavbu generovaných stromů můžeme logicky i strukturálně rozdělit na jednotlivé generace. První generací stromu je kmen. V parametrech generátoru ale nemá označení jako první generace, ale jako nultá generace. Je to z toho důvodu, že pro kmen i větve je poměrně velké množství společných parametrů a tímto značením lze snáze udržet přehled o jednotlivých generacích stromu.

Stejný trend následuje i pro větve, takže o větvích vyrůstajících z kmene hovoříme jako o větvích první generace. Z nich vyrůstají větve druhé generace, atd.

Algoritmus je téměř úplnou implementací algoritmu popsaném v [1] uvedeném v sekci Literatura.

3.1 Parametry generátoru

Popis jednotlivých parametrů uvádím hned z kraje, protože mi přijde důležitý pro pochopení a snadnější vizualizaci generujícího algoritmu. Vysvětlení vlastní funkce parametru a jeho významu na výslednou strukturu stromu bude v následující kapitole. Význam jednotlivých parametrů pak bude možné snadno pochopit z obrázků č. 1, 2 a 3.

3.1.1 Konvence pro parametry

Všechny parametry generátoru uvedené v tomto dokumentu jsou napsány tučně a kurzívou pro lepší čitelnost textu. Některé z parametrů jsou uvedeny s prefixem n, který nabývá hodnot 0 - 3 a pomocí kterého lze snadno identifikovat, ke které generaci stromu jednotlivý parametr patří.

Za mnohými parametry následuje sufix V, například ***nLenghtV***. Tyto parametry slouží pro nastavení variability. Určují, v jakém rozsahu se budou měnit hodnoty parametrů se stejným jménem, (pouze bez sufixu V) v tomto případě **nLengthV** určuje variabilitu parametru **nLength**. Toto detailní nastavování slouží k přesnější kontrole tvaru generovaných stromů. Pokud tedy nastavíme variabilitu každého parametru na hodnotu 0, budeme dostávat stále stejný výsledný strom.

Generátor má omezení týkající se nastavování parametrů pro jednotlivé generace větví. Parametry lze nastavovat maximálně pro 3. generaci. Všechny další generace už používají stejné nastavení jako generace třetí. Je to z toho důvodu, že 4 generaci tvoří už jen poměrně krátké větve a nemají tedy vliv na výsledný tvar stromu, pouze na jeho hustotu.

3.1.2 Parametry pro vzhled celého stromu

Shape	id tvaru stromu
BaseSize	délka kmene od země, na které nejsou větve
Scale, ***ScaleV***	velikost stromu v metrech
Levels	počet generací větví
Ratio	koeficient průměr/velikost
RatioPower	redukce průměru pro novou generaci větví
Lobes	počet sinusoidálních výběžků na začátku kmene
LobesDepth	hloubka výběžků
Flare	exponenciální rozšíření na začátku kmene

3.1.3 Parametry pouze pro kmen

BaseSplits	rozdělení kmene na více částí

3.1.4 Parametry pouze pro větve

nBranches	počet větví n-té generace
nBranchDist	vzdálenost mezi větvemi
nDownAngle, ***nDownAngleV***	úhel odklonu od rodičovské větve
nRotate, ***nRotateV***	úhel otočení mezi větvemi okolo rodičovské větve

3.1.5 Společné parametry pro kmen a větve

nLength, ***nLengthV***	délka n-té generace
nTaper	zúžení větve
nSegSplits	počet rozdělení na segmenty stejné generace
nSplitAngle, ***nSplitAngleV***	úhel, který svírají rozdělené segmenty kmene
nCurveRes	počet segmentů, ze kterých je složena větev
nCurve, ***nCurveBack***, ***nCurveV***	řízení zakřivení jedné generace

3.1.6 Parametry listů

Leaves	počet listů na jednu větev
LeafScale	výška listu
LeafScaleX	poměrná šířka listu

3.1.7 Parametry ořezávání

AttractionUp	tendence větví růst nahoru nebo dolů
PruneRatio	koeficient prořezávání
PruneWidth	největší šířka prořezávání
PruneWidthPeak	místo, kde se nachází nejširší místo ořezávací obálky
PrunePowerLow, ***PrunePowerHigh***	zakřivení obálky

3.1.8 Parametry pro texturování

TextureWrapX	počet opakování textury po obvodu větve
TextureWrapY	počet opakování textury po délce větve

4 Vytváření stromu

4.1 Kmen a větev

Model rozlišuje dva druhy generovaných entit. Jsou to kmen (větev) a list. Kmen a větve používají stejnou parametrickou reprezentaci pro určení jejich tvaru, takže pokud použiji v této sekci pojem kmen, platí stejné tvrzení i o větvích. Geometrickou reprezentací je zužující se tubus, který má téměř vždy tvar kužele. Každý kmen má vlastní relativní souřadný systém, ve kterém je směr *osy z* shodný s centrální osou jej reprezentujícího kužele. Pro hlavní větev například platí, že *osa z* míří kolmo (podle nastavení parametrů) od *osy z* kmene a tudíž je rovnoběžná s povrchem země. *Osa y* míří směrem od pozorovatele a *osa x* míří kolmo k zemi.

4.1.1 Tvar kmene a větve

Tubus stonku je rozdělen na několik segmentů téměř kuželového tvaru definovaných parametrem ***nCurveRes***. Zakřivení stonku se řídí parametry ***nCurve***, ***nCurveBack***, ***nCurveV***. Pokud je hodnota parametru ***nCurveBack*** nulová, je osa z každého segmentu stonku pootočena okolo *osy y* od *osy z* předcházejícího segmentu o hodnotu (***nCurve***/***nCurveRes***) stupňů. Pokud je ***nCurveBack*** různý od nuly, každý ze segmentů první poloviny stonku bude odkloněn od předchozího o hodnotu (***nCurve***/(***nCurveRes***/2)) stupňů a každý segment v druhé polovině bude odkloněn o hodnotu (***nCurveBack***/(***nCurveRes***/2)) stupňů. Toto dvou-parametrové zakřivení umožňuje generovat stonky ve tvaru písmene S. V obou případech je k zakřivení přidána náhodná hodnota, která dosahuje velikosti (***nCurveV***/***nCurveRes***).

U větví od druhé generace lze ještě dodatečně upravovat tvar pomocí parametru ***AttractionUp***, jehož vliv je popsán dále v této kapitole.

4.2 Větvení pomocí klonování

Kmen a větve se v mnoha případech rozdělují a vytváří klony podél celé své délky. Tyto klonované části jsou stejné generace jako kmen, ze kterého vznikly a dědí také všechny vlastnosti a nastavení. Četnost tohoto dělení je definována parametrem ***nSegSplit***. Jeho hodnota určuje počet vytvořených klonů pro každý segment stonku a nejčastěji se pohybuje mezi hodnotou 0 a 1. Obecně není žádné omezení na velikost hodnoty tohoto parametru, ale každý klon poté může generovat další klony a počet segmentů tak může dosáhnout nežádoucích hodnot. Pro toto dělení se ještě používá přídavný parametr ***nBaseSplit***, který je ekvivalentní parametru ***nSegSplit*** a definuje počet klonů vytvořených pouze na konci prvního segmentu kmene. To umožní generovat stromy, které vypadají jako by měly

větší počet kmenů. Zlomková hodnota parametru ***nSegSplit*** (např. 1,2; 0,5) způsobí, že dělení na klony bude rovnoměrně rozprostřeno přes všechny segmenty stonku stejné generace. Například hodnota ***nSegSplit*** = 1,2 způsobí, že na 80% segmentů se vytvoří jeden klon a na 20% se vytvoří klony dva.

Pokud dojde k vytvoření klonu, dojde k odklonu *osy z* generovaného stonku i jeho klonů od předchozího segmentu o hodnotu

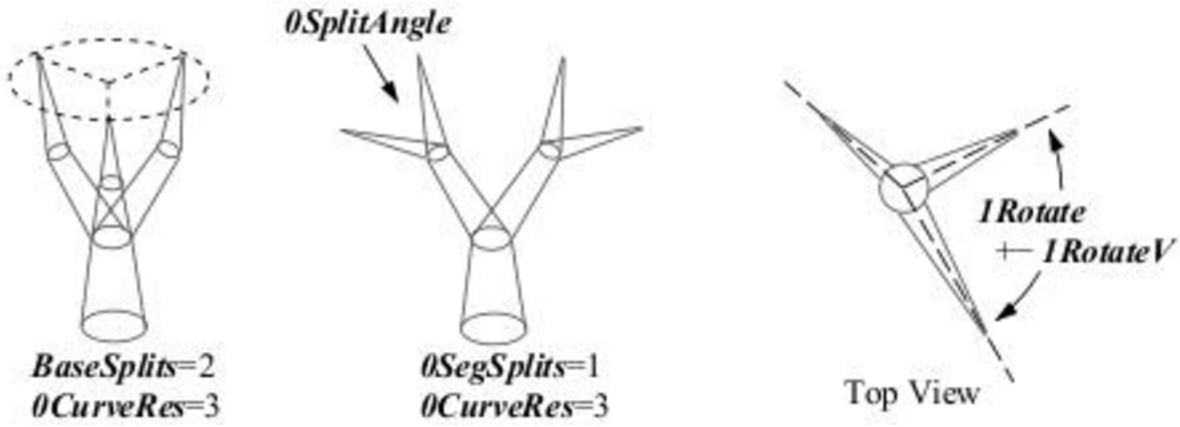
$$anglesplit = (\mathbf{nSplitAngle} \pm \mathbf{nSplitAngleV}) - declination$$

$$declination = cos^{-1}(transformation\_z_z)$$

kde declination je úhel odklonu lokální souřadné *osy z* od *osy z* celého stromu.

Stonky a jejich klony jsou také rozprostřeny rotací okolo osy rovnoběžné s *osou z* stromu. Tato osa rotace prochází bodem, kde se objevilo rozdělení. Na obrázku č. 3 je vidět význam těchto parametrů.

Pokud dojde k rozdělení klonováním, mají při generování následné segmenty tendenci navracet se do původního směru. To znamená, že hodnota *anglesplit* je distribuovaná mezi zbývající segmenty větve.



Obrázek 3 – ukázka významu některých parametrů [1]

4.3 Vytváření větví další generace

Dělení kmene a větví pouze pomocí klonů by omezilo různorodost stromů, které by bylo možno pomocí tohoto algoritmu generovat. Klonování totiž umožňuje vytvořit pouze rovnoměrnou distribuci nových větví, a proto vytváří velmi stejnorodé tvary stromů. Algoritmus umožňuje vytvářet větve vyšší generace. Každá generace větví má vlastní sadu parametrů kontrolujících tvar, odklon a rotaci. Proto může mít nastaveny úplně odlišné hodnoty u parametrů a tímto lze mnohem lépe kontrolovat výsledný tvar stromu. Některé parametry jsou nastavovány relativně podle rozměrů rodičovské větve, případně kmene, pokud se jedná o větve první generace. V případě kmene se pro tyto parametry bere nastavení podle celého stromu.

4.3.1 Počet větví nové generace

Parametr **nBranches** definuje maximální počet větví další generace, které mohou být vytvořeny na stonku aktuální generace. Skutečný počet nových větví je většinou menší než toto maximum a závisí na relativní délce větve oproti rodičovské větvi a maximální možné délce větví aktuální generace. Počet větví následující generace je vypočítán pomocí tohoto vzorce:

$$stems = stems_{max} * (0{,}2 + 0{,}8 * (length_{child}/length_{parent})/\text{length}_{\text{child,max}}$$

pro první generaci větví, a

$$stems = stems_{max} * (1 - 0{,}5 * offset_{child}/length_{parent})$$

pro následující generace větví, a kde $offset_{child}$ je vzdálenost aktuální větve, která generuje potomky od počátku rodičovské větve ve směru růstu. To znamená, že nejvíce větví následující generace je vytvořeno na nejdelších větvích.

Každá větev, která již byla klonována pomocí parametru **nSegSplit** snižuje náchylnost pro další dělení na klony na polovinu. Je to z důvodu velkého množství rozdělených segmentů a následné přílišné složitosti modelu. Pro dosažení větší hustoty stromu je lepší použít vyšší počet větví nebo přidat další generaci.

4.3.2 Délka větví

Maximální délka větve nové generace je počítána relativně k délce svého předka.

$$lenghtchild_{max} = (\mathbf{nLenght} + \mathbf{nLenghtV})$$

Například pokud je **nLenght** = 0,3 a délka rodičovské větve je rovna 10 metrům, pak maximální délka nové větve bude 3 metry. Vlastní délka se poté počítá takto:

$$lenght_{child} = length_{trunk} * length_{child,max} *$$

$$ShapeRatio(\mathbf{Shape}, (lenght_{trunk}\text{-}offset_{child})/(length_{trunk}\text{-}length_{base}))$$

pro větve první generace a

$$length_{child} = length_{child,max} * (length_{parent} - 0{,}6 * offset_{child})$$

pro další generace větví, kde $length_{base}$ je délka spodní části stromu v metrech, na které nejsou generovány žádné větve a která se spočítá jako (***baseSize****$scale_{tree}$), kde $scale_{tree}$ je definováno jako

$$\text{scate}_{\text{tree}} = (\textbf{\textit{Scale}}+\textbf{\textit{ScaleV}})$$

Funkce *ShapeRatio* je popsána v následující kapitole. Kmen stromu nemá žádného předka, je to větev nulté generace, takže jeho délka se spočítá takto

$$lenght_{trunk} = (\textbf{\textit{0lenght}} +\text{-} \textbf{\textit{0lenghtV}}) * scale_{tree}$$

4.3.3 Odklon a rotace větví

K odklonu a rotaci větví slouží parametry **DownAngle** a **Rotate**.

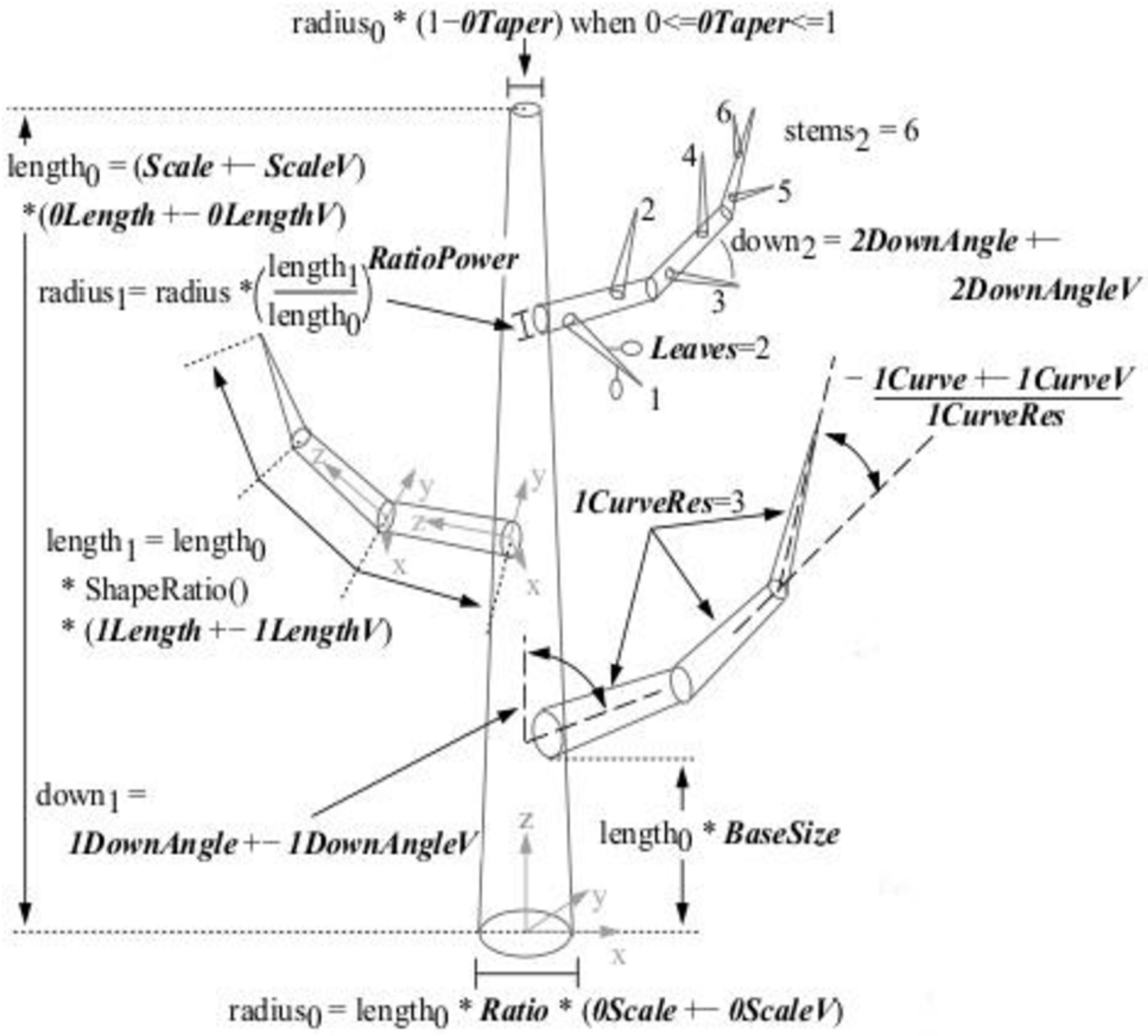
Pokud je hodnota parametru ***nDownAngleV*** kladná, je *osa z* potomka (stonek další generace) od *osy z* rodiče otočena podél *osy x* o hodnotu (***nDownAngle*** +- ***nDownAngleV***). Ale pokud je jeho hodnota záporná, je odchylka distribuovaná podle výšky stromu podle tohoto vzorce

$$downAngle_{child} = \textbf{\textit{nDownAngle}} +\text{-}$$

$$[\textbf{\textit{nDownAngleV}} * (1 - 2 * ShapeRatio(0, (length_{parent} - offset_{child}) / (length_{parent} - length_{base})))]$$

To umožňuje lineárně měnit downAngle podle toho, ve kterém místě délky rodičovského stonku dochází k větvení.

Pokud je hodnota parametru ***nRotate*** kladná, dochází k tomu, že každý potomek vytvořený podél svého předka je otočen podél *osy z* předka oproti předchozímu potomkovi o určitý úhel. Tímto dochází k umístění všech potomků podél rodičovského stonku přibližně do spirály. Úhel otočení oproti předchozímu potomkovi se vypočítá jako (***nRotate*** +- ***nRotateV***). Ve speciálním případě, kdy je hodnota parametru ***nRotate*** záporná, je každý potomek otočen podél *osy z* předka oproti předchozímu potomkovi o úhel (180 + ***nRotate*** +- ***nRotateV***). Tím dochází k umístění potomků na protějších stranách předka a umožňuje vytváření potomků téměř v jedné rovině.



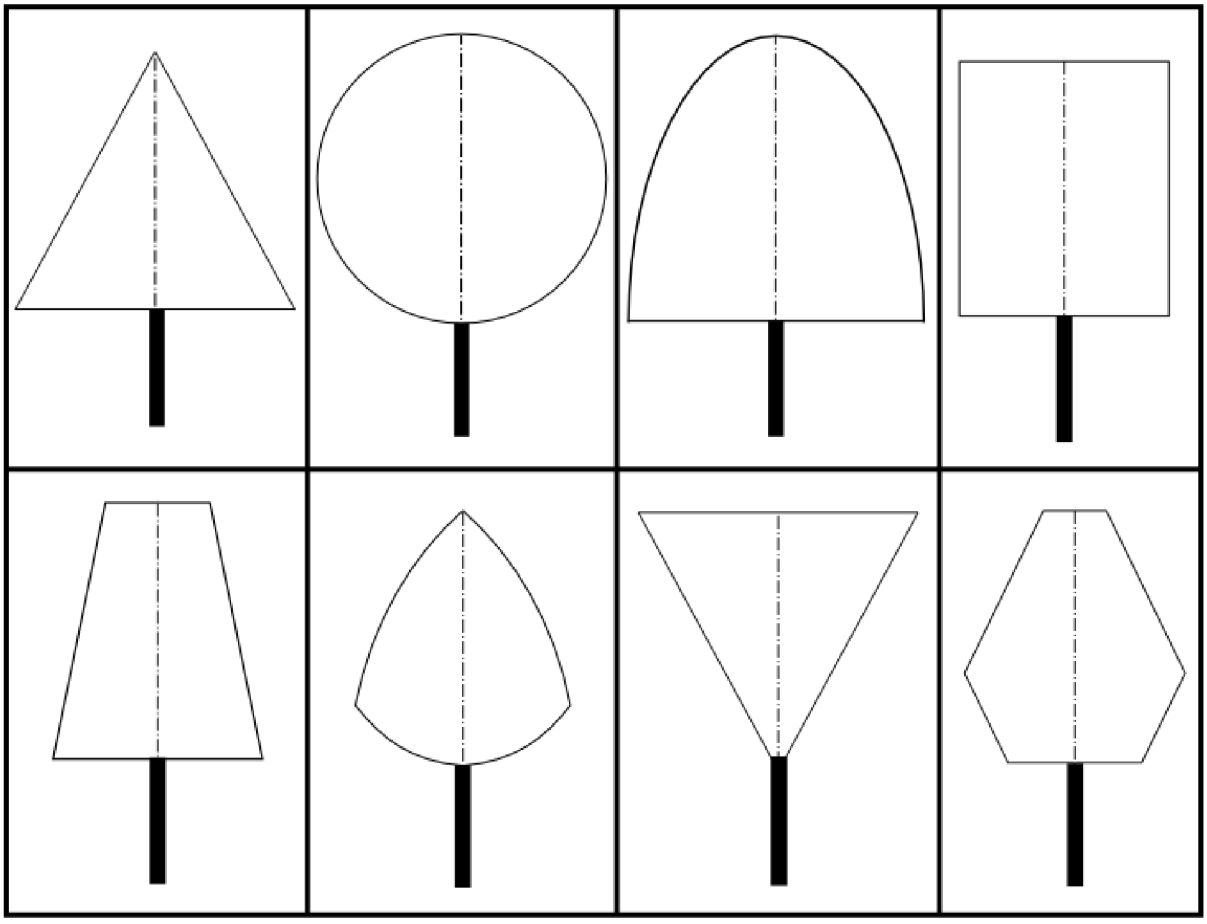
Obrázek 4 - ukázka významu parametrů [1]

4.4 Tvar koruny stromu

V přírodě se vyskytuje velké množství tvarů korun stromů. V algoritmu pro napodobení nejčastějších tvarů slouží parametr ***Shape***. Tento parametr ovlivňuje výpočet délky a zakřivení větví první generace podle toho, v jaké relativní pozici na kmenu se nacházejí (0 pro začátek kmene, 1 pro konec). Samotný výpočet koeficientů, kterými se násobí výsledná délka nebo odklon je realizován funkcí *ShapeRatio*. Tato funkce má 2 vstupní parametry. Jsou to hodnota parametru ***Shape***, který definuje tvar koruny a parametr ratio, který představuje koeficient vypočtený z délky rodičovské větve a aktuální pozice vzhledem k délce předka, na které je daný potomek generovaný. V tabulce č. 1 je uvedeno, jaké tvary je algoritmus schopen generovat a jaké hodnoty vrací funkce *ShapeRatio* pro jednotlivé hodnoty parametru ***Shape***. Samotné tvary jsou zobrazeny na obrázku č. 5.

ShapeRatio	
Shape	**Výsledek**
0 (conical)	0.2 + 0.8 * ratio
1 (spherical)	0.2 + 0.8 * sin(p * ratio)
2 (hemispherical)	0.2 + 0.8 * sin(0.5 * p * ratio)
3 (cylindrical)	1.0
4 (tapered cylindrical)	0.5 + 0.5 * ratio
5 (flame)	ratio/0.7 pokud je ratio ≤ 0.7
	(1.0 - ratio)/0.3 pokud je ratio > 0.7
6 (inverse conical)	1.0 - 0.8 * ratio
7 (tend flame)	0.5 + 0.5 * ratio/0.7 pokud je ratio ≤ 0.7
	0.5 + 0.5 * (1.0 - ratio)/0.3 pokud je ratio > 0.7
8 (envelope)	Využití ořezávací obálky popsané v kapitole xxx

Tabulka 1 - Popis výstupů funkce ShapeRatio



Obrázek 5 - typy tvarů stromů. Horní řádek: conical, spherical, hemispherical, cylindrical. Spodní řádek: tapered cylindrical, flame, inverse conical, tend flame

4.5 Poloměr kmene a větví

Pro všechny generace větví kromě kmene platí, že průměr tubusu na začátku větve je počítán relativně k průměru tubusu rodiče v místě, kde dochází k jejímu vytvoření. Průměr kmenu je přímo úměrný rozměru celého stromu.

$$radius_{trunk} = length_{trunk} * \mathbf{Ratio} * \mathbf{0Scale}$$

$$radius_{child} = radius_{parent} * (length_{child} / length_{parent})^{RatioPower}$$

Maximální průměr potomka je omezen na hodnotu průměru předka v místě, ve kterém došlo k větvení. Průměr větve nebo kmene může být podél své délky zužován. Toto zužování je řízeno parametrem ***nTaper***. Mohou nastat dvě extrémní situace. Pokud je ***nTaper*** roven nule, nedochází k žádnému zužování a větev je vygenerována jako válec. Pokud je roven 1, dojde k zúžení do bodu a větev je generována jako jehlan. Povolení nastavení pro tento parametr je tedy mezi těmito limitními hodnotami (včetně). V nejjednodušším případě dochází ke generování stonku jako zužujícího se kužele. V reálné situaci ale průměr vždy dosahuje určité minimální hodnoty, proto i při generování každé větve je nastaven minimální průměr, pod který vypočtená hodnota nesmí klesnout.

4.5.1 Flare a Lobes

Speciálně pro kmen jsou ještě definovány parametry ***Flare*** a ***Lobes***, které mění průměr na počátku kmene. Parametr ***Flare*** vytváří exponenciální rozšíření na začátku kmene. Pro normalizovanou pozici *Z* podle délky kmene v rozmezí 0 až 1 je průměr zvětšen o $flare_z$. Hodnota $flare_z$ se počítá následovně:

$$y = 1 - 8 * Z$$

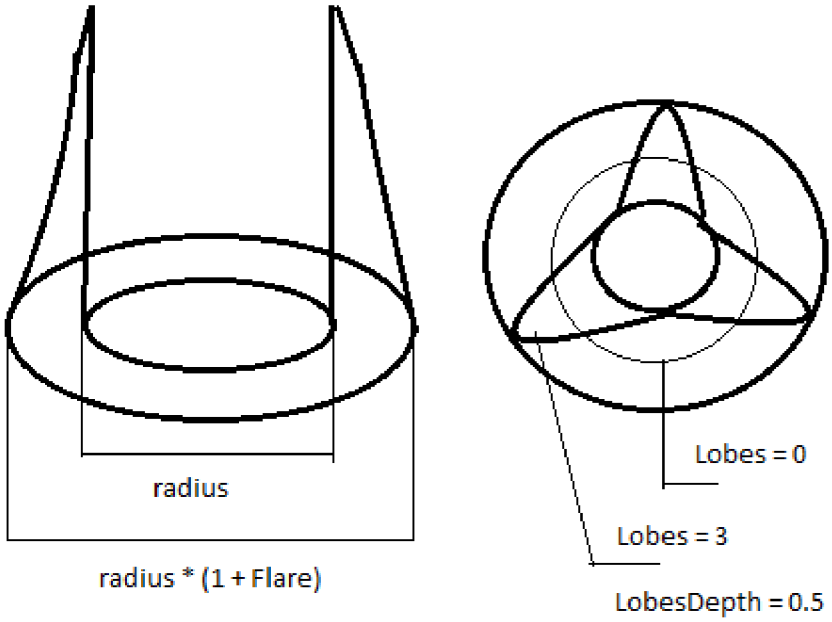
$$flare_z = \mathbf{Flare} * (100^y - 1) / 100 + 1$$

a kde minimální hodnota proměnné *y* je 0. Proměnná $flare_z$ změní rádius minimálně o 1 a maximálně o (1 + ***Flare***). Kmen je tímto rozšiřován pouze do 1/8 své výšky.

Parametry ***Lobes*** a ***LobeDepth*** slouží pro definování sinusového rozšíření v dolní části kmene. Hodnota parametru ***Lobes*** určuje, kolik těchto výběžků bude kmen v dolní části mít. A parametr ***LobeDepth*** určuje jejich šířku stejným způsobem, jako to dělá parametr ***Flare***. Hodnotu parametru ***Lobes*** je nejlepší nastavovat na lichá čísla, protože sudé hodnoty mají za následek viditelnou symetrii, která není moc přirozená. Rozšíření kmene $lobe_z$ vypočítáme takto:

$$lobe_z = 1.0 + \mathbf{LobeDepth} * sin(\mathbf{Lobes} * angle)$$

Význam těchto parametrů je vidět na obrázku č. 6.



Obrázek 6 - Parametry Flare a Lobes

4.6 Listy

Listy představují poslední krok generování stromu. Počet vygenerovaných listů se řídí parametrem ***Leaves***, který určuje hustota a počet generovaných listů stejným způsobem, jako určoval parametr ***nBranches*** pro větve. Hustota olistění je také závislá na dalších faktorech, jako například poměru délky rodičovské větve k maximální délce větví, do které rodičovská větev patří. Listy jsou generovány až od 2 levelu rekurze (kmen je level 0), to znamená od druhé generace větví dále.

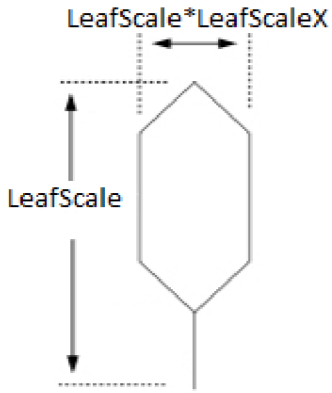
Výpočet hustoty větví se řídí následujícím vzorcem.

$$leaves\_per\_branch = \textbf{Leaves} * ShapeRatio(4\ (tapered), offset_{child} / length_{parent})$$

Funkce ShapeRatio pomáhá s rozložením největšího počtu listů do vnějších částí stromu, čímž je dosaženo vytvoření přirozeného vzhledu.

Rotace a odklon listů se opět řídí parametry ***DownAngle*** a ***Rotate***, které si bere od mateřské větve a jejichž aplikací se projeví stejný efekt jako u větví.

Samotná struktura listu je tvořena jednoduchým čtvercem, na který je nanesena textura s alfa-kanálem. Rozměry listu určují parametry ***LeafScale*** a ***LeafScaleX***. Jejich význam je ukázán na obrázku č. 7.



Obrázek 7 - parametry listu [1]

4.6.1 Orientace Listů

Listy vygenerované jen pomocí parametrů mají obecně pouze náhodnou orientaci. V reálném světě ale jsou listy natočeny vždy vzhůru, aby maximalizovali množství dopadajícího slunečního světla.

Tento efekt rotace ke slunci je kontrolován parametrem ***Bend***, který nabývá hodnot 0-1, kdy hodnota 0 efektivně toto natočení vypne. Pro výpočet je potřeba si z aktuální transformace vytáhnout pozici listu *(leafx, leafy, leafz)* a normálu (*leafnx, leafny, leafnz*), pomocí kterých dopočítáme požadované rotační úhly.

$$theta_{position} = atan_2\ (leafy, leafx)$$

$$theta_{bend} = theta_{position} - atan_2\ (leafny,\ leafnx)$$

a upravením rotační matice

$$rotate_z\ (\mathbf{Bend} * theta_{bend})$$

a poté dáme dohromady finální transformaci, pomocí které dopočítáme novou normálu.

$$phi_{bend} = atan_2\ (\ sqrt(\ Leafnx^2 + Leafny^2),\ leafnz)$$

$$rotate_z\ (\ -orientation\ )$$

$$rotate_x\ (\ \mathbf{Bend} * phi_{bend})$$

$$rotatez(\ orientation\ )$$

kde *orientation* získáme jako inverzní cosinus *z* souřadnice *z* vektoru aktuální transformační matice.

Takto upravené rotace listů zvýší difusní odraz světla od stromu a tím se zlepší jeho realistický vzhled.

4.7 Prořezávání

Prořezávání se používá k tomu, aby tvar stromu pasoval do určité obálky. Funguje tak, že růst větví je blokován hranicemi obálky. Každá větev musí zredukovat svou délku tak, aby nebyly překročeny hranice obálky. Obrázek č. 8 ukazuje konstrukci prořezávací obálky. Obálka má tvar rotačního elipsoidu a na obrázku č. 9 jsou vidět jednotlivé možnosti tvaru obálek.

Rozměry obálky jsou ohraničené zezdola hodnotou (***BaseSize****$\text{scale}_{\text{tree}}$) v metrech, maximální výška obálky je definována hodnotou (***PruneWidth****$\text{scale}_{\text{tree}}$) v metrech. Maximální šířka obálky je definována parametrem ***PruneWidth*** a nachází se na pozici ***PruneWidthPeak*** podél *osy z*. Tato pozice je definována jako procentuální vzdálenost výšky obálky od spodu obálky. Pokud bude hodnota parametru ***PruneWidthPeak*** rovna 0,5, bude se největší šířka nacházet přesně v polovině obálky.

Zakřivení obálky je definováno parametry PrunePowerHigh a PrunePowerLow, kde PrunePowerHigh je zakřivení nad nejširším místem (v horní polovině koruny stromu) a PrunePowerLow určuje zakřivení pod nejširším místem. Pokud je velikost těchto parametrů rovna 1, bude tvar obálky lineární, pokud větší než jedna, tvar bude konvexní a menší než 1 konkávní.

Abychom zjistili, jestli se daný bod *(x,y,z)* nachází uvnitř obálky, použijeme následující výpočet. Jeho výsledkem je proměnná *inside*, která je typu *boolen*. Pokud výsledná hodnota je *true*, daný bod splňuje podmínku a nachází se uvnitř obálky. Pokud je výsledná hodnota *false*, bod je mimo a musí dojít ke zmenšení délky větve, novému vygenerování a nové kontrole. Tento cyklus pokračuje do té doby, než je podmínka splněna.

$$r = sqrt(x^2 + y^2 + z^2)$$

$$ratio = (scale_{tree} - z) / (scale_{tree} *(1-\textbf{BaseSize}))$$

$$inside = [r / scale_{tree} < \textbf{PruneWidth} * ShapeRatio(8 (envelope), ratio)]$$

kde funkce pro **Shape** číslo 8 *ShapeRatio(8, ratio)* je definována jako

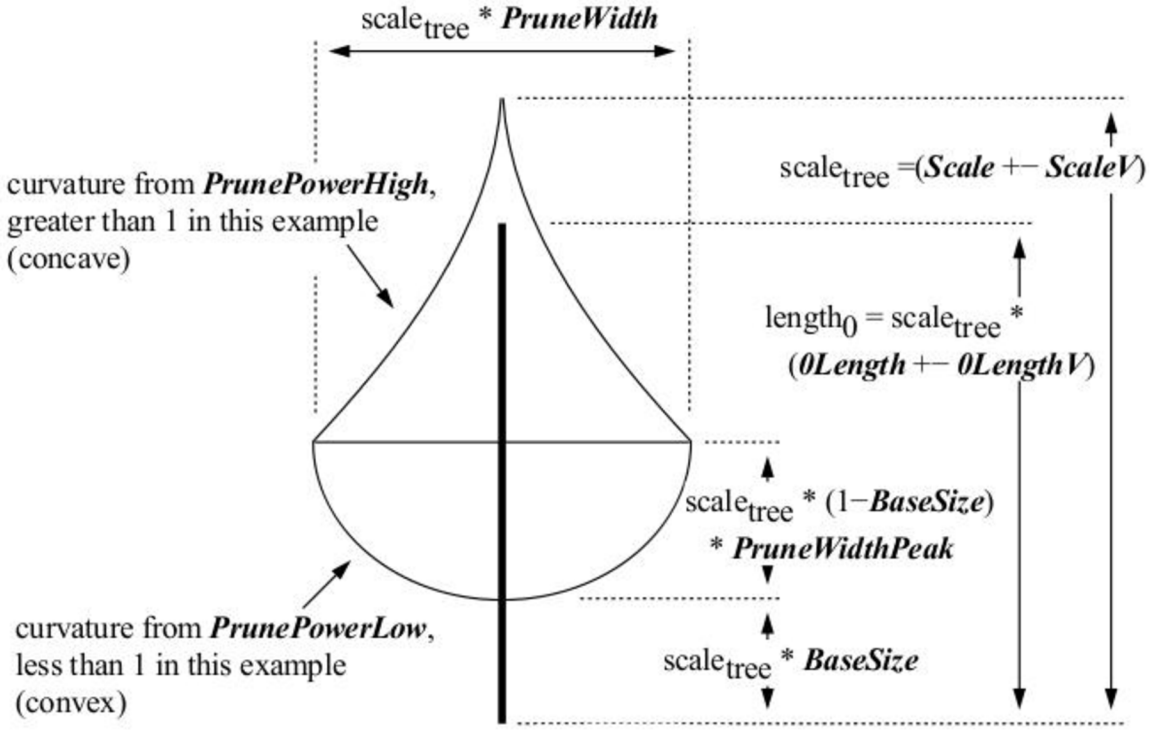
$$[ratio / (1 - \textbf{PruneWidthPeak})]^{\textbf{PrunePowerHigh}} \qquad pro\ ratio < 1 - \textbf{PruneWidthPeak}$$

$$[(1 - ratio) / (1 - \textbf{PruneWidthPeak})]^{\textbf{PrunePowerLow}} \qquad pro\ ratio \geq 1 - \textbf{PruneWidthPeak}$$

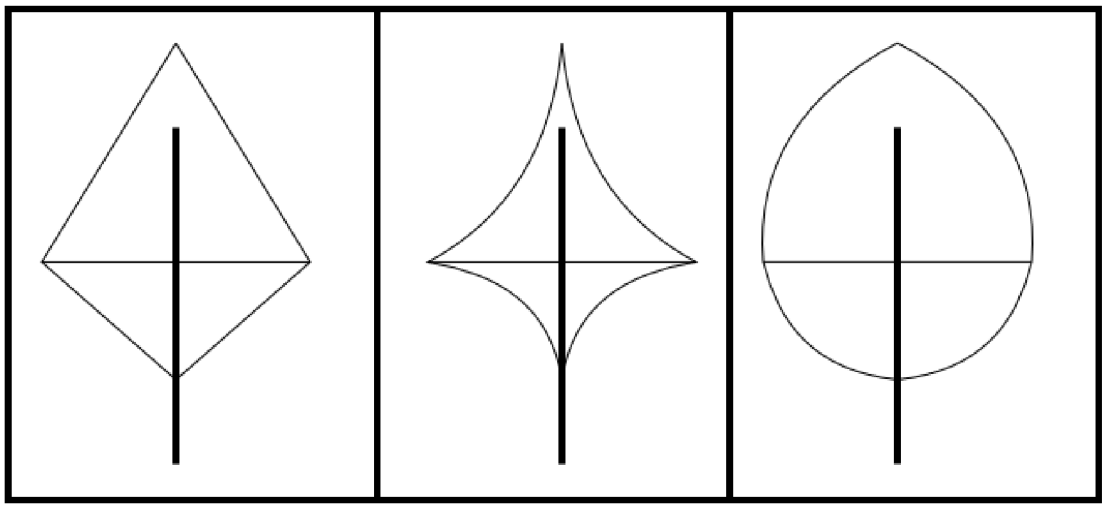
Funkce *ShapeRatio(8, ratio)* vždy vrátí hodnotu 0, pokud se hodnota ratio nepohybuje v rozsahu od 0 do 1. Při generování stromu může být použit parametr ***Shape*** = 8 i pokud není prořezávání zapnuto. Slouží to k definici vlastního typu tvaru stromu. Pokud je ořezávání zapnuto a tvar stromu bude nastaven na hodnotu 8, bude mít koruna stromu tendenci dodržet tvar daný obálkou ještě před prořezáváním.

Efekt prořezávání může být zredukován použitím parametru ***PruneRatio***, který definuje váhový průměr mezi původní a ořezanou délkou stonku. Pokud je hodnota parametru rovna 1, bude aktivováno úplné ořezávání, kdy žádná z větví nepřekročí danou obálku. Tento efekt je vhodný pokud chceme, aby stromy vypadaly uměle prořezané. Naopak hodnota 0 znamená, že je ořezávání vypnuté. Částečné prořezávání funguje na způsobu váhového průměru mezi již zkrácenou minimální délkou a původní délkou větve. Stromy vygenerované s částečným prořezáváním mají přirozenější tvar.

Efekt Prořezávání je aplikován pouze na větve, protože délka kmene spadá většinou do celkové výšky stromu.



Obrázek 8 - význam parametrů prořezávání [1]



Obrázek 9 - ukázky obálek. Zleva do prava. lineární, konkávní, konvexní

Obrázek 10 - vlevo neprořezaný strom, vpravo prořezaný lineární obálkou

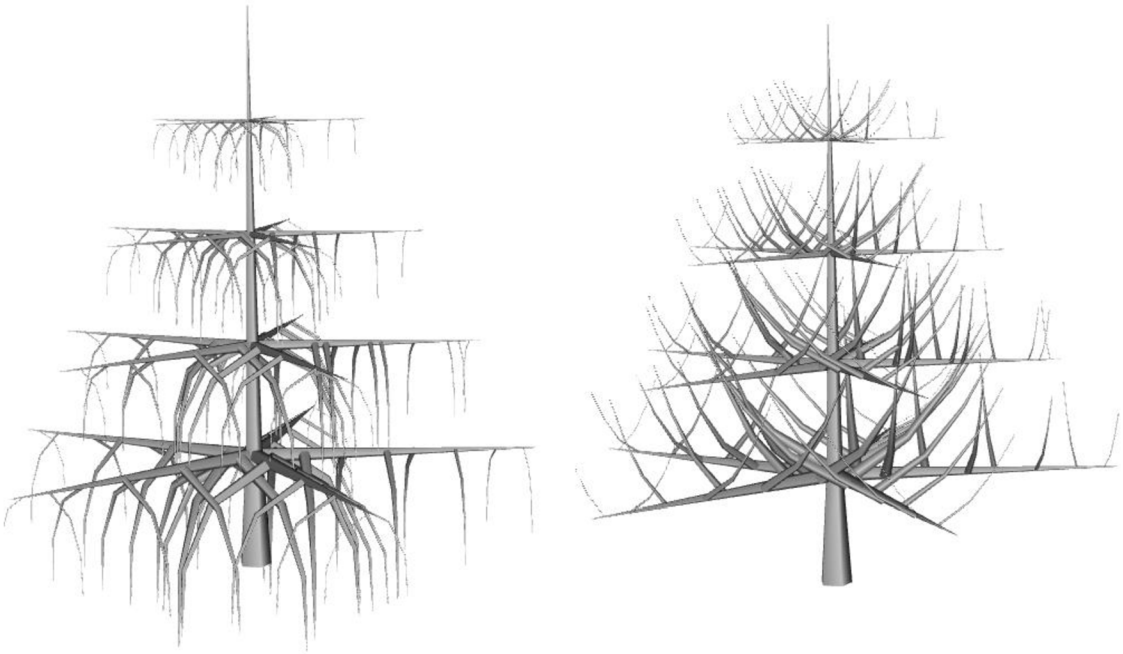
4.8 Vertikální přitažlivost

Vzhled stromů je ovlivněn také tím, že se větve snaží růst směrem nahoru k vyšší intenzitě světla. Toto je v modelu implementováno tak, že se mění zahnutí a orientace všech segmentů každé větve od druhé generace. Kmen a první generace větví tímto nejsou v algoritmu ovlivněny, protože lze jejich tvar snadno kontrolovat pomocí ostatních parametrů. Tento efekt se pak přidává k ostatním, které ovlivňují zahnutí větví. Parametr ***AttractionUp*** specifikuje tendenci větví růst nahoru. Pokud je jeho hodnota rovna 0, znamená to, že je tato možnost neaktivní. Pokud je hodnota větší než 0, mají větve tendenci stáčet se nahoru tak, aby jejich poslední segmenty mířily kolmo vzhůru. Naopak hodnota menší než 0 obstarává efekt, že větve rostou více dolů. Tento efekt růstu větví lze dobře pozorovat třeba u vrby. Výpočet zahnutí probíhá pro každý segment a výsledná hodnota se přidá k již spočtenému zahnutí segmentu. Pro simulaci tohoto jevu opět použijeme odklonění (declination) lokální souřadné *osy z* od globální a orientaci (orientation) *osy*, která míří směrem k nebi. V našem případě *osy y*.

$$declination = cos^{-1}(transformation\_z_z)$$
$$orientation = cos^{-1}(transformation\_y_z)$$

$$curve\_up\_segment = \mathbf{AttractionUp} * declination * cos\ (\ orientation\ ) / \mathbf{nCurveRes}$$

kde *transform_z* je z komponenta jednotkového vektoru spočítaná z souřadnic aktuálního segmentu a *transform_y* z komponenta téhož vektoru. Hodnota proměnné *curve_up_segment* je následně přidána k zahnutí segmentu.



Obrázek 11 - vlevo AttractinUP = -1, vpravo AttractionUp = 1

5 Implementace

Hlavním úkolem této diplomové práce bylo implementovat generátor realistických modelů stromů. Generátor implementuje algoritmus představený v předchozí kapitole. V této části představím řešení a jednotlivé třídy, které byly v implementaci použity.

Aplikace je napsána v jazyce C++, její vývoj probíhal na operačním systému Windows, ale aplikace je implementovaná pomocí platformě nezávislých knihoven, takže je přeložitelná i v prostředí Linux. Při implementaci byla použita volně použitelná knihovna OpenSceneGraph [3], která zajišťuje uchovávání geometrie jednotlivých částí stromu a její vykreslování. OpenSceneGraph je postaven na OpenGL a zapouzdřuje všechnu jeho funkcionalitu do snadno použitelných tříd. Výsledná aplikace je ovládána pomocí jednoduchého GUI, které je postavené na knihovně Qt.

Při implementaci jsem se snažil, aby navrhované objekty a třídy byly co nejjednodušší a zaměřené čistě na jeden účel. Snažil jsem se tím dosáhnout čitelného a snadno rozšiřitelného kódu.

5.1 Parametry

V této části následuje popis implementace parametrů generátoru. Rozhodl jsem se této tématice věnovat jednu kapitolu, protože práce s parametry je poměrně důležitá část jak generátoru, tak implementace. Navíc zde byl použit zajímavý přístup při návrhu výpočtových tříd, který také stojí za zmínku.

5.1.1 CParams

Základ pro práci algoritmu i generátoru je nastavování parametrů a práce s nimi. Pro uchovávání parametrů slouží třída *CParams*, která obsahuje struktury pro uchování jednotlivých skupin parametrů. Struktury s parametry odpovídají rozdělení uvedenému v kapitole 2. Pouze parametry větví a kmenu jsou sloučeny do jednoho celku, a to struktury *sLevelParams*. Toto spojení jsem provedl z toho důvodu, že z pohledu generátoru je kmen i větev představována stejnou strukturou. Třída *CParams* se stará o jejich uchování, načítání a ukládání do a ze souborů, konverzi úhlových parametrů ze stupňů na radiány a obráceně. Třída CParams obsahuje pro každou generaci stromu jednu sadu parametrů sLevelParams a jednu sadu výpočtů pro danou generaci.

5.1.2 Výpočty

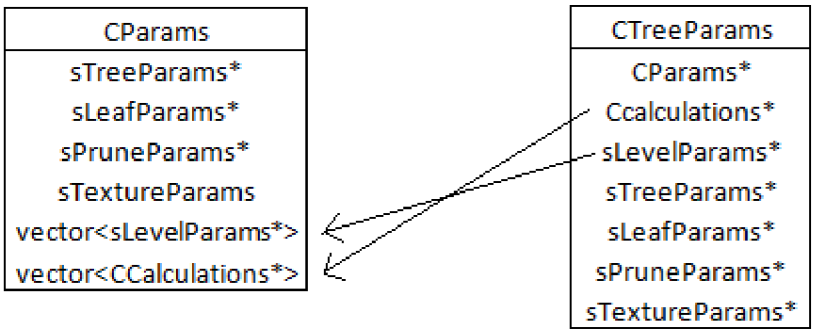
Při návrhu tříd zapouzdřujících parametry jsem došel k závěru, že výpočty úhlů odklonu, rotací, délek větví a kmenu a většina výpočtů uvedených v kapitole 3, které souvisí s parametry,

by bylo vhodné přesunout mimo vlastní algoritmus generování struktury stromu. Proto vznikla třída *CCalculations*, která se stará právě o tyto výpočty. Výpočty společné pro všechny generace jsou uloženy v této třídě. Výpočty, které jsou odlišné, jsou uloženy v potomcích třídy *CCalculations*. Pro kmen je tedy k dispozici třída *CTrunkCalc*, pro větve *C0BranchCalc* a *C1BranchCalc*.

5.1.3 CTreeParams

Pro správné získávání hodnot jednotlivých parametrů a hodnot výpočtů jsem vytvořil třídu CTreeParams. Tato třída obsahuje ukazatele na všechny struktury parametrů a zajišťuje získání správných hodnot pro generačně závislé parametry a výpočty. Na obrázku č. 12 je ukázáno, jakým způsobem závisí třída CTreeParams na třídě CParams.

Instanci této třídy obsahuje každý objekt, který vytváří strukturu stromu.



Obrázek 12 - ukázka závislosti tříd parametrů

5.2 Struktura stromu

Generování struktury stromu je nejdůležitější součástí implementace. Důležité bylo si správně navrhnout strukturu generovaného stromu, aby bylo snadné s ní pracovat v dalších částech aplikace. Při návrhu objektů jsem záměrně oddělil strukturu od následného generování meshe. Toto řešení jsem zvolil jednak z toho důvodu, abych udržel jednotlivé objekty kompaktní a čitelné, ale také proto, že to vyhovovalo následné tvorbě více modelů z jedné struktury a následnému použití v GUI.

V této části budu větve i kmen označovat anglickým názvem stem, který je použit při implementaci jednotlivých generací stromu a shoduje se s názvem třídy, která implementuje strukturu větví i kmenu, třídou CStem.

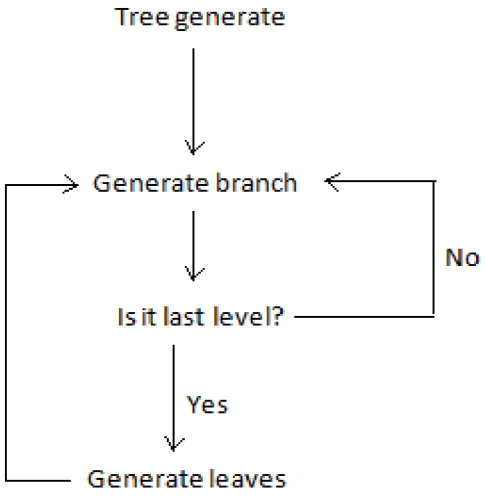
5.2.1 Strom

Strukturu stromu představuje třída *CTree*. Tato třída obsahuje ve std::vectoru uložené jednotlivé generace stromu. Každou jednotlivou větev a kmen představuje třída CStem. Třída CTree slouží jako generátor výsledné struktury. Umí vygenerovat celou strukturu stromu najednou, nebo generovat po jednotlivých generacích. Generování po jednotlivých generacích je funkcionalita, která byla potřebná pro práci GUI, kde si lze interaktivně měnit parametry jednotlivých generací a sledovat okamžitě výsledek.

Třídu CTree využívá pro následné sestavení meshe a generování 3D struktury třída TreeCreator, která bude popsána dále v této kapitole.

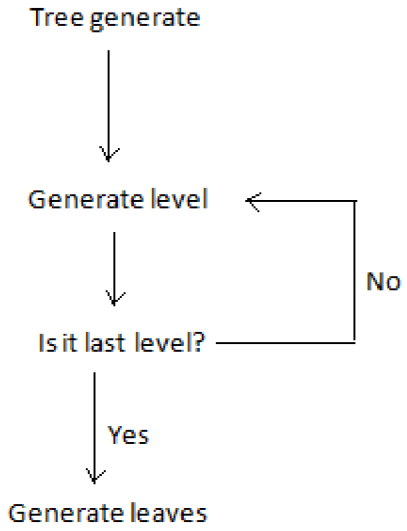
Při generování struktury mohou být použity dvě odlišné strategie. Rekurzivní přístup a přístup generování po jednotlivých generacích kmene a větví.

Rekurzivní přístup nejdříve vygeneruje kmen. Poté při generování větví vezme první a vygeneruje ji. Následuje generování větví další generace, které z této vyrůstají, až dojde k vygenerování listů. Pak se opět vynoříme z rekurze o jedno patro výš a generování pokračuje.



Obrázek 13 - schéma rekurzivního generování

Generování po jednotlivých generacích funguje tak, že dojde k vygenerování celé jedné generace větví a poté se vygeneruje generace další. Jakmile jsou vygenerovány všechny, dojde ke generování listů. Tento přístup jsem zvolil ve své implementaci, protože lépe vyhovuje práci s GUI. K rekurzivnímu generování přistoupím nejspíše při generování LOD. Protože umožní efektivnější reprezentaci struktury stromu.



Obrázek 14 - schéma generování po jednotlivých generacích

5.2.2 Základy struktury

Společným základem tříd, které tvoří strukturu stromu je abstraktní bázová třída *CGeneratorBase*. Tento interface představuje kostru pro generované entity stromu (třídy *CStem*, *CSegment*). Tato třída obsahuje virtuální metody pro vygenerování jednotlivých částí modelu, další pro vygenerování částí, ze kterých je složena a metodu pro vygenerování větví další generace. Obsahuje také ukazatel na předka, aby bylo možné traversovat strukturou stromu v obou směrech a ukazatel na třídu CTreeParams,
která obsahuje parametry pro aktuální generaci stromu.

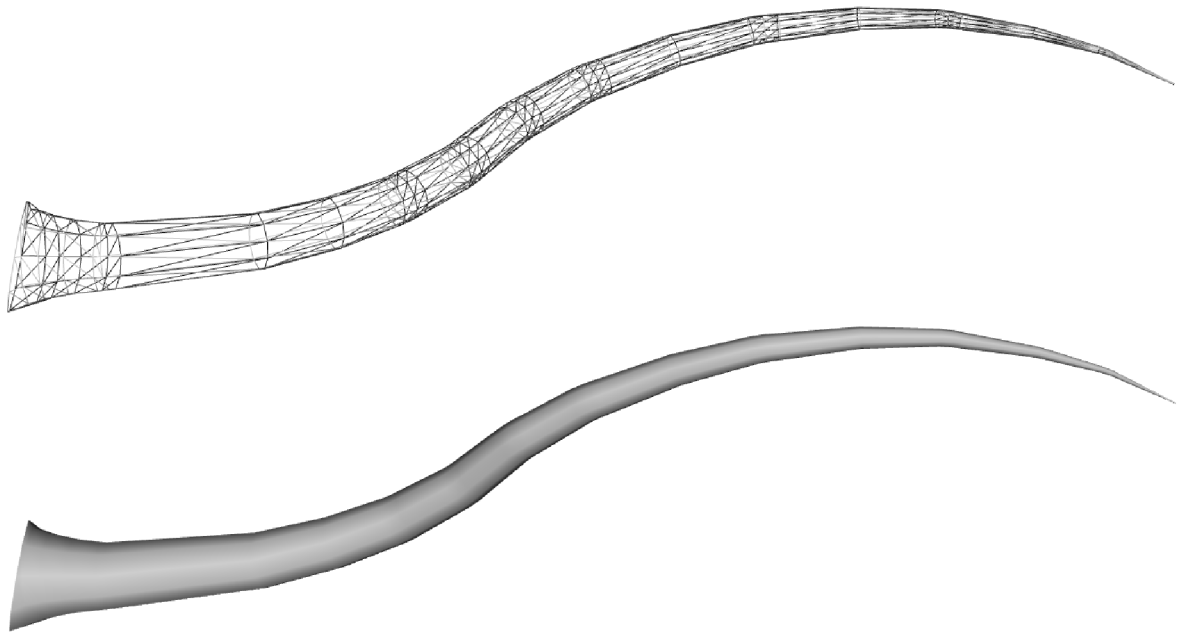
Třída *CGeneratorBase* obsahuje také základní informace o poloze jednotlivé části a základních rozměrech. Tyto informace dědí ze třídy *CGeometryBase*. Tato třída obsahuje informace o délce jednotlivé části a spodním a horním průměru. Pro správné umístění dané části stromu v prostoru obsahuje také třídu *CTransformation*, která uchovává aktuální transformační matici a polohový vektor. Transformační matice je představována třídou *osq::Quat*, která je reprezentována jediným vektorem. Jedná se o quaterion[4], který dokáže reprezentovat transformaci stejně, jako klasická transformační matice, ale zabírá méně místa v paměti. Toto řešení jsem zvolil z důvodu budoucího vývoje, kdy mám v plánu využít vertex shader pro simulaci pohybu stromu ve větru. Více informací o tomto budoucím rozšíření je v kapitole 6.

5.2.3 Stem

V modelu stromu třída *CStem* představuje větev a kmen. Tato třída dědí své vlastností přímo od třídy *CGeneratorBase*. Další dodatečné informace, které tato třída nese, jsou ty, které souvisí s parametry větví v generátoru. Jsou to tedy úhel odklonu a rotace od mateřské větve.

Vygenerování jednoho stemu probíhá tak, že je zavolána metoda *make*. V této metodě se postupně získají věechny hodnoty úhlů a délek, které jsou získány z parametrů a jejich třídy *CCalculations*. Poté je zavolána metoda *DoTheTransformationMagic*, která nejprve získá transformaci od rodičovského stemu a podle offsetu na něm si vygeneruje vlastní polohu a rotační matici. Pozice v prostoru je tedy absolutní, a pokud by došlo ke změně u rodiče, musí být stem generován znovu. Toto řešení jsem zvolil proto, že relativní transformace prakticky nemá v tomto modelu smysl. Pokud totiž dojde ke změně u rodiče, změní se i jeho tvar, a proto by relativní poloha poté neodpovídala původní.

Poté se spustí generování segmentů pomocí metody *MakeLowerLevel*, která vygeneruje podle nastavení v parametrech správný počet segmentů. Po dokončení této operace je generování stemu u konce. Pokud je ale zapnuté prořezávání, dojde ještě k volání metody Prune. Ta podle algoritmu, popsaném v kapitole o prořezávání, zkontroluje, jestli poslední bod větve nepřesahuje hranice obálky. Pokud ano, zmenší se velikost větve o určitou délku a dojde k volání metody Prune u všech jejích segmentů. Tento proces iterativně pokračuje do té doby, než větev vyhovuje a je celá obsažena v obálce.



Obrázek 15 - ukázka geometrické reprezentace stemu

5.2.4 Segment

Segment je základní jednotka, která definuje tvar větve. V nastavení parametrů se definuje, z kolika segmentů se bude daný stem skládat. Tato hodnota je uložena v parametru **CurveRes**.

Základem pro segment je třída *CSegmentBase*, která je potomkem třídy *CGeneratorBase*. Tato třída rozšiřuje vlastnosti o ukazatel na předchozí a následující segment, které jsou potřeba pro generování struktury i následné generování meshe. Třída *CSegmentBase* je základem i pro subsegmenty, které jsou reprezentovány třídou *CSubSegment*.

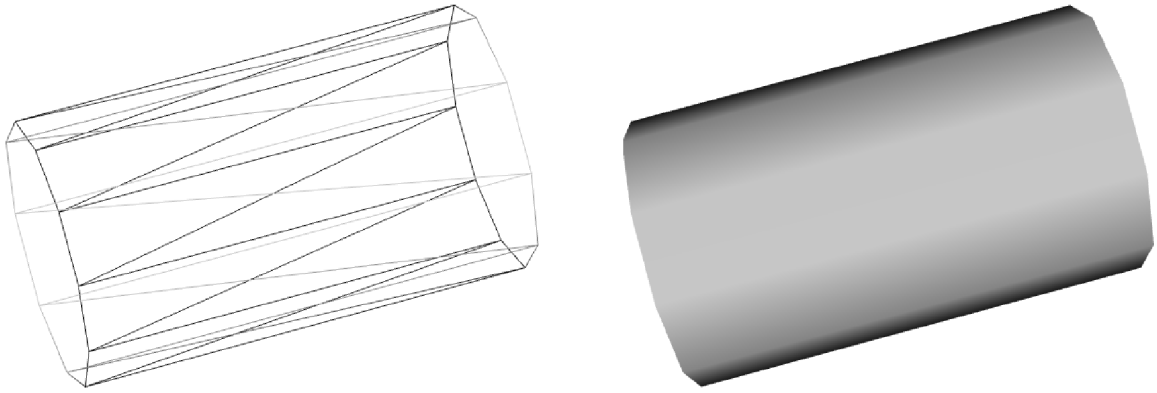
Generování segmentu probíhá podobným způsobem jako u stemu. Metoda *Make* si nejprve získá z parametrů správné hodnoty, a poté dojde ke generování segmentu. V průběhu generování ale dochází k rozvětvování, které je popsané v kapitole 4.2. Toto klonování má na starosti metoda *Clone*, která je volaná po vygenerování každého segmentu ze třídy *CStem*. Pokud došlo k rozvětvení, jsou tyto klony poté zařazeny do fronty pro generování segmentů ve třídě *CStem*.

Z toho vyplývá, že při klonování nejsou vytvořeny nové stemy, ale celá struktura originálního stemu i jeho klonů je držena v rámci klonovaného stemu. Proto pak neodpovídá počet jeho segmentů hodnotě v parametrech.

Nakonec je při generování také volána metoda *DoTheTransformationMagic*, která opět zajistí správné nastavení pozice a rotací v rámci stromu.

Segmenty také obsahují pole potomků, které představuje třída *CSubSegment*. K jejich generování ale nedochází v průběhu generování segmentů, ale probíhá až následně poté, co je vytvořena detailní struktura stromu pomocí třídy *CSmoother*. Popis tohoto vytváření jemnější geometrie je popsán dále v této kapitole.

Poté co dojde k vygenerování posledního segmentu, je zkontrolováno, jestli koncový bod je obsažen v prořezávací obálce. Pokud ano, generování segmentu tímto končí. Pokud ne, dojde k zavolání metody *Prune* mateřského stemu. Ta poté iterativně upravuje délku větve, aby vyhovoval obálce.

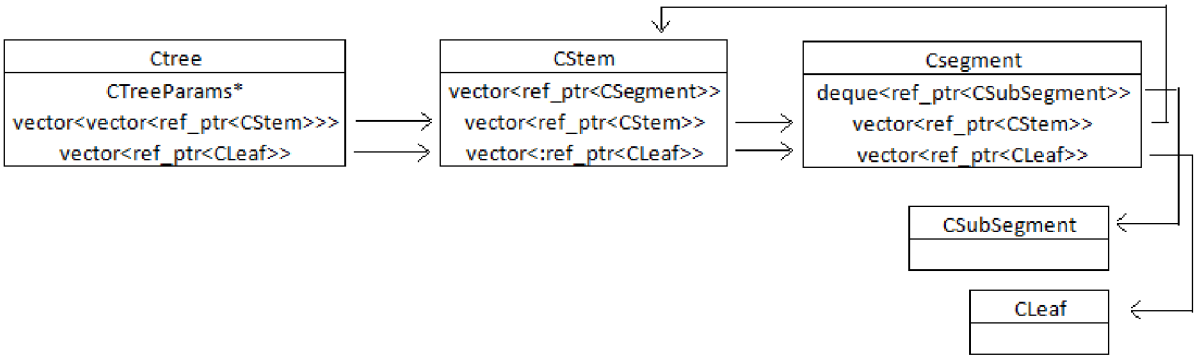


Obrázek 16 - ukázka geometrické reprezentace segmentu

5.2.5 List

Objektovou reprezentaci listu představuje třída *CLeaf*. Tato třída jako jediná z částí struktury stromu není děděná od bázové třídy *CGeneratorBase*. Třída *CLeaf* obsahuje pouze svou transformaci (*CTransformation*). Vygenerování má na starosti také metoda *Make*, která získá hodnoty z parametrů a poté zavolá metodu *DoTheTransformationMagic*, která zajistí správné umístění v rámci stromu. Poté ještě následuje úprava rotace podle postupu z kapitoly 4.6.1.

Geometrickou reprezentací listu je čtverec, na který je nanesena textura s alfa-kanálem.



Obrázek 17 - ukázka struktury stromu

5.3 Generování polygonální sítě

Geometrickou reprezentací jednotlivých částí stromu jsou ve většině případů válce, které reprezentují jednotlivé segmenty. Z nich jsou pak složeny kužely, které představují stemy. Přeměnu struktury stromu na polygonální síť jsem od implementace tříd *CStem* a *CSegment* oddělil záměrně. Generování polygonální sítě přichází na řadu až poté, co je dokončeno generování celé struktury stromu. Samotnému generování sítě ještě předchází dva kroky.

První je zjemnění struktury stromu, aby byly omezeny ostré přechody mezi segmenty, poté následuje výběr částí, které se budou zobrazovat. Tyto vybrané části jsou převedeny na polygonální síť.

Výsledná polygonální síť není generována jedna pro celý strom, ale pro jednotlivé větve. Celý strom je poté vykreslován jako po částech, které reprezentují kmen a větve.

5.3.1 Zjemnění struktury stromu

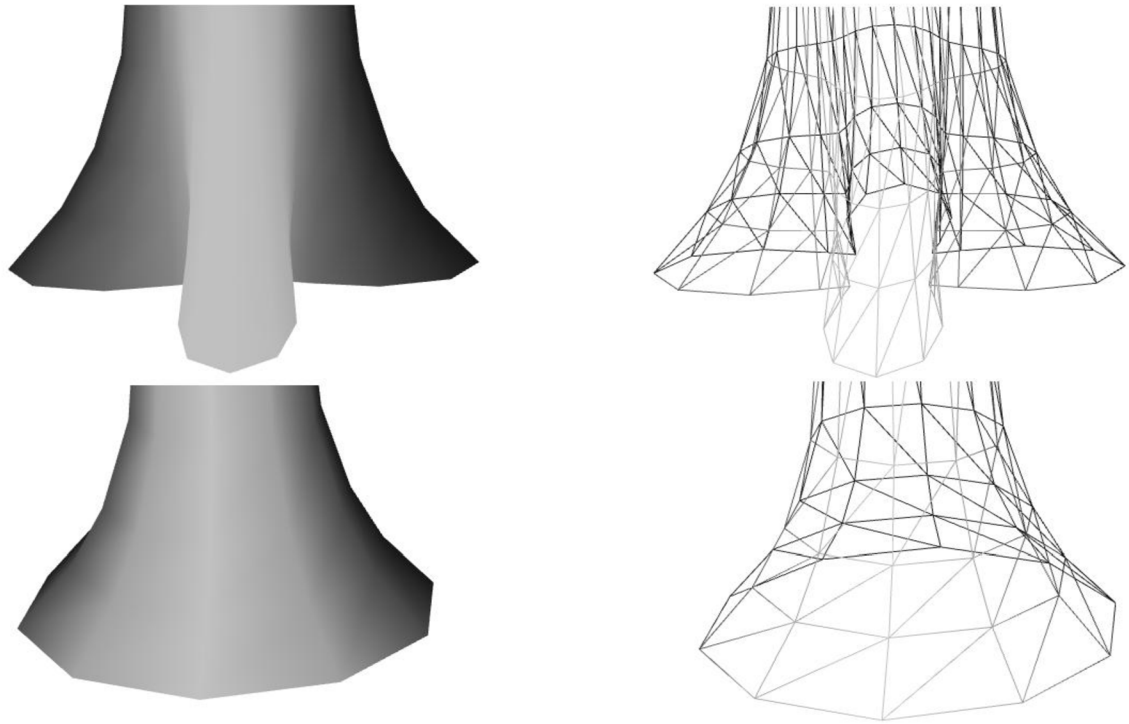
Zjemnění struktury stromu má na starosti třída *CSmoother*. Tato třída postupně prochází celou strukturu stromu a v případě, že narazí na část stromu, která potřebuje upravit, tak vezme segmenty, ze kterých se daný stem skládá a vytvoří jemnější strukturu tím, že pro ně připraví odpovídající subsegmenty. *CSmoother* momentálně upravuje strukturu segmentů, které obsahují *flare* nebo *lobes* (jen u kmenu), upravuje napojení stemů na rodičovský stem (jen pro větve) a vyhlazuje příliš velké úhly odklonu mezi jednotlivými segmenty (pro kmen i větve).

Jako budoucí rozšíření mám v plánu dodělat vyhlazování pro klonované segmenty.

5.3.1.1 Zjemnění flare a lobes

Flare a ***lobes*** jsou parametry určené jen pro kmen a reprezentují jeho exponenciální rozšíření a sinusoidální výběžky v jeho spodní části. V generujícím algoritmu je definováno, že tato rozšíření zasahují pouze do jedné osminy délky kmene. Vygenerování nové struktury má na starosti funkce *SmoothFlareAndLobes*.

Při vytváření jemnější struktury ***flare*** a ***lobes*** dojde vždy k vytvoření čtyř nových subsegmentů. Toto řešení jsem zvolil jako kompromis mezi dobrou kvalitou výsledku a zbytečně hustou polygonální sítí, která následné nepřinesla viditelný efekt. Pokud do jedné osminy délky stromu patří více segmentů, dojde k vygenerování subsegmentů do všech, které do této části kmene zasahují. Na obrázku č. 18 je ukázka výsledné jemné struktury sítě pro ***flare*** i pro ***lobes.***

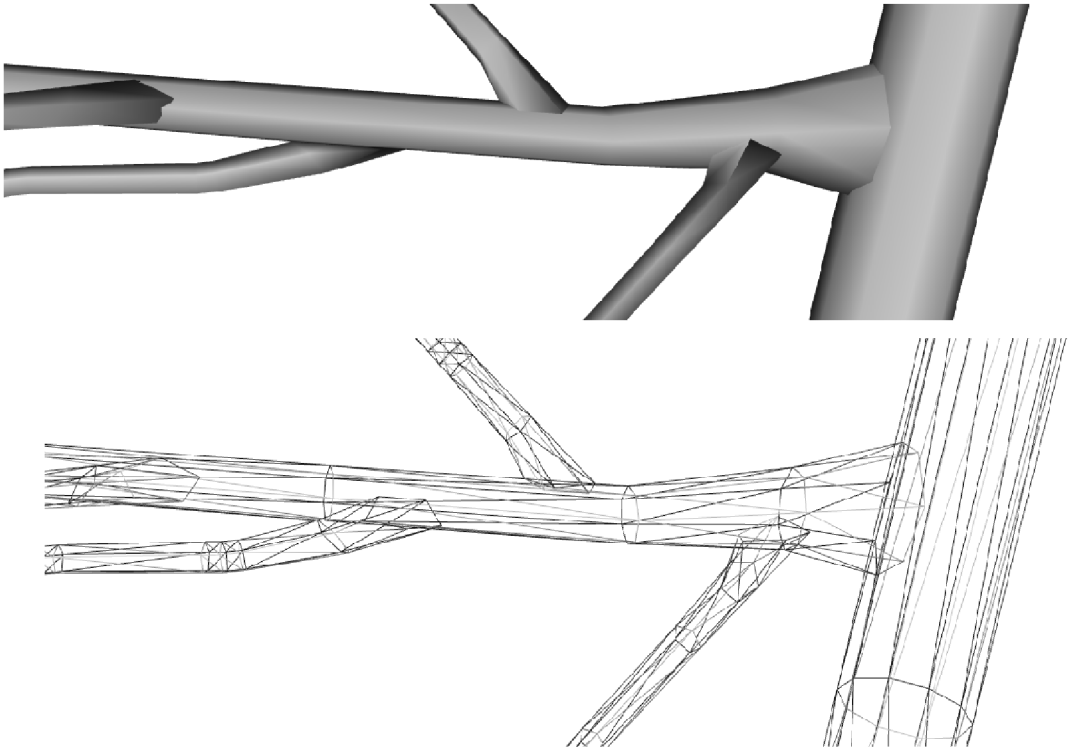


Obrázek 18 - zjemnění flare a lobes

5.3.1.2 Zjemnění napojení větví

Zjemnění napojení větví byl jeden z důležitých požadavků, které jsem si dal při vytváření generátoru. Na modelech stromů je často vidět, že větve vyrůstají bez jakéhokoliv napojení na kmen, jako by je na něj někdo jen našrouboval. Proto jsem pro zachování co možná největší realističnosti vytvořil subsegmenty, které toto napojení představují. Dosažený výsledek ještě není optimální a bylo by potřeba trochu doladit nastavení konstant, které zajišťují generování vertexů těchto subsegmentů.

Samotné generování je popsáno dále v této kapitole.

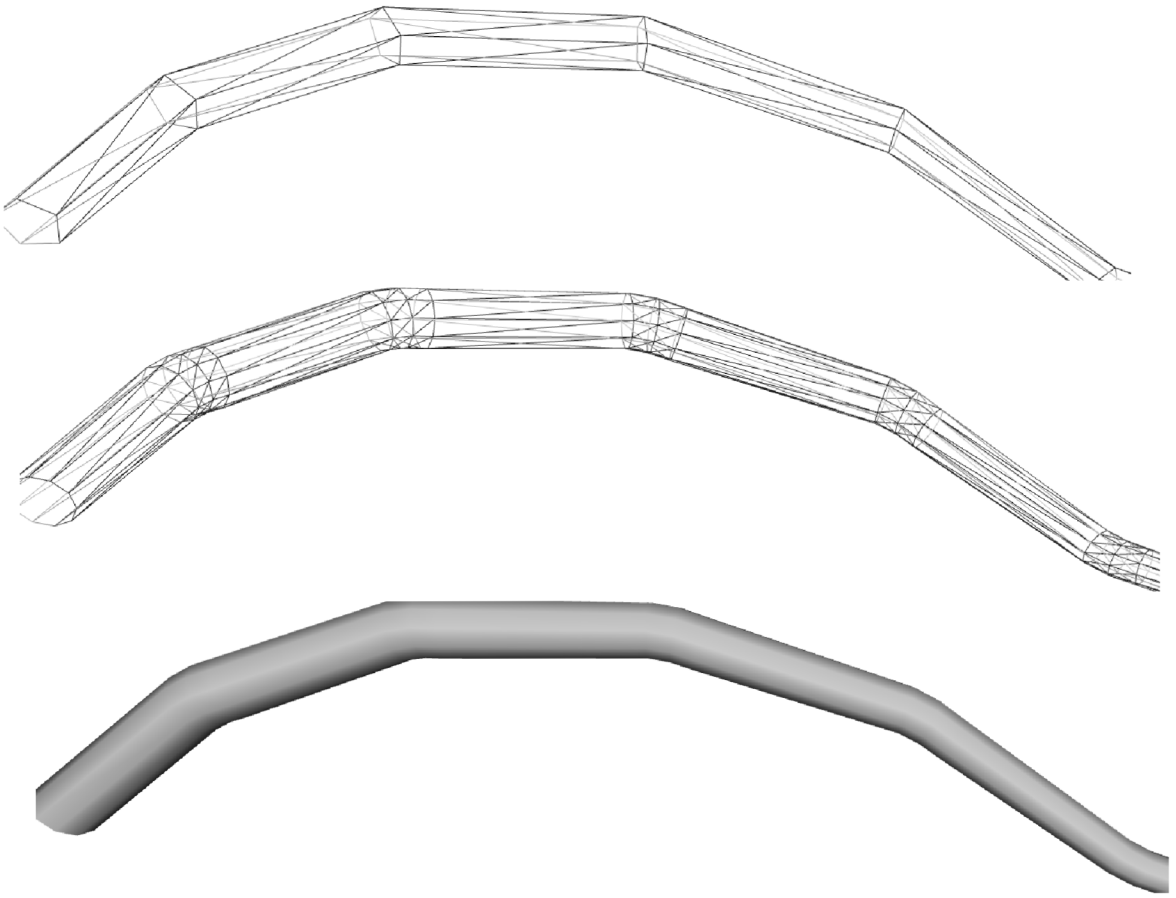


Obrázek 19 - zjemnění napojení větví

5.3.1.3 Zjemnění přechodů mezi segmenty

Původní algoritmus generování vytvářel ostré přechody mezi segmenty, které byly od sebe odkloněné o úhly větší než 20°. Tento problém jsem vyřešil přidáním subsegmentů do míst, kde je tento úhel větší než 20°. V inkriminovaném místě se poté nevytváří tak ostré hrany, což napomáhá realističnosti výsledného modelu stromu.

Zjemnění těchto velkých úhlů má na starosti funkce *SmoothCurveAngles*, která postupně prochází jednotlivé segmenty stemu a kontroluje, jestli není úhel větší než povolené maximum. Pokud dojde překročení tohoto úhlu, je každému z těchto dvou segmentů přidán na konec, respektive začátek přidán jeden subsegment.



Obrázek 20 - zjemnění přechodů mezi segmenty

5.3.2 Výběr částí pro generování polygonální sítě

Pro výběr částí, které se nakonec budou generovat, slouží třída CSimplyfier. Tento mezikrok jsem přidal pro budoucí rozšíření pro generování Level Of Detail modelu.

Momentálně CSimplyfier obstarává výběr segmentů pro plný model a pro jednoduchý model, který slouží, pro rychlou interakci mezi v GUI při úpravě parametrů.

Výběr těchto segmentů obstarává funkce PrepareParts, která projde daný stem a vrátí pole částí, které budou zobrazovány. V případě použití jednoduchého modelu vrátí pouze segmenty daného stemu. Pokud se však jedná o plný model, funkce PrepareParts projde jednotlivé segmenty a zkontroluje, jestli je složený z nějakých subsegmentů. Pokud ano, tak do vraceného pole přidá tyto subsegmenty. Pokud ne, tak do pole přidá pouze segment samotný.

CSimplyfier poté ještě každé částí, která se bude generovat, přidá informaci, z kolika bodů bude vygenerována. Tyto informace jsou uloženy ve struktuře *sPartsToVertices*, která je poté předána generátoru bodů.

5.3.3 Generování vertexů polygonální sítě

Nyní se dostáváme k samotnému generování polygonální sítě. Nejdříve jsou vytvořeny jednotlivé body a texturovací souřadnice, které jsou následně předány třídě *CMeshGenerator*. Vygenerování bodů pro části připravené třídou *CSimplyfier* má na starosti třída *CPointGenerator*.

Tato třída postupně prochází pole částí a generuje pro ně vrcholy, normály k daným vrcholům a texturovací souřadnice. Výsledné body, jejich normály a souřadnice následně ukládá do struktury *sVerticesForMesh*, která je poté předána třídě *CMeshGenerator*, která už zajišťuje generování výsledné polygonální sítě.

Generování texturovacích souřadnic probíhá až poté, co jsou vygenerovány body pro jednotlivé části. Nastavení, kolikrát se bude textura opakovat po obvodu i po délce stemu si bere z parametrů.

Generování jednotlivých bodů mají na starosti funkce *GenerateCrossSection*, *GenerateCrossSectionEllipsis*,*GenerateCrossSectionLobes*,*GenerateCrossSectionBranchConnection*.

Přípravu transformací a vstupů pro tyto funkce zajišťují funkce *GeneratePartNormal*, *GeneratePartLobes*, *GeneratePartKnee* a *GeneratePartBranchConnection*. Pokud se generuje první část, jsou vygenerovány body pro spodní průřez i pro vrchní průřez. Pro všechny další části se generuje už jen vrchní průřez.

5.3.3.1 Generování normálních bodů

Generování normální části má na starosti metoda *GeneratePartNormal*, která pouze vezme transformaci a průměr jednotlivé části a vygeneruje správný počet bodů. Body jsou v tomto případě generovány na kružnici a poté transformovány na své místo. Generování bodů zajišťuje funkce *GeneratePartNormal*.

5.3.3.2 Generování bodů pro lobes

Generování této části zajišťuje *GeneratePartLobes*, která také jen vezme transformaci pro danou část a volá metodu *GenerateCrossSectionLobes*. Navíc ještě upravuje počet bodů pro jednotlivé sinusoidové záhyby. Momentálně je to 8 bodů pro jeden záhyb, ale tato hodnota se v další fázi bude odvíjet od toho, jakou část LOD modelu bude potřeba generovat.

Funkce *GenerateCrossSectionLobes* funguje podobně jako *GenerateCrossSectionNormal*. Generuje body na kružnici a přidává k nim hodnotu ***lobesDepth*** sin(**lobes** * angle),* kde *angle* je úhel, pro který probíhá generování bodu.

5.3.3.3 Generování bodů pro napojení větví

Generování zajišťuje *GeneratePartBranchConnection*. Tato funkce připraví koeficienty pro generování elipsovitého průřezu a připraví speciální transformaci pro generování prvních bodů. Tyto koeficienty slouží k vygenerování průřezu s větším průměrem, než je původní průměr větve. Toto slouží k vytvoření realističtějšího napojení. Funkce *GenerateCrossSectionBranchConnection* poté vygeneruje jednotlivé body na elipse a přitiskne je k povrchu mateřskému stemu. Tím vytvoří napojení, větve na kmen. V dalším kroku dojde k vytvoření mezistupňového přechodu mezi originálním průměrem větve průměrem napojení.

5.3.3.4 Generování bodů pro zmenšení úhlů mezi segmenty

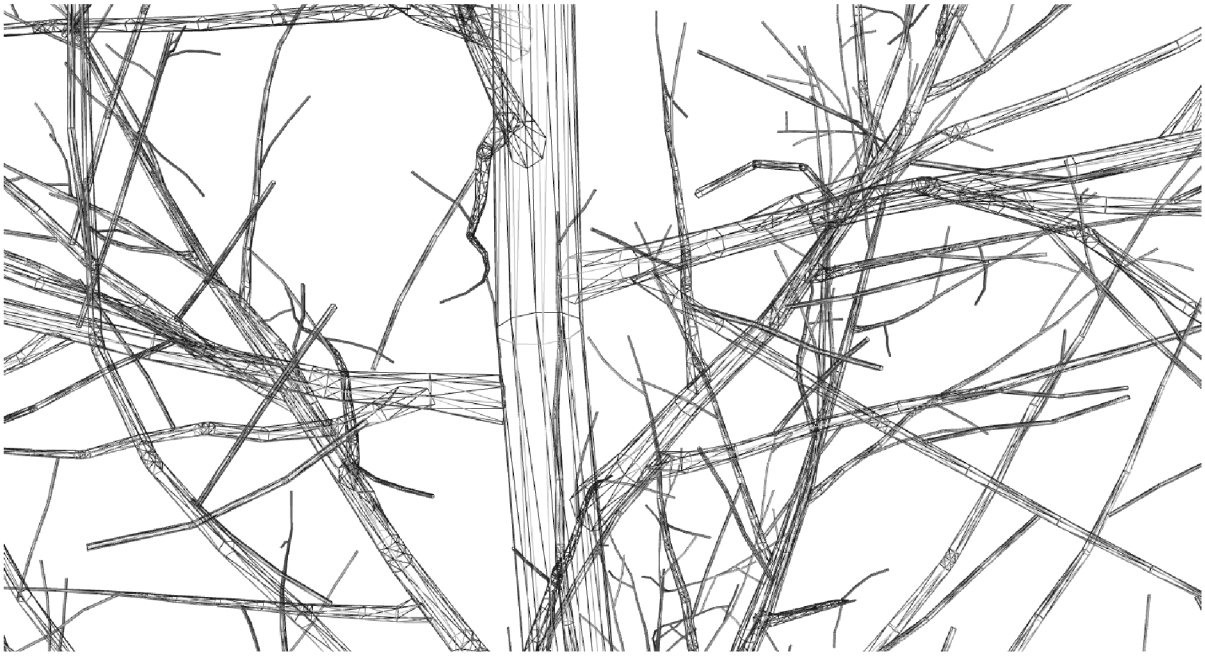
Generování probíhá ve funkci *GeneratePartKnee*, která musí vytvořit speciální transformaci, která je pouze poloviční oproti původnímu odklonu segmentů. Tento mezistupeň je generován jako elipsa, aby nedošlo k deformaci polygonální sítě. Samotné body se poté generují funkcí *GenerateCrossSectionEllipsis*.

5.3.4 Generování polygonální sítě

Vstupním bodem pro generování bodů je struktura *sVerticesForMesh*, která obsahuje seznam částí, pole vertexů, texturovací souřadnice a normály jednotlivých vrcholů. Samotné generování zajišťuje třída *CMeshGenerator*. Tato třída generuje výslednou síť pomocí objektů OpenSceneGraphu osg::Geometry, který zapouzdřuje volání OpenGL příkazů. Celá geometrie je složena z objektů osg::DrawElementUint. Tyto objekty obsahují indexy do pole vertexů, a zapouzdřují typ GL_TRIANGLE_STRIP, které je uložené v osg::Geometry.

Samotné generování těchto indexů probíhá dvěma způsoby. V prvním případě, kdy je počet vrcholů pro horní a dolní průřez stejný, dojde pouze sekvenčně k uložení indexů pro vytvoření triangle stripu. V druhém případě, kdy je počet vrcholů dolního a horního průřezu různý, se musí indexy vybírat tak, že např. na jeden bod horního průřezu připadají dva body dolního. Generování sítě má nastarosti funkce *GetStemMesh*.

Po dokončení generování jednoho stemu se zkontroluje, jestli má nějaké listy. Pokud ano, zavolá se funkce *GetLeavesForStem*, která vygeneruje listy. Listy jsou představovány jedním GL_QUAD, a je na něj nanesena textura listu.



Obrázek 21 - ukázka výsledné polygonální sítě

5.4 GUI

GUI generátoru je postaveno na knihovně Qt. Pro tuto možnost jsem se rozhodl jednak z důvodu přenositelnosti kódu, ale také kvůli snadnosti implementace a výhodné rozšiřující vlastnosti. Qt má také podporu pro přímé vykreslování OpenGL do okna a dobře spolupracuje s knihovnou OpenSceneGraph a jejím prohlížečem osgViewer.

GUI jsem navrhl takovým způsobem, aby bylo možné okamžitě se změnou hodnoty jakéhokoliv parametru sledovat, jaký vliv to bude mít na výsledný model stromu.

5.4.1 Nastavování parametrů

Hlavní součástí GUI je záložka s parametry. Ta umožňuje editovat všechny parametry stromu. Při změně parametrů dojde k regenerování jen té části stromu, ke které parametry náleží. Pokud se změní parametry pro větve třetí generace, dojde pouze ke změně struktury této generace. Jestliže se však změní parametry druhé generace, dojde k regenerování i u třetí generace, protože poloha a tvar těchto větví závisí na generaci předchozí. To znamená, že pokud dojde ke změně u parametrů pro strom, nebo pro kmen, dojde ke generování struktury celého stromu. Ke změně celé struktury dojde také při změně parametrů pro texturování a prořezávání.

V záložce parametrů texturování lze načítat texturu pro listy a pro kmen a větve.

5.4.2 Funkce

Při editaci parametrů GUI zobrazuje jen jednoduchý model stromu, jehož geometrie je pouze orientační. Tvar větví a kmene je stejný jako u plného modelu, jen není tak detailní a větve nejsou správně napojeny na své rodiče.

Tlačítko FullModel vygeneruje plný model stromu. Poté, co je vygenerován, je možné ho uložit do podporovaných formátů. Tlačítko Wired přepíná mezi drátovým a plným modelem.

Jakékoliv generování ať už plného nebo jednoduchého modelu probíhá na pozadí ve vlastním vlákně. Tento separátní běh zajišťuje třída *TreeCreatorThread*, která se pomocí signálů synchronizuje s GUI aplikace, zajišťuje update parametrů a generování stromu nebo jednotlivých generací větví.

Tlačítko Generate spustí generování celého modelu od začátku.

5.4.3 Podporované formáty

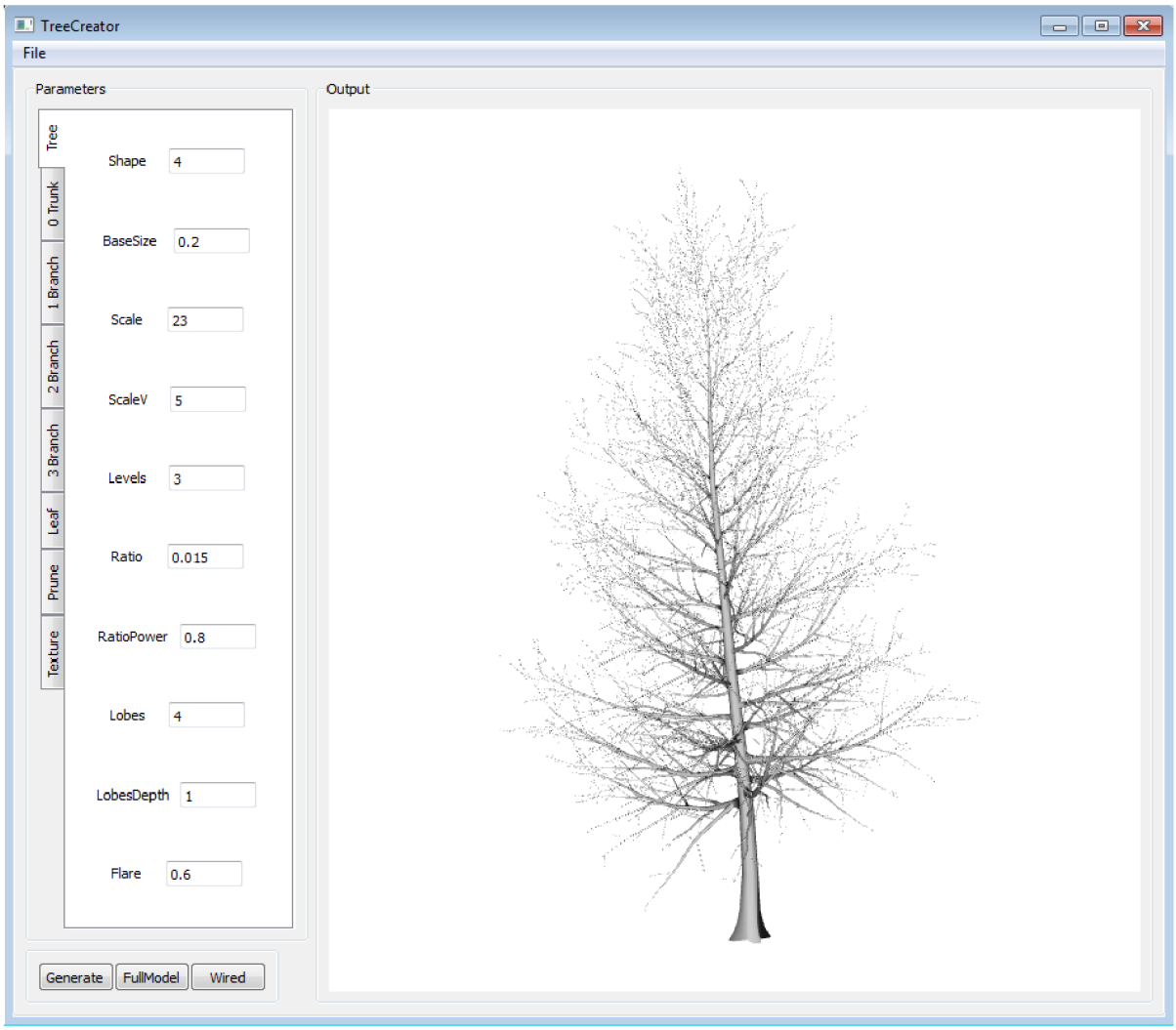
V menu je možné ukládat a načítat soubory s parametry. Tyto soubory mají příponu **.tree*. Jsou to obyčejné textové soubory s jinou koncovkou. Umožňují snadnou editaci parametrů bez nutnosti otevírat GUI

Výstup generátoru je možné ukládat do souboru OpenSceneGraphu **.osg*, do formátu 3DSMax *.3ds, do formátu programu PowRay *.pow a do formátu LightWaveObject *.lwo. Ukládání

do souborů zajišťuje OpenSceneGrahp a jeho knihovna osgDB. Při ukládání do souboru .3ds jsem nenarazil na žádné problémy, ale ukládání do .pow souboru občas trvalo delší dobu.

5.4.4 Ovládání výstupu

Výstup je zajišťován pomocí OpenSceneGraph a její knihovny osgViewer, která zobrazuje výslednou 3D scénu. Ovládání modelu je možné pohybem myši. Při stisku levého tlačítka dochází k rotaci modelu. Pravé tlačítko ovládá zoom, který je možné vyvolat i otáčením kolečka. Při stisku obou tlačítek je možné měnit pozici modelu. Stisknutím klávesy mezerník dojde k vycentrování modelu.



Obrázek 22 - GUI generátoru

5.5 Návrhy možných vylepšení

Vylepšení tohoto generátoru je celá řada. První z nich je vylepšení samotného algoritmu a jeho rozšíření o nové věci. Například by bylo dobré přidat podporu pro generování květů a plodů a rozšíření generátoru o jehličnaté stromy.

Další vylepšení se týká listů. Model momentálně podporuje pouze jednoduché listy, takže by bylo dobré přidat podporu pro listy, které vyrůstají po dvojící nebo do vějíře.

Další rozšíření, které mám v plánu implementovat, je generování level of deail modelů a využití grafických shaderů pro generování stromu.

Dále by bylo vhodné přidat podporu pro animaci stromu působením větru. Tuto část jsem měl společně s LOD rozpracovanou, ale z důvodu nedostatku času jsem ji nestihl dokončit do finální verze.

Jako poslední vylepšení bych rád generátor vylepšil o podporu DirectX.

6 Závěr

Cílem této práce bylo vytvoření generátoru realistických modelů stromů. Implementované řešení splňuje toto zadání a poskytuje poměrně kvalitní generátor modelů.

Při studiu a implementaci jsem se seznámil s novými trendy v počítačové grafice, z nichž některé jsem aplikoval již při samotném vývoji a některé budu moci uplatnit při vylepšování tohoto generátoru.

Vývoj této aplikace odevzdáním této práce nekončí. V předchozí kapitole jsem uvedl několik rozšíření, které bych k modelu rád doimplementoval.

Literatura

[1] Jason Weber, Joseph Penn: Creation and rendering of realistic trees
International Conference on Computer Graphics and Interactive Techniques, New York USA, 1995

[2] Przemyslaw Prusinkiewicz, Aristid Lindenmayer: The Algorithmic Beauty of Plants, Springer, 1990

[3] Open Scene Graph, repase 2.9.11 – graphic toolkit, 2011
Dostupné z: http://www.openscenegraph.com

[4] http://en.wikipedia.org/wiki/Quaternion

Ukázky výstupů