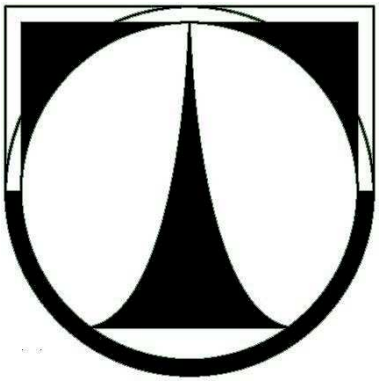
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií



Diplomová práce

Virtuální ekosystém

Liberec 2011 Petr Fabián

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N 2612 – Elektronika a informatika

Obor: Informační technologie

Virtuální ekosystém

Virtual ecosystem

Diplomová práce

Autor: Petr Fabián

Vedoucí práce: Ing. Jiří Hnídek

V Liberci 20.5.2011

ORIGINÁL ZADÁNÍ

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména §60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že s o u h l a s í m s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum:

Podpis:

1 Poděkování

Chtěl bych poděkovat vedoucímu diplomové práce Ing. Jiřímu Hnídkovi za cenné připomínky k fungování vytvořeného skriptu, za užitečné rady a nasměrování při jeho programování.

Dále bych chtěl poděkovat mému učiteli biologie, Mgr. Miloni Podoubskému, který mi na střední škole pomohl nabýt znalosti, které i po letech umožnily vytvořit základní pravidla růstu rostliny.

Abstrakt

Cílem této diplomové práce je na základě znalostí o programech sloužících k vytváření modelů rostlin vypracovat vlastní simulátor růstu rostliny.

Simulátor bude implementován do programu pro 3D[1] grafiku Blender prostřednictvím skriptu v jazyce Python.

Kromě generování stromové struktury a její vykreslení do 3D prostoru by měl simulátor reagovat na vnější podněty, které reprezentují běžné faktory ovlivňující růst živé rostliny.

Abstract

Aim of this thesis is using knowledge about programms generating trees in 3D to design new simulator to simulate and animate growing plant.

Such simulator will be implemented into 3D graphics aplication Blender as a Python script.

The program shall generate tree structure and model it, but also react to aspects of the surrounding world same as a real plant.

Klíčová slova

simulátor růstu rostliny, virtuální ekosystém, 3D grafika, Blender

Keywords

plant growth simulator, virtual ecosystem, 3D graphics, Blender

[1] trojrozměrný

Obsah

Seznam obrázků

Slovník znaků, symbolů

3D	trojrozměrný, prostorový
open-source software	software s veřejně dostupnými zdrojovými kódy
vertex	bod, vrchol trojrozměrného objektu

0 Úvod

S rostoucím využitím počítačové 3D grafiky, spojené se zvyšující se výpočetní kapacitou počítačů, rostou neustále požadavky na kvalitu výstupů z grafických programů.

Jsou pryč doby, kdy byla důležitá pouze myšlenka a její počítačové zpracování se muselo přizpůsobit omezeným možnostem počítačů. Ať už jde o film nebo ukázku nějaké stavby, nestačí naznačit myšlenku a spoléhat se na to, že zákazník (ať už je to investor či člověk kupující zboží nebo službu) bude trávit mnoho času ve snaze zorientovat se, co změť čar a různobarevných ploch znamená a jak by mohl výsledek vypadat. Stejně tak v počítačem animovaných filmech je dnes nutné diváka zaujmout realističností a věrohodností.

Přes to, že v současné době existují různé programy pro modelování a následnou vizualizaci, které nabízejí více či méně fotorealistické výstupy, je i nadále nezbytné mít kvalitní model.

Přestože je možné prakticky cokoli vymodelovat ručně a stejným způsobem i animovat, v řadě případů by takováto práce byla příliš zdlouhavá a málo efektivní. Za tímto účelem existují v lepších 3D programech různé nástroje, které umožňují automatizaci těchto úkonů, čímž šetří drahý čas grafika.

Jedním z těchto případů je i vytvoření modelu stromu. To je díky značné členitosti samotného kmenu velmi náročné na čas i umění grafika, zároveň vyžaduje jistou kombinaci mezi náhodností a deterministočností danou vlastnostmi daného druhu stromu. Vzhledem ke skutečnosti, že většinou je potřeba v modelu použít stromů více, nastalo by u ručního modelování mnoho problémů jak s časem, který grafik touto prací stráví, a dále pak s jejich originalitou, aby nedošlo k patrnému kopírování některých tvarů.

V současné době existuje mnoho programů určených ke generování stromů, ale vesměs jde o drahé specializované programy [9].

Pro stále populárnější Blender existují drobné nástroje pro tvorbu stromu, od skriptu pro olistění připraveného kmenu až po velmi silný generátor L-System [7], ale doposud neexistuje žádný simulátor růstu.

1 Představení problematiky rostlin ve 3D scéně

1.1 Využití rostlin ve 3D programech

Od svého vzniku našla počítačová 3D grafika využití v celé řadě odvětví, od architektury, přes různé ilustrace v periodikách až po kinematografii.

Protože možnosti programů pro počítačovou grafiku neustále rostou, jsou tyto programy ve fázi finální vizualizace schopny produkovat již víceméně fotorealistické snímky. Současně s tím roste i potřeba vytvářet stále kvalitnější modely různých přírodních útvarů.

Jedním z příkladů takovýchto modelů mohou být právě rostliny, hlavně stromy. Jejich využití je velmi široké, od scén budov, k nimž patří nějaké zeleň v jejich těsné blízkosti, až po velké venkovní stavby, jako jsou silniční úseky, parky a podobně.

1.2 Tvorba rostlin ve 3D scéně

Protože v současné době často nestačí pouze znázornit strom siluetou a vložení plošného obrazu stromu stačí maximálně pro vzdálené stromy, velmi často je potřeba vytvořit plnohodnotný 3D model stromu. Patrné je z obrázku 1, přes použití kvalitních textur je při pohledu z blízka patrné, že stromy jsou značně neplastické.

Ruční modelování je ale velmi náročné nejen na čas, ale i na schopnosti toho, kdo strom modeluje. Samotný kmen se často bohatě větví, což vyžaduje vytvoření velkého množství objektů, reprezentující tyto větve. Nejde ale jen o vysoký počet, stejně důležité je i dbát na jejich rozložení tak, aby strom působil rovnoměrně, byl dost košatý a nepůsobil příliš uměle. Následující doplnění stromu o listy by zabralo mnoho dalšího času úplně zbytečně.

Z tohoto důvodu je výhodné používat různé programy nebo alespoň nástroje, které tuto práci usnadňují.

V současné době existuje řada profesionálních programů, které disponují nástroji pro

Obrázek 1: Vizualizace exteriéru s rostlinami

generování vegetace. Ty nabízejí více či méně kvalitní nástroje pro tvorbu stromů a často i velmi vydařené vizualizace (obr. 2), ale jejich zásadní slabinou je samotná práce ve 3D, k níž nabízejí často velmi slabé nástroje, nebo absence možnosti exportu do jiných grafických formátů [9].

Důležitou nevýhodou všech generátorů je neschopnost reagovat na okolí, což simulátory dovolují. Hlavním rozdílem mezi generátorem a simulátorem spočívá právě ve schopnosti reagovat na množství prostoru, světla či další vnější faktory, stejně jako na vnitřní faktory, jako mohou být dostatek živin a zdraví jednotlivých větví.

1.3 Existující simulátory rostlin

Přes různé snahy vytvořit simulátory růstu stromu se více prosadil snad pouze program Ivy Generator [8].

Obrázek 2: Ostrov s palmou v programu Vue

Ten umožňuje importovat scénu vytvořenou v jiném 3D grafickém programu a v několika krocích simulovat růst břečťanu.

Další simulátory se většinou týkají různých profesionálních nástrojů, jako jsou Maya nebo 3D Studio [9].

1.4 Generování rostlin v Blenderu

Pro samotný Blender v současné době neexistuje dostatečně vypracovaný simulátor automatizaci tvorby stromů.

Jedním z nejjednodušších postupů je vytvoření plochy, na níž je nanesena textura stromu, doplněná o alfa kanál[2]. Tento způsob je snadno realizovatelný a výpočetně nenáročný. Na druhou stranu může být obtížné pořízení kvalitní textury, ačkoli je na internetu možné sehnat řadu textur různých stromů. Hlavními slabinami tohoto postupu jsou velmi plochý výsledek, který se projeví při bližších pohledech.

O něco lepších výsledků lze dosáhnout ručním vytvořením modelu stromu a následným využitím skriptu pro jeho olistění. Tento postup již vede na věrohodnější model při lepším ručnímu vytvoření kmenu a zároveň výrazně usnadňuje práci automatizací olistění, které by v případě, že by jej měl vykonávat člověk, trvalo neúměrně dlouho. Nicméně stále tento postup klade na autora modelu vysoké nároky.

[2]složka určující průhlednost pixelu

2 Cíl práce

Cílem diplomové práce je vytvořit pro aplikaci Blender skript v jazyce Python, který bude simulovat růst rostliny. Kromě možnosti exportovat hotovou rostlinu do jiného programu, což je záležitost samotného Blenderu, je požadována možnost využít i animaci samotného růstu.

Jak vychází z požadavků na simulátor stromu, růst rostliny z jisté části musí záviset na náhodě, aby dva stromy stejného nastavení byly dostatečně odlišné a při použití v jedné scéně nepůsobily dojmem kopií. Na druhou stranu je potřeba, aby do velké míry byl růst ovlivnitelný důležitými parametry, které umožní tvořit stromy stejného druhu.

Za tímto účelem je potřeba stanovit důležité faktory a určit co a jakým způsobem ovlivní.

Strom musí reagovat na vnější faktory, jako je slunce, pomocí jasně definovaných parametrů pevně daným mechanismem. Je potřeba mít kontrolu nad tím, do jaké míry který faktor ovlivní růst rostliny a do jaké míry bude ovlivňovat jeho růst náhoda. Dále musí strom reagovat na sebe sama tak, aby nedocházelo ke křížení větví.

Původní představa animace růstu se opírala o využití animačních nástrojů Blenderu, které umožňují měnit velikost jednotlivých objektů a tyto změny přiřazovat jednotlivým snímkům, což by umožňovalo vytvoření animace a její zpětné prohlížení a vizualizaci.

Od této představy jsme po konzultaci upustili, protože animování samotné velikosti jednotlivých kusů větve by vedlo k nutnosti zachovat poměr mezi začátkem a koncem každého segmentu větve. Ve výsledku by pak nebylo možné plnohodnotně uplatnit specifické vlastnosti větví, které mají mít vliv na délku a tloušťku každého úseku větve.

Ačkoli by tento problém šlo řešit použitím jiných nástrojů, jejich použití by bylo výrazně náročnější. Proto jsme upustili od možnosti klíčování animace jako takové a rozhodli jsme se vytvořit animaci pouhým vykreslením scény požadovaných kroků.

3 Návrh řešení

Protože výsledkem této práce je vytvoření komplexního skriptu, byla celá práce rozdělena do několika vývojových etap tak, aby bylo možné plánovat menší úkoly a snáze sledovat průběžný vývoj.

3.1 Vytvoření stromové struktury a práce s ní

Vzhledem k potřebě silného aparátu, který by dokázal správu a snadný přístup ke všem parametrům takřka neomezeně rostoucího stromu, využívá skript simulátoru možností objektově orientovaného programování, které je v jazyce Python velmi dobře podporováno.

S přihlédnutím ke struktuře skutečného stromu byl deklarován objekt segment, který odpovídá kusu stromu mezi dvojicí větvení. Taktovému objektu jsou v průběhu programování doplňovány jeho vlastnosti a funkce.

V počátku má každý segment proměnné *věk* a *potomci*, což je pole všech segmentů, které z něho na jeho konci vyrůstají. V průběhu programování zamýšlím podle potřeby vkládat další vlastnosti reflektující potřeby skriptu pro simulaci růstu.

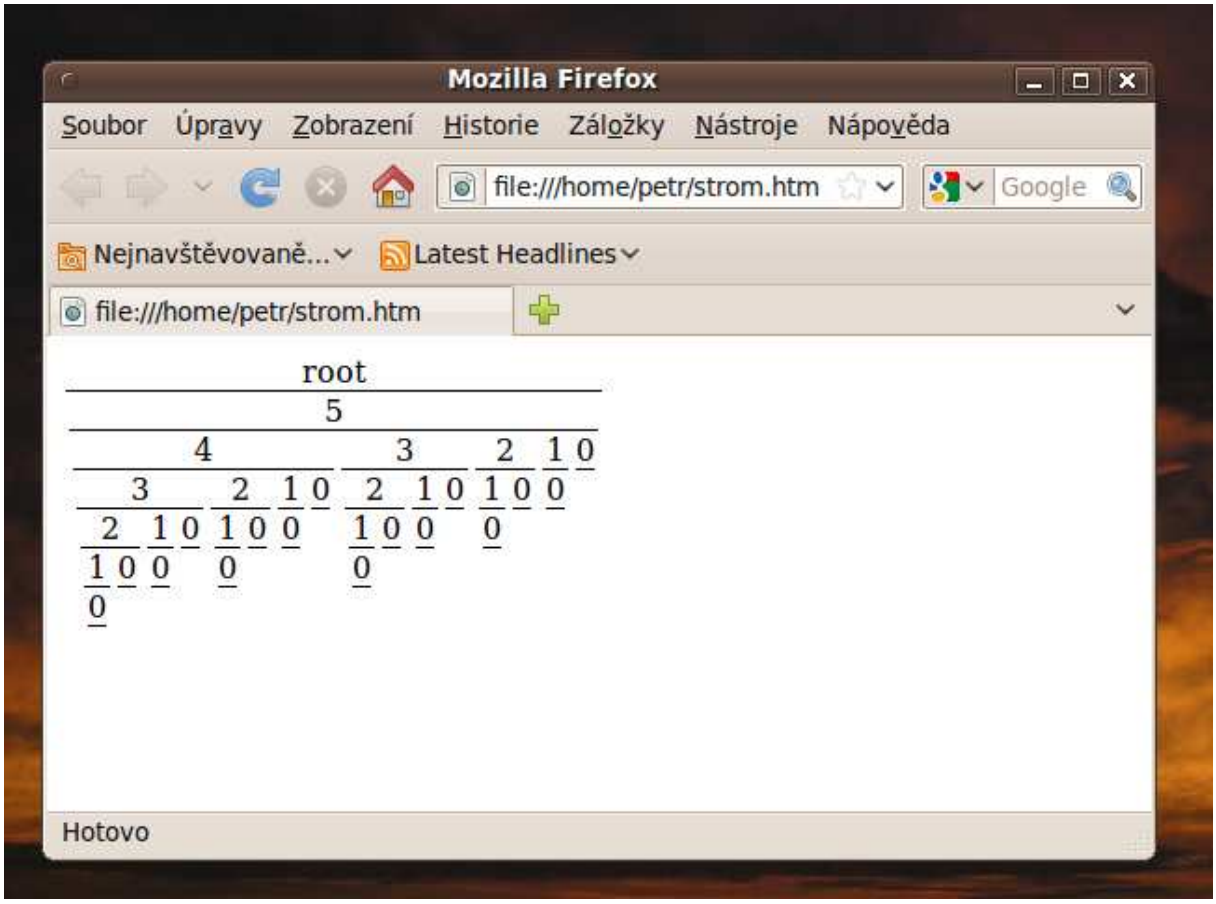
První vytvořenou funkcí je *Růst*, která bude při běhu skriptu volána na kořenový segment a při procházení všech potomků zajistí realizaci růstu. Tato funkce při svém běhu volá další pomocné funkce.

Při samotném spuštění programu je vždy vytvořen jeden kořenový prvek, na němž se v určeném počtu iterací volá funkce pro jeho růst. Následně, v předem definovaných krocích, se volají funkce pro vykreslení modelu a jeho vizualizaci.

Pro testovací účely je celá stromová struktura v této fázi reprezentována v .html souboru (obr 3), který přehledně zobrazuje vztahy a požadované vlastnosti vzniklého stromu.

Zde je patrný kořenový element *root*, který má jednoho potomka, který zde reprezentuje

jeho věk. Horizontální čáry reprezentují vztahy mezi jednotlivými generacemi větví.



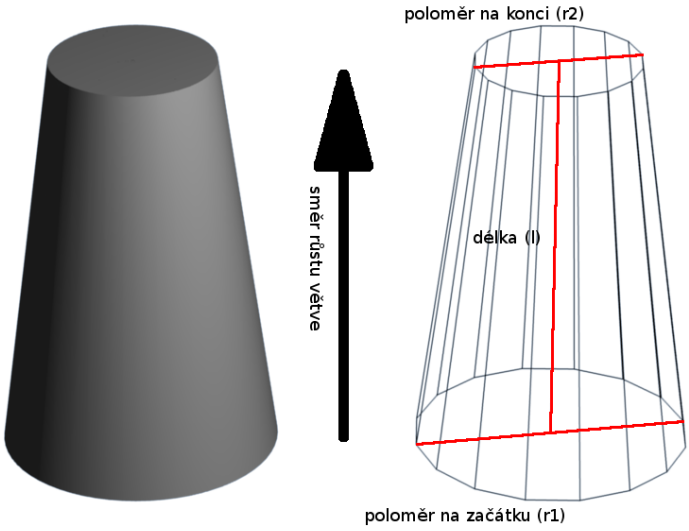
Obrázek 3: html reprezentace stromové struktury

Přes omezené možnosti tohoto jednoduchého zobrazovacího mechanismu je možné jeho prostřednictvím již v této fázi sledovat fungování skriptu a použít jej pro ověřování fungování mechanismu vykreslování v další etapě.

3.2 Informace o poloze segmentu

Pro potřeby modelování stromu bylo v první řadě potřeba vybrat jednoduchý geometrický útvar, který by mohl být skládán do modelu stromu a následně jej potřebnými rozměry.

Protože se větve musí průběžně zužovat, padla volba na jednoduchý komolý kužel (obr 4).



Obrázek 4: model segmentu a jeho rozměry

Jasné je určení délky a poloměru začátku, které jsou pro samotné vykreslení jednotlivých segmentů nezbytné. Poloměr jeho konce je snadno získáván z poloměru začátku prvního potomka, který je faktickým pokračováním větve. Tímto je zajištěna návaznost segmentů při zachování možnosti ovládat mohutnost jednotlivých segmentů.

Pokud segment nemá potomka, je považován za konec větve a poloměr konce je tedy nulový.

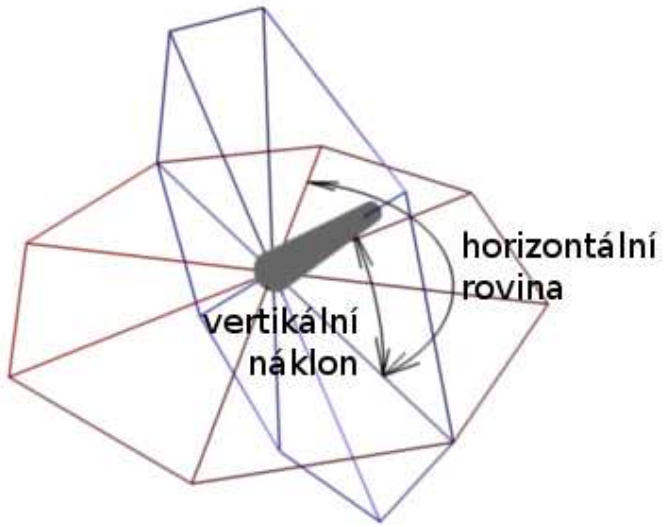
Stejněě jasné bylo i určení polohy začátku, které je ve 3D scéně určeno pomocí souřadnic

x, y a z. U jednotlivých segmentů jsou tyto souřadnice určeny z konce rodičovského segmentu, který je možné snadno spočítat ze souřadnic jeho začátku, délky a směru.

O něco složitější byla situace s určením směru segmentu. V prostředí Blender se rotace popisuje pomocí tří rotací podle všech hlavních os. Navíc je možné rotaci sledovat podle os globální či lokální souřadné soustavy.

Převedením celého problému do lokálních souřadnic se ukázala třetí rotace jako nepotřebná, protože je každý segment větve reprezentován komolým kuželem a tudíž jeho rotace podle podélné osy nepřináší žádný užitek.

Přesto, že nakonec je rotace prováděna podle globálních os, stačí znalost dvou úhlů k přesnému popisu směru symetrického rotačního segmentu v prostoru (obr. 5). Zároveň dochází k maximální úspoře nároků skriptu na paměť.

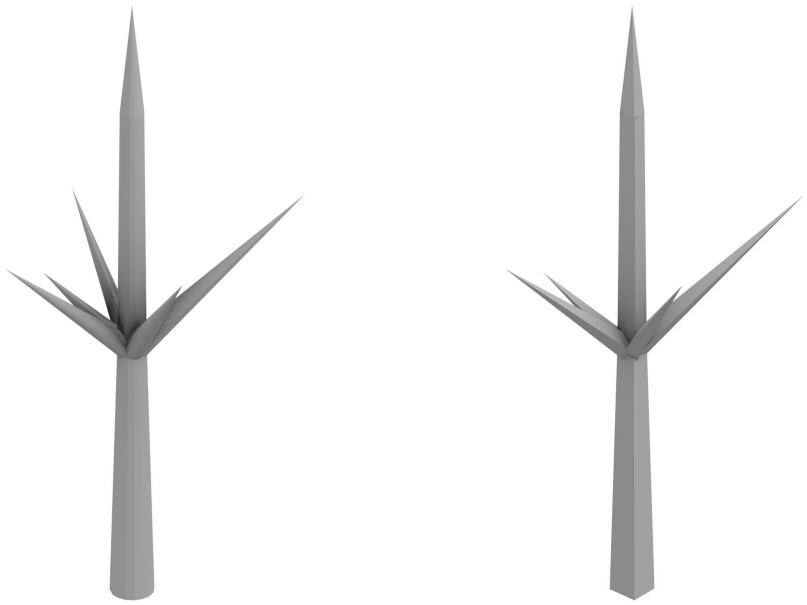


Obrázek 5: hlavní úhly popisující směr segmentu v prostoru

3.3 Vykreslování stromové struktury v prostředí Blender

První otázkou při modelování segmentů v Blenderu bylo určení podstavy komolého jehlanu. Objekt v Blenderu nemůže být vykreslen jako kružnice, ale jako n-úhelník. Z toho vyplývá rozpor mezi výpočetní náročností a kvalitou modelu.

Z hlediska výpočetního času je vhodné mít co nejmenší počet vrcholů a stěn, aby se model vykresloval co nejrychleji. Na druhou stranu by z takto modelovaných větví ve výsledné vizualizaci nešlo získat dostatečně kulaté větve. Jak je patrné z obr. 6 vlevo je jako podstava použit šestnáctiúhelník, vpravo na stejným stromu je použit čtyřúhelník.



Obrázek 6: vliv počtu vrcholů podstavy na výsledný strom

Aby nebylo potřeba pracně upravovat zdrojový kód, rozhodl jsem se definovat průřez větví pomocí proměnné, určující počet vrcholů podstavy každého segmentu. To si sice

vyžádá robustnější kód, schopný pracovat s různými hodnotami, ale ve výsledku bude i tak kód dostatečně dobře čitelný a umožní mnohem lepší výsledky, protože při detailním pohledu na malý strom se využije lépe jiné nastavení nežli při pohledu na mnohem členitější strom.

Samotné vykreslení segmentů je odděleno od funkce Růst, aby nebylo potřeba modelovat každý krok v případě, že to není potřeba.

3.4 Využití parametrů pro řízení simulace stromu

Ve chvíli, kdy je možné strom vykreslit ve 3D scéně, je možné velmi snadno sledovat jeho chování při simulaci růstu. To umožňuje lépe sledovat vliv provedených změn ve skriptu, protože textová reprezentace stromu z první etapy nenabízí snadné sledování většího množství vlastností pro větší množství segmentů v modelu.

Dosavadní náhodné nebo zcela pevně dané chování se postupně rozšiřuje tak, aby reflektovalo existenci vnějších faktorů, které růst ovlivňují.

V první řadě je potřeba ošetřit kolize jednotlivých větví, aby nedocházelo k jejich vzájemnému prorůstání. Silný vliv bude v simulaci hrát světlo, jehož nedostatek ovlivní rychlost růstu takovéto větve.

Realizován bude také fototropismus, tedy směřování větví za světlem. Ten bude ovlivňován proměnnými, které reprezentují intenzitu citlivosti na světlo.

Řada výpočtů je realizování prostřednictvím samostatných funkcí, což umožňuje jejich opakované využití na více místech skriptu. Navíc je tak snazší tyto části skriptu snáze vyhledat a zlepšovat jejich chod.

3.5 Vylepšení vzhledu stromu

V této fázi programování je na konci každého kroku ve scéně Blenderu vytvořený model stromu. Ten v této etapě ale stále vypadá pouze jako holý kmen.

V této poslední zásadní fázi budou na větvích podle jistých podmínek vytvořeny listy a všemu bude přidán potřebný materiál tak, aby se doposud bílá struktura začala blížit vzezřením skutečnému stromu.

3.6 Finální úpravy

Poslední úpravy zdrojového kódu se budou týkat hlavně některých mechanismů růstu stromu tak, aby se vytvořil lépe vyhlížející model.

Předpokládám změny některých konstant, které mají vliv na tloušťku větví či intenzitu větvení, a další drobné změny ovlivňující vzhled stromu. Tyto změny by neměly ovlivnit funkčnost jednotlivých algoritmů, ale pouze zlepšit celkový dojem z vymodelovaného stromu.

4 Realizace řešení

4.1 Vývojové prostředí Blenderu

3D grafický program Blender využívá jazyk Python a kromě spouštění skriptů v tomto jazyce je schopen je i vytvářet, k čemuž slouží interní editor.

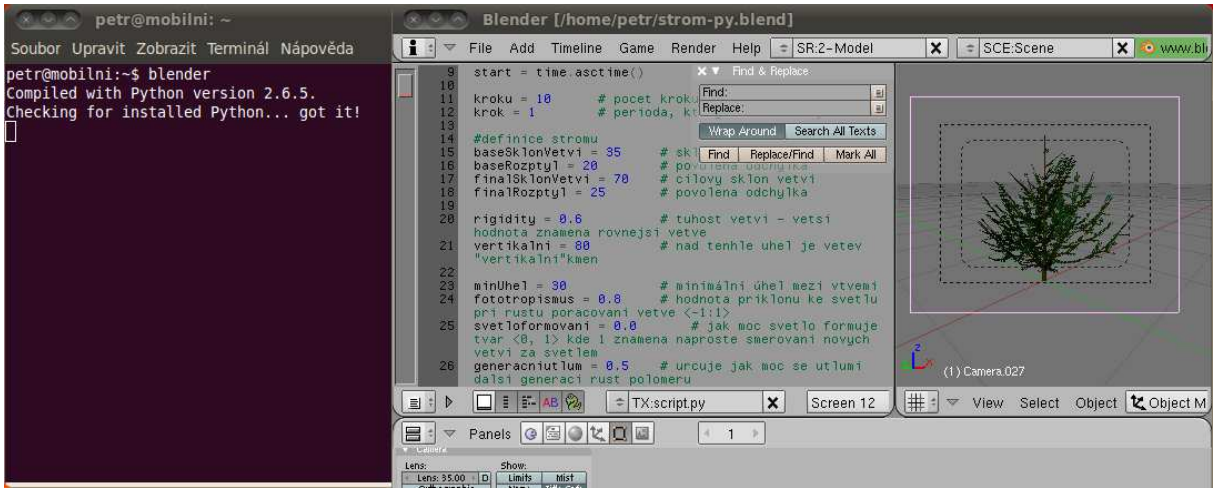
Programovací jazyk Python spadá mezi jazyky interpretované, což znamená, že není potřeba před jeho spuštěním provádět překlad. To umožňuje rychlé spouštění a díky možnosti vykreslování výsledku ve 3D okně Blenderu a výpisu pomocných informací v okně terminálu prakticky okamžitou kontrolu nad jeho chodem.

K editaci skriptu slouží vlastní textový editor implementovaný v Blenderu. Ten nabízí pouze základní nástroje, jako jsou číslování řádků a zvýrazňování syntaxe. Společně s výpisem chyb do konzole sice umožňuje i identifikaci a odstraňování chyb, ale na druhou stranu nenabízí pokročilejší možnosti vývojových prostředí, které výrazně usnadňují programování. Z tohoto důvodu jsem se rozhodl pro usnadnění programování některých částí práce využívat integrované vývojové prostředí IDLE.

Jak je patrné z obr. 7, při programování je potřeba vidět na obrazovku konzole (vlevo) a v Blenderu pak sledovat programový kód (uprostřed) a výsledný model. Nicméně toto rozložení je velmi přehledné a díky možnosti roztahování oken v Blenderu je možné mít dobrý přehled podle potřeby nad zdrojovým kódem nebo nad vykreslovaným modelem.

4.2 Vytvoření základní struktury programu

Jak bylo zmíněno dříve, programování skriptu jsem rozdělil do několika po sobě logicky navazujících kroků, které umožňují plynulou práci na zdrojovém kódu a průběžnou kontrolu jeho běhu.



Obrázek 7: Blender při tvorbě Python skriptu

4.2.1 Běhový cyklus programu

Protože je potřeba vést růst stromu v předem definovaném množství kroků, probíhají veškeré důležité události ve smyčce, jejíž počet kroků lze snadno ovlivnit. Před spuštěním této smyčky dojde k inicializaci proměnných a konstant, které řídí běh celé simulace.

Dále se vytvoří instance objektu *segment*, což je kořenový element stromu. Ten reprezentuje první nadzemní část kmene.

Následuje opakující se smyčka, která při každém opakování volá na kořenový element funkce, zajišťující růst celého stromu.

Protože vykreslování stromu nemusí být požadováno v každém kroku, rozhodl jsem se vytvořit proměnnou, která určuje periodu, s níž je stav stromu vizualizován. Pokaždé, když se provádí krok, který je celým násobkem této periody, volají se funkce pro modelování stromu, jeho vizualizaci a uložení vzniklého obrázku.

To přináší výraznou časovou úsporu hlavně při provádění delších simulací. Při provádění více kroků roste počet objektů ve scéně, které klade vysoké nároky na vytváření modelu

a jeho vizualizaci.

Z těchto důvodů jsem od začátku oddělil od růstové funkce samostatné funkce *modeluj*, která při svém zavolání modeluje strukturu kmenu, a později funkce *olisti*, která strom zakrývá listovím. Pro vytvoření samotné vizualizace a jejímu uložení do paměti počítače slouží funkce *renderuj*.

Základní schéma fungování skriptu je znázorněno v příloze A.

4.2.2 Vzhled stromové struktury

Stromová struktura je vytvářena prostřednictvím vlastnosti *potomci* každé instance objektu *segment*. Vlastnost potomci je inicializována jako prázdné pole a vkládají se do ní odkazy na všechny instance potomků daného objektu.

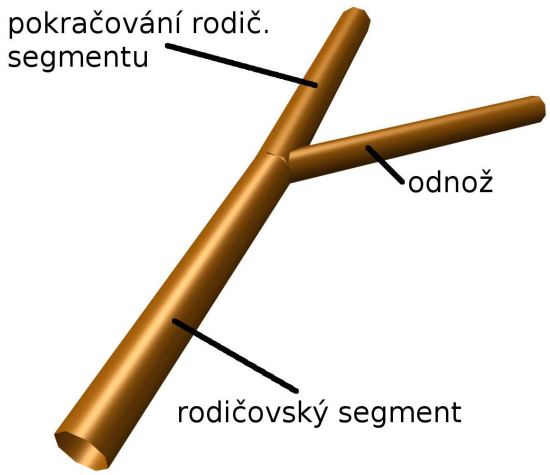
Je-li pole potomci prázdné, je segment koncem větve a tak k němu následně při jeho růstu přistupuji. Například jen konec větve může růst do délky, poloměr konce posledního segmentu je nulový, aby došlo k zakončení větve.

První potomek je považovaný za *pokračování větve*, proto tyto dva segmenty sdílejí společný poloměr. To realizuji nastavením poloměru konce rodičovského segmentu podle poloměru začátku prvního potomka. Všechny ostatní potomci segmentu jsou *odnože větve* a jejich poloměr je odlišný od rodičovského segmentu (obr. 8).

4.2.3 Struktura řídících dat

Pro chod simulace jsou potřeba různá data. Ačkoli z programátorského hlediska by se daly rozdělit jako lokální a globální proměnné, pro srozumitelnost jejich použití v simulaci jsem je rozdělil do tří základních skupin:

1. **data řídící simulaci** se týkají samotného procesu simulace a ovlivňují, kolik kroků simulace vypočítá a periodu, s níž se budou snímky vizualizovat a ukládat. Jejich



Obrázek 8: Popis větve z datového pohledu

místo je hned na začátku skriptu, aby bylo možné je snadno najít a modifikovat.

2. **globální data týkající se simulace** obsahují hodnoty, které ovlivňují strom jako celek a jsou tedy pro celý strom vždy stejné. Tato data se nachází na začátku kódu a jejich změnou se ovlivňuje chování stromu.

   Zde uložená data jsou například citlivost stromu na světlo, vliv světla na tvarování stromu, minimální délky větví před tím, než se smějí větvit. Dále sem patří poloha světla a průřez větví.

3. **lokální data týkající se simulace** se týkají každého konkrétního segmentu větve. Tyto hodnoty jsou definovány jako atributy objektu *segment*.

   K těmto vlastnostem nemá uživatel přístup, protože je spravuje samotný skript prostřednictvím funkcí *růst*, která je modifikuje podle průběhu růstu a následně zapisuje, a *modeluj*, která na základě přečtených informací vytváří model stromu.

Každý segment má například svůj věk, zdraví, své potomky, pozici v prostoru určenou jeho souřadnicemi na osách x, y a z, úhly rotace a další vlastnosti.

4.2.4 Růst stromu

Na začátku běhu programu je vytvořena instance objektu *segment*, která reprezentuje celý strom. Na tento strom je v každém kroku simulace volána její funkce *růst*. Ta provádí všechny změny spojené s růstem na tento element a následně rekurzivně pro všechny jeho potomky, čímž projde celým stromem.

Protože v tuto chvíli je cílem vytvořit skript, který vygeneruje data pro modelování stromu bez vlivu simulace, probíhá většina výpočtů na základě konstantních nebo náhodných čísel. To je vhodné hlavně pro testování dalších funkcí.

Přesto se již v této části ukázala potřeba variabilnějšího aparátu pro generování nových větví: nový segment je definován pozicí svého začátku, který se snadno spočítá z informací o jeho předchůdci jako jeho konec. Jiná situace nastává při určování jeho směru. Rozdíl je mezi pokračováním větve, které by mělo ve velké míře navazovat na směr svého předchůdce, a odnoží, která naopak musí výrazně vybočovat.

Řešením této situace bylo vytvoření dvou rozdílných funkcí, z nichž každá generuje jiné hodnoty. Přestože v této chvíli se může zdát takovéto řešení zbytečné (funkce se liší pouze v rozsahu generovaných hodnot), bylo již v tuto chvíli jasné, že výsledný kód by měl k oběma situacím přistupovat velmi odlišně a proto bude vhodné používat dvě samostatné funkce.

Také je potřeba odlišně generovat odnož od vertikálního segmentu (kmen) a od vertikálního segmentu (větev). Zatímco od kmenu se větve mohou vydat libovolným směrem a je žádoucí, aby rostly pokud možno všemi směry rovnoměrně a se stejným náklonem, z horizontální větve její odnože nejčastěji směřují pouze do stran.

Dalším faktorem, který byl již v této fázi využit, je princip růstu větví. Zatímco do šířky rostou všechny větve, do délky rostou pouze jejich konce. Proto jsem přidal omezení, že délka se může zvětšovat pouze v případě, že segment nemá žádné potomky.

Protože se ale větev může v každém kroku větvit, ačkoli za jeden krok dojde k pouze malé změně délky, stanovil jsem další pravidlo, které umožňuje větvení až po dosažení jisté limitní minimální délky.

Tuto vlastnost jsem později pro dosažení bohatší horizontální struktury stromu rozdělil na dvě, pro horizontální a vertikální segmenty. To zajišťuje dostatečnou vzdálenost mezi jednotlivými patry větví a zároveň o něco hustější větvoví v jednotlivých patrech.

4.3 Vytvoření modelu v Blenderu

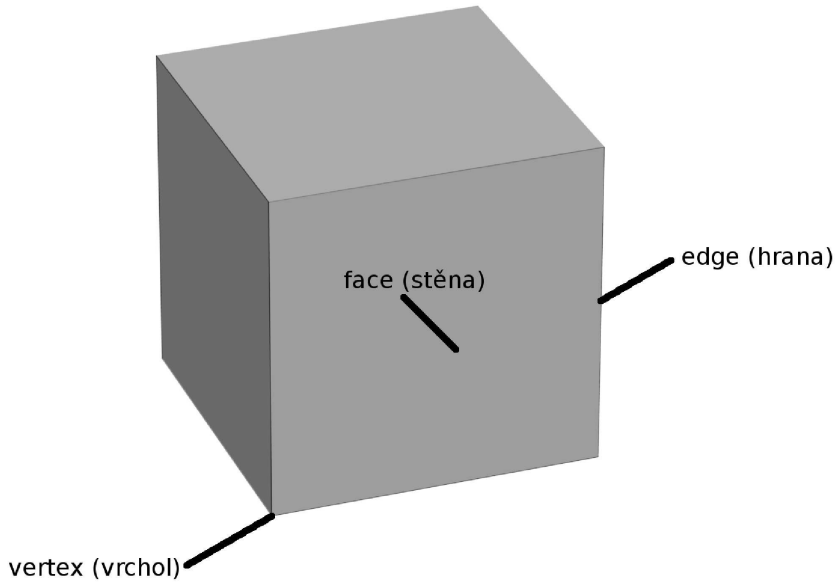
Protože není potřeba modelovat a vizualizovat každý krok simulace, vytvořil jsem samostatnou funkci *modeluj*, která je volána pouze v případě potřeby.

Funkce prochází celým stromem a na základě informací o jeho rozměrech, umístění a rotaci modeluje každý segment.

Vytvoření každého segmentu začíná vytvořením prázdného objektu, který je v několika po sobě jdoucích krocích modelován. Jak je patrné z obrázku 9, každý objekt je reprezentován vrcholy (vertex), hranami (edge) a stěnami (face). Prvním krokem při modelování je vytvoření vrcholů, které se následně přidávají plochám, čímž vzniká objekt. Hrany vznikají automaticky jako hrany plochy.

Protože je potřeba podstavy modelovat jako n-úhelníky, vycházím z popisu kružnice pomocí funkcí *sin* a *cos*:

$$x = polomer \times \sin \frac{360}{kmenN \times i} \tag{1}$$



Obrázek 9: objekt ve 3D

$$y = polomer \times \cos \frac{360}{kmenN \times i} \tag{2}$$

kde parametr i odpovídá pořadí daného bodu v n-úhelníku a $kmenN$ reprezentuje počet vrcholů podstavy.

Hodnota souřadnice z je nulová pro spodní podstavu a rovna délce větve pro horní podstavu.

Každý takto připravený vertex se vloží do pole vrcholů, odkud jsou následně přiřazovány jednotlivým stěnám.

Protože počet vrcholů v podstavě je proměnlivý, je potřeba, aby se i vrcholy přiřazovaly stěnám podle jejich počtu. To zajišťuje jednoduchá smyčka, která pro každý vrchol ve spodní podstavě najde správné indexy sousedních vrcholů tak, aby společně vytvořily lichoběžníkovou stěnu útvaru. Spodní ani horní podstavy se neuzavírají, protože na ně navazují další větve.

Obrázek 10: Vybraný segment a jeho vlastnosti určující pozici a rotaci v prostoru

V případě, že by se příliš prudkým zahnutím větve tvořily ve stromu trhliny, bylo by možné mechanismus tvorby segmentů upravit tak, aby se jejich negativní dopad na vzhled stromu co nejvíce vyrušil. Na druhou stranu je potřeba brát v potaz výrazné prodloužení doby modelování a vizualizace. S ohledem na velmi malé a mezery mezi segmenty, které nejsou většinou ani vidět, nejeví se takovéto opatření jako opodstatněné.

Poslední fází modelování je nastavení rotace a pozice segmentu. Tuto část umožňuje Blender realizovat velmi jednoduše nastavením vlastností daného objektu. Konkrétně jde o vlastnosti *LocX*, *LocY* a *LocZ* pro polohu počátku podle jednotlivých os, a rotací *rotZ* pro rotaci podél svislé osy (směr růstu) a *rotX* pro úhel náklonu segmentu větve (obr. 10). Další vlastnosti vybraného objektu zobrazené v obrázku se týkají jeho rozměrů a nemají pro skript žádné využití.

V této fázi je skript schopen generovat strom a následně jej modelovat, nicméně jeho růst stále podléhá pouze vlivům náhody. Přesto je tento stav důležitý, protože vykreslený model ve 3D umožňuje snáze sledovat vzhled stromu. Proto je snazší kontrolovat vliv a funkčnost implementovaných funkcí realizující reakci na vnější faktory. Zároveň model umožňuje vyhledávat další problémy, které je potřeba vyřešit.

4.4 Implementace mechanismů simulace

4.4.1 Omezení větvení pro segmenty

Protože se všechny segmenty mohou ve kterémkoli kroku větvit, bylo nutné toto větvení nějak omezit.

V první fázi jsem proto rozdělil segmenty na horizontální a vertikální pomocí proměnné, která určuje úhel dělící obě skupiny.

Větve vertikální považuji za kmen a jako takové se mohou větvit všemi směry takřka neomezeně, dokud pro ně existuje místo. Horizontální větve se do stran mohou větvit pouze v omezené míře.

Je možné, aby se větev horizontální začala chovat jako vertikální, pokud úhel náklonu překročí hraniční hodnotu, a naopak.

4.4.2 Vitalita segmentů

Různé větve mohou mít různé podmínky k růstu, což odráží vlastnost segmentu *vitalita*. Její počáteční hodnota je 1. Vitalita má na rychlost růstu do délky i šířky.

V průběhu simulace se její hodnota snižuje vlivem různých přidávaných faktorů až k nulové hodnotě, která znamená ukončení růstu. Takovýto segment se již nevětví a neroste, stejně jako všichni jeho potomci. Nastavením záporné hodnoty se odstraní celá větev od

zvoleného segmentu.

4.4.3 Rigidita větví

Rigidita neboli tuhost větví patří mezi globální konstanty, tedy jediná hodnota má vliv na celý strom. Její zavedení je nezbytné kvůli určení úhlů, pod kterými má růst nová větev.

Nový úhel se počítá na základě různých vlivů, které jsou rozpracovány později v rámci zlepšování simulační části programu. Tento úhel vzniká jako kombinace náhody (v menší míře) a z části pod vlivem vnějších faktorů (slunce, prostor). To často vede ke vzniku velmi prudkých zalomení větví, které se oddělily od rodičovské větve pod úhlem, který je nevede do nejvhodnějšího prostoru.

Potřeba omezit úhel, o který se větev může ohnout, jsem nechtěl určovat krajní hodnotou úhlu, protože takovéto omezení mi přijde vhodné spíše pro matematicky definované problémy nežli pro ovlivnění růstu rostliny.

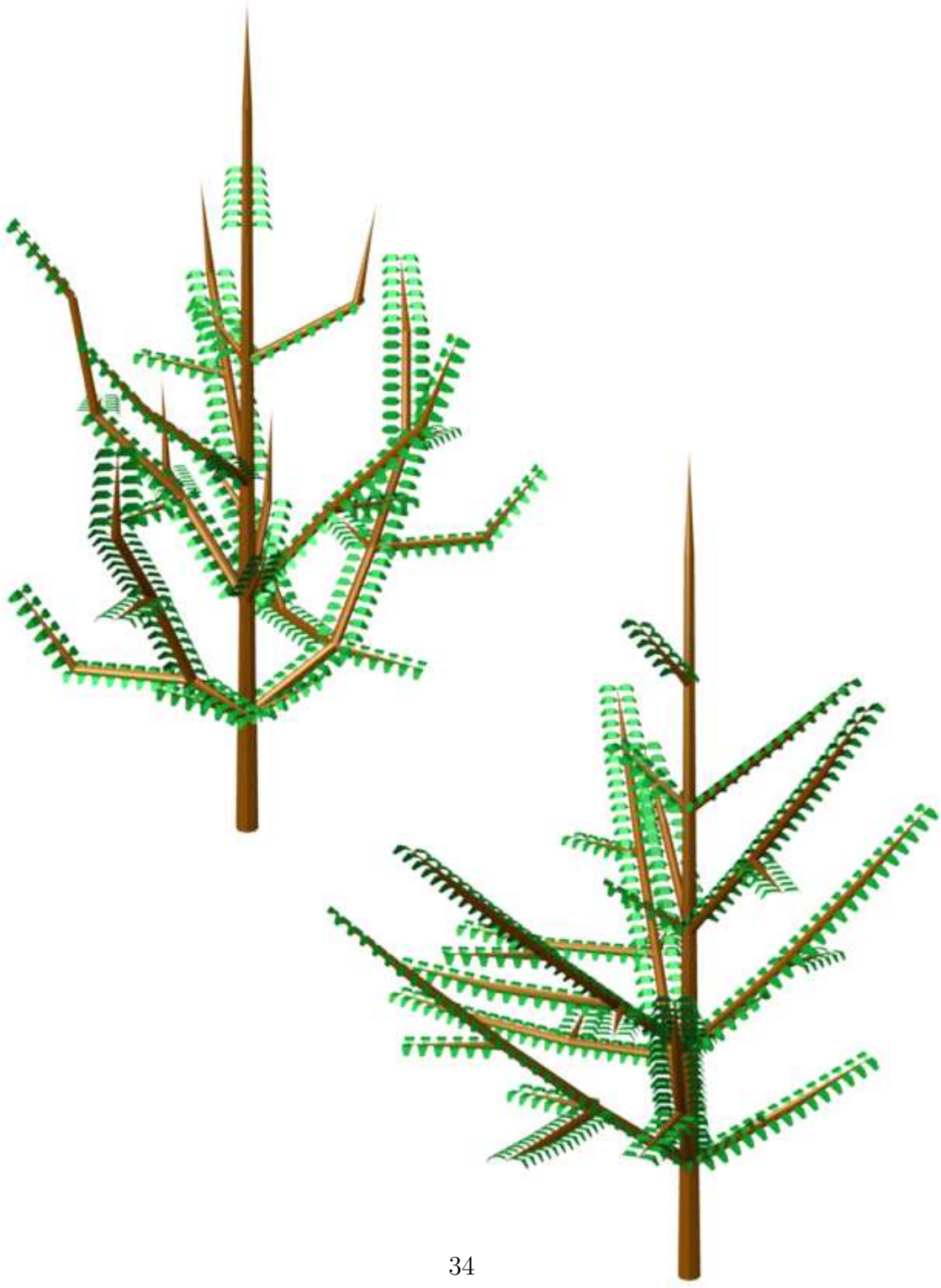
Běžící simulátor proto počítá novou hodnotu úhlů *Nalfa* a *Nbeta* na základě hodnoty *rigidity* z úhlů rodičovského segmentu (*alfa* a *beta*) a požadovaných hodnot ((*Falfa* a *Fbeta*)) podle následujících vzorců:

$$Nalfa = rigidity \times alfa + (1 - rigidity) \times Falfa \tag{3}$$

$$Nbeta = rigidity \times beta + (1 - rigidity) \times Fbeta \tag{4}$$

Kde vlastnost *rigidity* reprezentuje tuhost větve na intervalu $\langle 0, 1 \rangle$.

Vliv rigidity je dobře patrný na modelu (obr. 11): vlevo nahoře je tuhost větví nastavena na velmi nízkou hodnotu, proto se větve snadno ohýbají, což je zřejmé z velké úhlu v takřka každém větvení.



Obrázek 11: Vliv rigidity (0,1 vlevo nahoře a 0,9 vpravo dole)

Obrázek 12: Původní podoba stromu vytvořeného skriptem

Obrázek vpravo dole naopak reprezentuje strom, ovlivněný vysokou hodnotou rigidity s hodnotou blížící se maximální hodnotě. Zde je naopak patrné, že větve se takřka neohýbají a rostou pod takovým úhlem, pod jakým z rodičovské větve vyrostly.

4.4.4 Reakce na segmenty v okolí

Prvním problémem, na který jsme při vymodelování stromu narazili je skutečnost, že větve rostou zcela náhodně bez jakékoli reakce na okolní segmenty (obr. 12).

Vlivem toho dochází k situaci, kdy některé segmenty rostou s velmi malým úhlem mezi

sebou, což vede k výraznému srůstání. Dalším nežádoucím jevem je křížení a prorůstání větví.

Za tímto účelem jsem vytvořil funkci *Zjisti*, kterou volá vždy rostoucí segment (tj. segment, na který je volána funkce *Růst*). Tato funkce prochází celým stromem a porovnává tak každý segment se všemi ostatními segmenty.

Tento mechanismus sice zvyšuje výpočetní náročnost, protože vzniká dvojitá smyčka s velkým množstvím opakování. Proto je potřeba co nejlépe ošetřit samotné propočty v jejím vnitřku tak, aby se provádělo minimum obtížných výpočtů.

Zjednodušený běh programu pro jeden krok simulace vypadá zhruba takto:

```
definice funkce Růst:
   Pro každý segment (hlavni):
      Pro každý segment (soused):
         ZjistiKolizi(hlavni, soused)
      volej funkci Růst(...)
```

Funkce *Zjisti* se volá před provedením růstu, aby bylo možné informace o kolizích okamžitě využít. Voláním funkce *Růst* na konci této smyčky dochází ke spuštění novému cyklu. Navíc při zjišťování kolizí může dojít k odstranění části větve, pro níž již není potřeba vypočítávat její růst.

4.4.5 Výpočty kolizí mezi segmenty

Funkce *Zjisti* probíhá podle schématu z obrázku v příloze B.

V první řadě je potřeba zajistit, aby oba porovnávané segmenty nebyly totožné. v takovém případě se celý cyklus přeskočí a následuje porovnávání dalších dvojic. Pokud dvojice segmentů začíná ve stejném bodě (tj. mají stejné souřadnice *x0*, *y0* a *z0*, je nutné zajistit, aby obě větve směřovaly dostatečně daleko od sebe, tedy svíraly úhel větší než je nastavený

limit.

Spočítá se úhel mezi nimi a pokud je výsledná hodnota menší než stanovený minimální úhel mezi větvemi, mladší z větví je ukončena a odstraněna z modelu.

Pokud testované segmenty nezačínají ve stejném bodě, je potřeba zjistit, zda nekolidují. Aby se předešlo zbytečným složitým výpočtům, počítá se v první řadě vzdálenost mezi počátky větví, která je porovnána se součtem jejich délek.

Vzdálenost dvojice bodů lze spočítat ze vzorce

$$|V_1V_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{5}$$

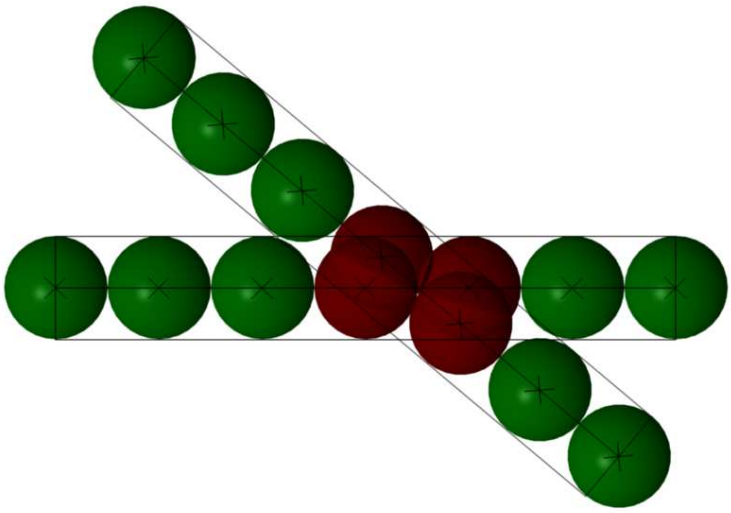
Součet délek segmentů je prostý součet dvou hodnot.

Pokud se tyto dva segmenty nemohou pro svojí vzájemnou velkou vzdálenost protnout, není potřeba u nich provádět důkladnější testování.

Testování na průnik segmentů se provádí metodou *průniku koulí*. Do dvojice větví se vepíše několik koulí a jejich průniky lze opět spočítat vzájemnou vzdáleností obou bodů (obr. 13). Ve skutečnosti je ale každému segmentu vepsáno takovýchto koulí mnohem více a i jejich poloměr neodpovídá přesně poloměru segmentu v jejich bodě, ale o něco větší hodnotě. Tato opatření slouží k zamezení vzniku míst, kde segment existuje, ale nepočítá se s ním (bílá místa na obrázku 13). Naopak nepatrně větší citlivost v blízkosti segmentu není vážný problém, protože pomáhá simulovat ubývající volné místo v okolí větve.

Programově je toto řešeno dvojicí smyček s patnácti opakováními, které modelují patnáct koulí v každém segmentu. Celkem je tak pro každou dvojici provedeno 225 výpočtů vzdáleností.

V případě, že se takto dva segmenty setkávají na příliš malém prostoru, dochází k redukci *vitality* mladšího segmentu. Pokud dochází k ještě těsnějšímu kontaktu, dochází k úplnému ukončení růstu mladší větve.



Obrázek 13: Řešení kolize dvou větví metodou průniku koulí

V rámci zrychlení běhu simulace jsem se rozhodl testovat kolize pouze mezi koncovými segmenty a zbytkem celého stromu (namísto původního testování všech segmentů se všemi), čímž se podstatně snižuje počet výpočtů. Zde vycházím z představy, že kolidovat se zbytkem stromu mohou spíše segmenty rostoucí do délky, nežli ty, které pouze mohutní.

Touto drobnou změnou bylo dosaženo významné úspory výpočetního času pro kratší simulace: v prvních 10 krocích dochází k redukci výpočetního času o 99 %, v prvních 20 krocích je úspora stále na úrovni 74 %.

4.4.6 Vliv na tvar stromu - vysoký či široký

Protože se větve i nadále od vertikálního kmene větví zcela náhodně a pouze při kolizi s jinými staršími segmenty jsou odstraňovány, rozhodl jsem se přidat do modelu další dvě dvojice parametrů, které ovlivňují náklon větví.

První dvojice ovlivňuje úhel, pod jakým větve vyrůstají z vertikálního kmenu: první

Obrázek 14: Tvar koruny ovlivněný směřováním větví do šířky (nahoře) a do výšky (dole)

určuje střední hodnotu a druhý určuje rozptyl trojúhelníkového rozdělení.

Druhá dvojice určuje stejným způsobem úhel, pod jakým se větve snaží růst. Tento parametr je nadále konfrontován s vlivem slunce, které směr růstu větví ovlivňuje.

Nicméně kombinací těchto vlastností lze do velké míry ovlivnit tvar koruny a její siluetu, která je pro celkový vzhled modelované rostliny velmi důležitá (obr. 14).

4.4.7 Množství dopadajícího světla

Stromy a rostliny obecně potřebují ke svému růstu světlo. Obecně platí, že čím méně světla má strom k dispozici, tím pomaleji roste.

Proto jsme přidali mezi důležité faktory ovlivňující růst stromu množství dopadajícího světla. Tento faktor reflektuje proměnná *světlo* s inicializační hodnotou 1.

S každou detekcí stínícího objektu se tato hodnota zmenšuje vynásobením s empiricky určenou konstantou; menší hodnota následně zpomaluje růst segmentu.

Zjišťování probíhá v rámci funkce *Zjisti* pouze pro koncové segmenty. Ty jsou porovnávány se všemi segmenty stromu a počítá se jejich vzdálenost mezi nimi a světelným paprskem, který dopadá na zjišťovaný segment, viz obr. 15.

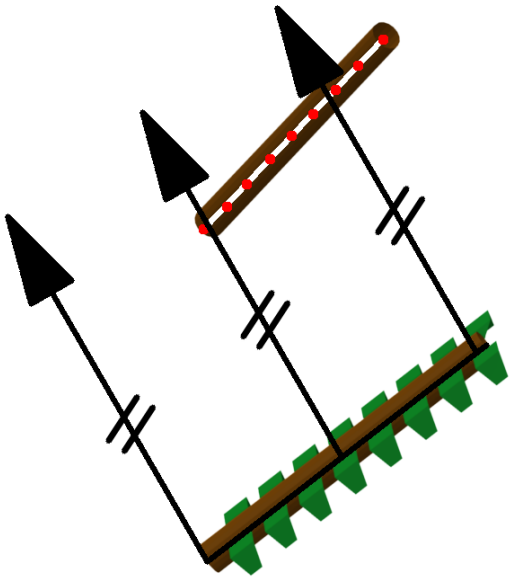
Podobně jako při zjišťování vzdáleností mezi segmenty větví i nyní se procházejí dvojice segmentů, z nichž jedna je koncovým segmentem větve (*Master*) a u druhé (*Slave*) zjišťujeme, zda tomuto segmentu stíní.

Na obou se najde pevně stanovený počet bodů, z nichž se v rámci jedné smyčky vynese vektor rovnoběžný se slunečním paprskem (černé šipky).

Vzorce níže operují se souřadnicemi x, y a z všech tří entit: bodů z obou segmentů (*Master* a *Slave*) a světla (*Light*).

$$u = \frac{(x_M - x_S) \times x_L + (y_M - y_S) \times y_L + (z_m - z_s) \times z_L}{|x_L^2 + y_L^2 + z_L^2|} \tag{6}$$

Hodnota u značí, kolikrát je potřeba vynásobit původní vektor světla, abychom se dostali do bodu X (obr 16), který leží na přímce a je nejblíže bodu, jehož vzdálenost od přímky počítáme. Pokud je jeho hodnota záporná, znamená to, že testovaný segment leží blíže zdroji světla nežli sousední segment (slave). V takovém případě nemůže stínit a není potřeba dále počítat.



Obrázek 15: Model výpočtu dopadání světla

Pokud může vybraný segment stínit testovaný segment, vypočtou se souřadnice tohoto bodu:

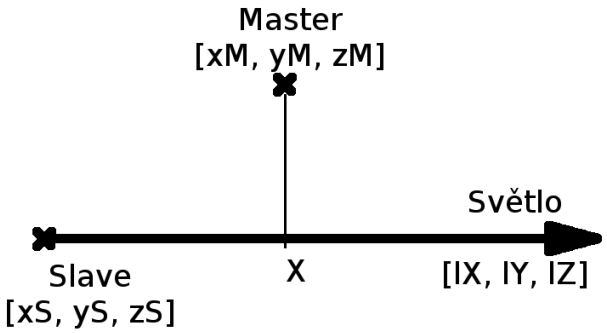
$$x_X = x_S + u \times x_L \tag{7}$$

$$y_X = y_S + u \times y_L \tag{8}$$

$$z_X = z_S + u \times z_L \tag{9}$$

Takto získané souřadnice umožňují převést problém vzdálenosti na otázku dvou bodů, k čemuž již existuje jednoduchá funkce využívající vzorec (5).

V takovém případě mladší segment stínil starší, což je stav, který neovlivňuje růst stromu. Starší segmenty dostávají živiny (a tedy i světlo) prostřednictvím mladších segmentů. v opačném případě, pokud by simulace brala v potaz situace, kdy mladší segment



Obrázek 16: Výpočet vzdálenosti přímky a bodu

stínil starší, by nedocházelo pouze ke zpomalení růstu větví na stinné straně stromu, ale zpomalil by se i růst celého kmene, což se ve skutečnosti neděje.

S každou detekcí stínění je zmenšena hodnota přijímaného slunečního záření o poměrnou část, čímž se odlišuje malé stínění jedním segmentem v několika mála bodech od velkého stínění více větvemi.

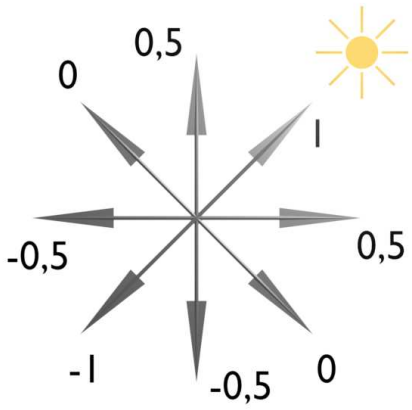
4.4.8 Fototropismus

Slovem *fototropismus* se označuje směřování rostliny ke zdroji světla.

Pro potřeby simulátoru je poloha slunce zadána pro uživatele snadno pochopitelnými souřadnicemi x,y a z. Z nich jsou následně pro potřeby simulace spočítány úhly, ze kterých světlo přichází. To je potřeba pro porovnávání s větvemi, jejichž směr růstu je zadán právě dvojicí úhlů.

Reakce stromu na světlo je volena prostřednictvím parametru *fototropismus* a může nabývat hodnoty z intervalu $\langle -1; +1 \rangle$.

Krajní hodnota $+1$ znamená, že větve se snaží (v rámci svých omezení) co nejrychleji zamířit ke zdroji světla. Naopak hodnota -1 působí, že se rostoucí větve snaží růst co možná nejrychleji od zdroje světla. Hodnota mezi těmito krajními mezemi směřuje větve



Obrázek 17: Směřování větví pro různé hodnoty fototropismu

do obou stran s odpovídající mírou příklonu či odklonu ke slunci (obr. 17).

V běhu skriptu bývá tento jev značně obtížně sledovatelný, protože se stáčející se větve mísí s novými výhonky, které vyrůstají do obou stran z rodičovského segmentu.

Přesto je možné z pohledu shora sledovat projevy této aktivity (obr. 18).

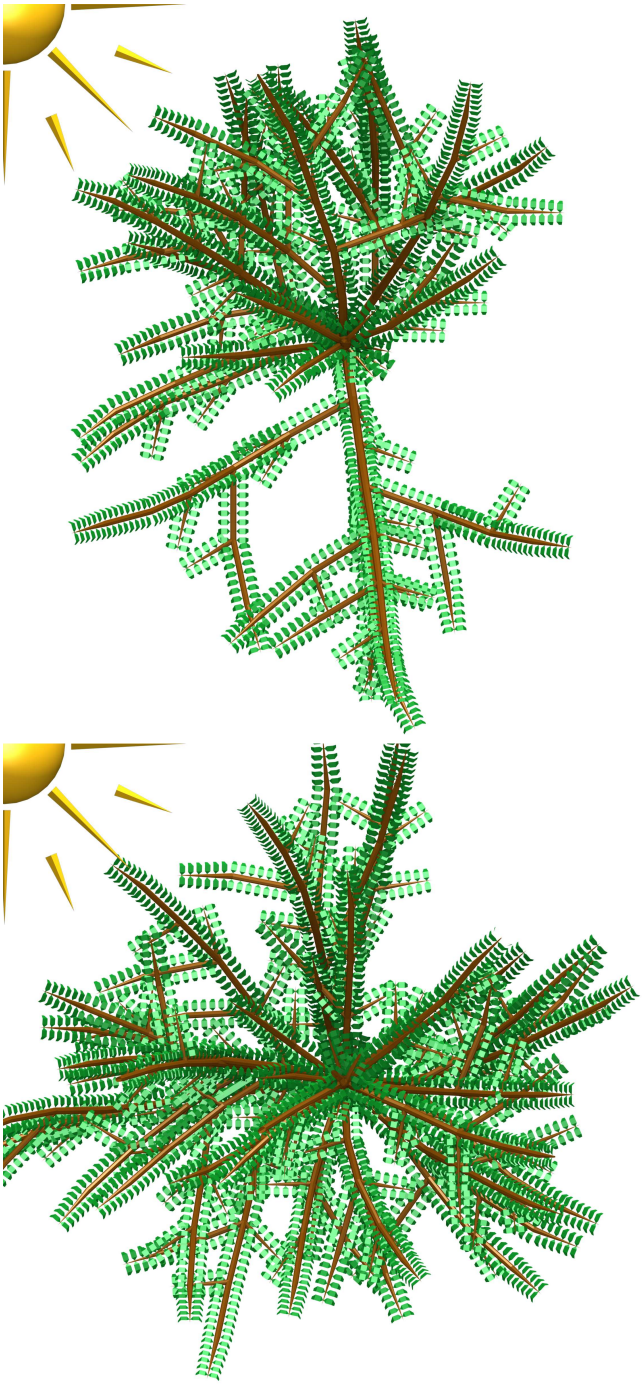
4.4.9 Vliv slunce na směřování nových odnoží

V návaznosti na předchozí vlivy světla jsme se rozhodli vytvořit další mechanismus, který by ovlivňoval úhel, pod jakým bude větvení probíhat.

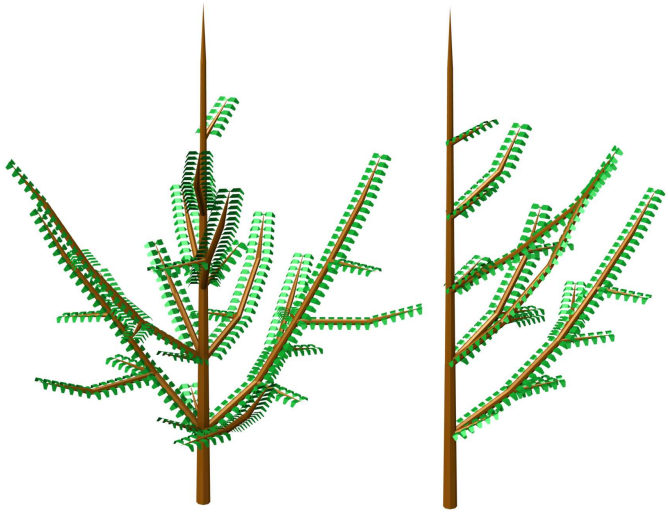
Prvotní představa byla jasně založena na tom, že nový parametr bude určovat, do jaké míry se nově vzniklé větve budou přiklánět či odklánět od slunce.

Na rozdíl od *fototropismu* se tato vlastnost netýká pokračování větví, ale nových odnoží. Parametr, který ovlivňuje formování stromu světlem může nabývat libovolné hodnoty mezi nulou a jedničkou.

Menší hodnoty znamenají menší vliv světla, větší hodnoty naopak vliv zesilují. Mechanismus tohoto vlivu světla jednoduše omezuje úhel nově vzniklého segmentu. Ten se může



Obrázek 18: Vliv konstanty fototropismu na simulaci (hodnoty 0,9 nahoře a 0,1 dole)



Obrázek 19: Vliv hodnoty *světloformování* na tvar koruny (0,0 vlevo a 0,7 vpravo)

od úhlu, pod nímž dopadají sluneční paprsky, lišit o maximálně 180-násobek rozdílu 1 a této konstanty.

To, teoreticky, zajišťuje správnou funkci této konstanty. Při nulové hodnotě vlivu světla je umožněn rozptyl větví 180 stupňů na obě strany, což pokrývá celou rovinu. Maximální hodnota konstanty naopak zajišťuje, že se nově vytvořená větev nebude nijak odchylovat od směru slunečního světla.

V praxi ale tento mechanismus funguje o poznání hůře pro malé hodnoty, pro které nedokáže při generování malého počtu větví pokrýt celou rovinu a strom tedy nepůsobí dostatečně věrohodně. Proto se pro nižší hodnoty vlivu světla využívá náhodná hodnota mezi 0 a 360.

4.5 Finální úpravy

Přesto, že se podařilo vytvořit funkční simulaci stromu, která reaguje na různé vnitřní i vnější faktory, předávané skriptu pomocí konstant a proměnných, jsou jeho výsledky značně neuspokojivé z hlediska vzhledu.

4.5.1 Vznik tenčích větví

Abych mohl omezovat tloušťku větví oproti jejich rodičovskému segmentu, vytvořil jsem lokální vlastnost *generace*. Kořenový segment je označen jako nultá generace.

Každý první potomek je stejné generace jako jeho rodič, generace každého dalšího potomka je o jednu inkrementována oproti svému rodiči (obr. 20). To ve spojení s konstantou *generační útlum* umožňuje flexibilně omezovat růst tloušťky větví.

Vzorec pro tloustnutí větví je rozšířen do podoby:

$$polomer = polomer \times \left(1 + prirustek \times generacniUtlum^{generace-1}\right) \tag{10}$$

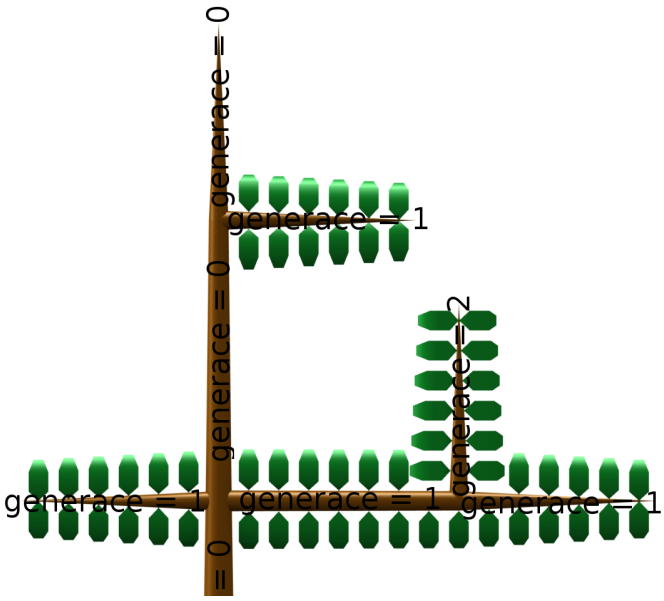
Kde hodnota *přírůstek* je hodnota vypočítaná na základě *vitality*, *světla* a konstant určujících rychlost růstu.

Výsledkem je pak věrohodnější vzhled stromu, kterému zůstává mohutný kmen, ale jeho větve jsou podstatně subtilnější.

4.5.2 Přidání listů do modelu stromu

Jednou z posledních etap vytváření modelu je přidání listů do hotového modelu.

Původní představa olistění stromu počítala s vytvořením modelu listů a jejich vložením na segmenty splňující všechny požadavky na olistění. Mezi těmito podmínkami byl požadavek na malý poloměr segmentu, protože listy vznikají většinou na mladších větvích, zatímco na



Obrázek 20: Znázornění hodnoty *generace* ve větvi

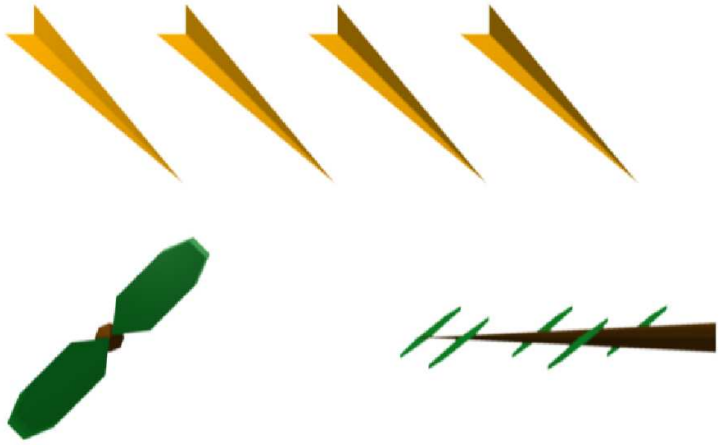
starších, mohutnějších, větvích listy vznikají zřídka. Tento poloměr je možné uživatelsky volit podle potřeby.

Následně byla využita znalost přísunu světla k jednotlivým segmentům tak, aby listy vznikaly pouze na těch segmentech, ke kterým přistupuje dostatek světla.

Naopak u těch segmentů, kde došlo k porušení byť jediné z podmínek, listy okamžitě zanikají.

Nicméně takto vzniklé listy působí velmi umělým dojmem při větším odstupu, protože jsou vzájemně identické a rostou velmi pravidelně. Proto bylo do rozměrů listů přidána malá míra náhodnosti.

S ohledem na vliv světla na růst stromů se ukázalo vhodné umožnit listům směřovat za sluncem. Zde je proto vypočten pro každý segment úhel, jaký svírá se světelnými paprsky. Poté dochází k naklánění listoví v závislosti na velikosti tohoto úhlu.



Obrázek 21: náklon listů proti slunci

Dvě krajní situace jsou zobrazeny na obrázku 21, který zobrazuje segment větve v bočním pohledu kolmém na přicházející světelné paprsky. Větev vlevo roste kolmo na směr této roviny a proto se nakloněním listů zvýší přísun světla na listy. Větev z obrázku vpravo roste rovnoběžně s rovinou snímku a proto by naklonění listů podle podélné osy segmentu nepřineslo žádný účinek.

4.5.3 Vizualizace

Vizualizace hotového stromu je časově náročná úloha a jak se ukázalo při testování, na rozdíl od modelu nemusí být vždy potřeba. Například pokud chce uživatel hotový model importovat do jiné scény, je vizualizace zbytečnou ztrátou času, protože stačí sledovat stav modelu po několika krocích. Z tohoto důvodu se funkce *renderuj* volá při vytvoření modelu pouze v případě, že je požadována.

Pro urychlení vizualizace se využívá v Blenderu implementovaného renderování ve více vláknech, které při rozdělení scény na několik menších oblastí umožňuje paralelní

vykreslování více částí najednou podle počtu procesorů. Zde byla skriptu přidána možnost ovládat množství horizontálních a vertikálních částí tak, jak je možné nastavit přímo v Blenderu.

Hotové snímky jsou následně ukládány na pevný disk ve formátu PNG s alfa kanálem, aby byly použitelné k dalšímu zpracování.

5 Vyhodnocení řešení

Při vypracovávání této diplomové práce byl vytvořen skript umožňující simulaci růstu rostliny a její modelování v 3D prostředí grafického programu Blender.

Při simulaci růstu vznikající strom je schopen reagovat jak na vnější podněty, jakými jsou světlo a volný prostor v jeho okolí, tak i na vnější stavy uvnitř stromu, jako jsou zdraví jeho větví a genetické informace.

Pro samotné modelování stromu bylo zcela nezbytné použití nástrojů objektově orientovaného programování, které umožňuje snadnou práci s daty uvnitř stromu.

Ačkoli se nepodařilo vytvořit animovaný model, který by v sobě obsahoval informace o růstu tak, aby bylo možné po jeho vytvoření procházet jeho vývoj, podařilo se vytvořit použitelný nástroj pro animování růstu stromu.

Přesto, že skript funguje a umožňuje celkem rychlé a efektivní modelování stromů, existuje řada možností, jak jej dále vylepšit.

V první řadě by bylo vhodné využití stále rostoucích možností vícejádrových procesorů k paralelizaci celého procesu generování. Rozdělení programu do několika vláken, z nichž by každé běželo na vlastním jádru, která by vykonávala na sobě nezávislé výpočty, by umožnilo výrazné zrychlení běhu skriptu.

Například by bylo možné oddělit fázi výpočtů růstu, vykreslování a následné vizualizace (ta je již sama v Blenderu paralelizována). Protože se ukázalo, že právě poslední dvě fáze zabírají nejvíce výpočetního času, došlo by k výraznému zrychlení, kdyby mohlo více vláken modelovat vytvořenou stromovou strukturu.

Další pro uživatele užitečnou změnou by bylo lepší uživatelské rozhraní, které by umožňovalo editaci vstupních dat nikoli prostřednictvím změn v kódu, ale v rámci nějakého grafického rozhraní.

5.1 Nároky na běh programu

5.1.1 Počet objektů ve scéně

Při každém kroku simulace dochází ke vzniku nových segmentů větví a případně i zániku některých starých segmentů. Přesto je logické a z běhu programu jasně patrné, že počet segmentů ve scéně roste.

Počet všech objektů ve scéně je možné zjistit přímo v okně Blenderu. Tato hodnota se ale v průběhu vykreslování velmi rychle mění tak, jak jejich počet přibývá při vykreslování a následně se rychle nuluje při vymazání scény.

Snazší přístup k počtu objektů ve scéně je prostřednictvím proměnné, do které se uloží pole všech objektů ve scéně. Na konci každého vykreslování je možné vypsat do konzole velikost tohoto pole, čímž je možné snadno sledovat počet objektů ve scéně.

Zde je ale důležité nejen odečíst vždy objekty nesouvisející se stromem (světla, kamera), ale mít na paměti i skutečnost, že skript segmenty nemaže, ale pouze je vyřazuje ze scény. Proto je potřeba od takto získané hodnoty odečíst předchozí hodnotu.

Pro provádění těchto výpočtů jsem deaktivoval funkci pro olistění stromu, protože jejich počet je mnohem variabilnější s ohledem na rozměry segmentů.

Na obrázku 22 je znázorněn vývoj počtu segmentů, tvořících model stromu. Spodní čára určuje nejmenší naměřený počet objektů, horní pak nejvyšší počet objektů ve scéně. Prostřední čára reprezentuje průměrnou hodnotu z deseti testů.

Protože každý segment se vykresluje jako velké množství bodů a ploch, klade rostoucí počet segmentů zvýšené časové nároky na fázi modelování a vizualizace.

Například při použití relativně úsporných šestibokých jehlanů je při dosažení 12 000 segmentů (což je průměr pro 20 kroků simulace) potřeba vykreslit 144 tisíc vertexů a 72 tisíc plošek jen pro kmen bez listí.

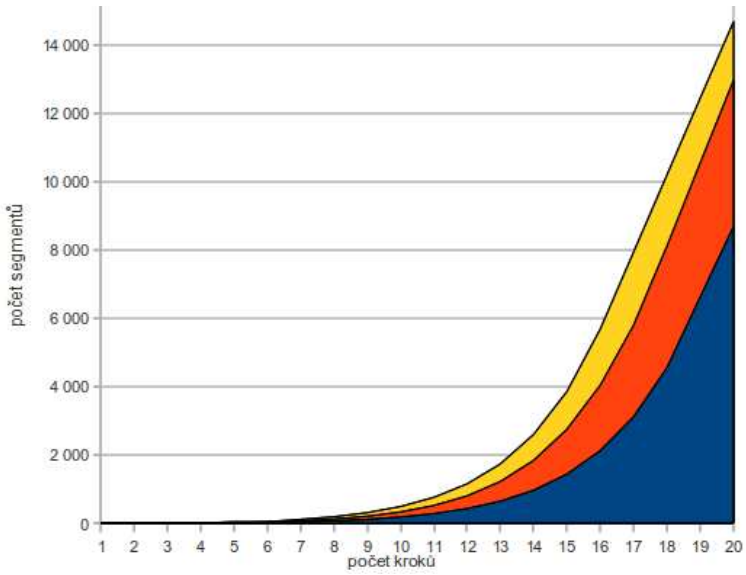
5.1.2 Časová náročnost simulace

Zřejmě nejdůležitější charakteristikou skriptu je jeho časová náročnost.

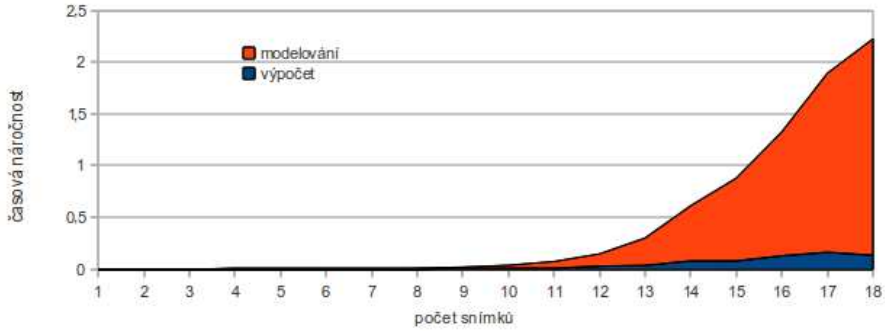
Protože je celý běh programu rozdělen do tří základních etap, sleduje graf časové náročnosti (obr. 23) dobu provádění jednotlivých etap: výpočet růstu stromu, modelování celé rostliny včetně listí a následnou vizualizaci.

Vzhledem k tomu, že strom může v různých simulacích růst různým způsobem, opět jsem provedl několik opakování testů a do grafu jsem znázornil průměrné hodnoty časů jednotlivých etap.

Test byl prováděn na počítači Dell Inspiron 1720 s dvoujádrovým procesorem 2,1 GHz, 2 GB RAM a 256 MB nesdílené paměti grafické karty. Operační systém Ubuntu 10.04, Blender ve verzi 2.49b.



Obrázek 22: Graf vývoje počtu segmentů v závislosti na počtu kroků



Obrázek 23: Graf časové náročnosti v závislosti na počtu kroků simulace

6 Závěr

Cílem této diplomové práce bylo vytvoření skriptu v jazyce Python simulující růst stromu v grafickém prostředí programu pro počítačovou 3D grafiku Blender.

Po prostudování problematiky fungování simulací růstu rostlin byla vytvořena základní představa o fungování mechanismu vlastní simulace.

Vzhledem k omezeným možnostem tvorby animací v prostředí Blender byla pozměněna vize tvorby animace růstu rostliny, jejíž původní představa se ukázala být pro naše potřeby nerealizovatelná.

Následně jsem vytvořil jednoduchý skript, obsluhující rostoucí stromovou strukturu pomocí nástrojů objektově orientovaného programování.

Takovýto jednoduchý skript byl následně rozšířen o schopnost modelování připravené stromové struktury do trojrozměrného modelu v okně Blenderu.

Funkční generátor rostliny, pracující v této chvíli čistě s náhodnými hodnotami, se po konzultacích s vedoucím diplomové práce rozšiřoval o důležité funkce, realizující simulaci prostřednictvím reakcí na vnější i vnitřní aspekty.

Skript je schopen reagovat na nedostatek prostoru při koncentraci většího množství větví na omezeném prostoru a omezit růst takovýchto větví. Stejně tak počítá přísun světla a dokáže utlumit či úplně zastavit růst těch větví, které mají malý přístup ke světlu.

Na tvar koruny má velký vliv jak nastavení požadovaného tvaru (zda má být koruna vysoká nebo spíše široká), tak i jeho reakce na světlo, které ve volitelné míře ovlivňuje způsob větvení a směr růstu větví.

Takto připravený simulátor byl kvůli lepšímu vzhledu hotové rostliny rozšířen o listy na větvích a doplněn materiálem, který dává jednotlivým částem modelu realističtější vzhled. Dále bylo provedeno několik drobných zlepšení s cílem dosáhnout realističtějších proporcí mezi větvemi.

Nakonec byl celý skript opatřen funkcemi pro vizualizaci hotového modelu v požadovaných krocích. To umožňuje vytvoření fotorealistického obrázku a jeho uložení do uživatelského adresáře, odkud mohou být jednotlivé snímky dále zpracovávány, včetně tvorby videa.

Taktéž je možné hotový model v rámci možností Blenderu exportovat do souborů jiných 3D grafických programů a následně je využít pro obohacení scény.

Literatura

[1] ŽÁRA, Jiří - BENEŠ, Bedřich - SOCHOR, Jiří - FEKEL, Petr: *Moderní počítačová grafika*
ISBN 80-251-0454-0, Vydavatelství Computer Press, Brno 2004.

[2] POKORNÝ, Pavel: *Naučte se 3D grafiku*
ISBN 80-7300-244-2, Vydavatelství BEN - technická literatura

[3] HARMS, Daryl - MCDONALD, Kenneth: *Začínáme programovat v jazyce Python*
ISBN 978-80-251-2161-0, Vydavatelství Computer Press, Brno 2003

[4] *Python Programming Language - Official Website [online]*. Dostupné z:
`http://www.pythong.org`

[5] *blender.org - Home [online]*. Dostupné z:
`httl://www.blender.org`

[6] *Blender API Documentation*. Dostupné z:
`http://www.blender.org/documentation/249PythonDoc/`

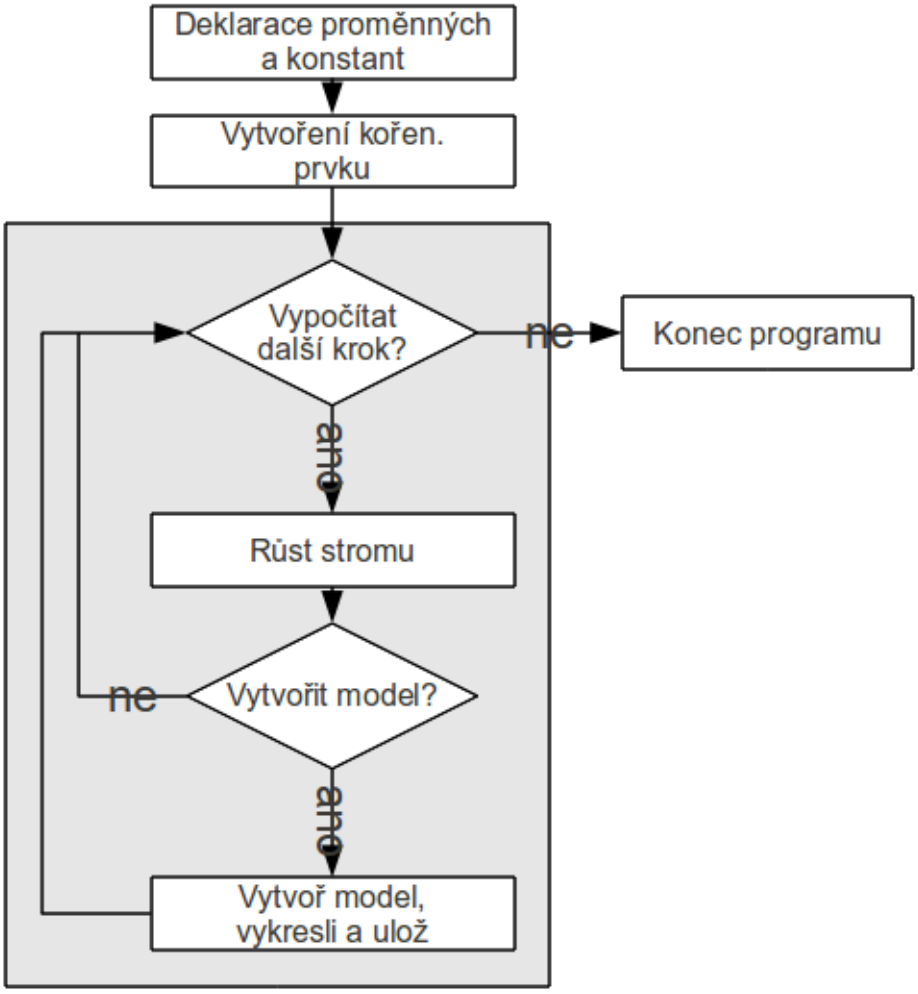
[7] *L-System [online]*. Publikováno 17. 5. 2011. Dostupné z:
`http://www.lsystem.liquidweb.co.nz/`

[8] LUFT, Thomas. *An Ivy Generator [online]*. Publikováno 6. 10. 2008. Dostupné z:
`http://www.graphics.uni-konstanz.de/~luft/ivy_generator/`

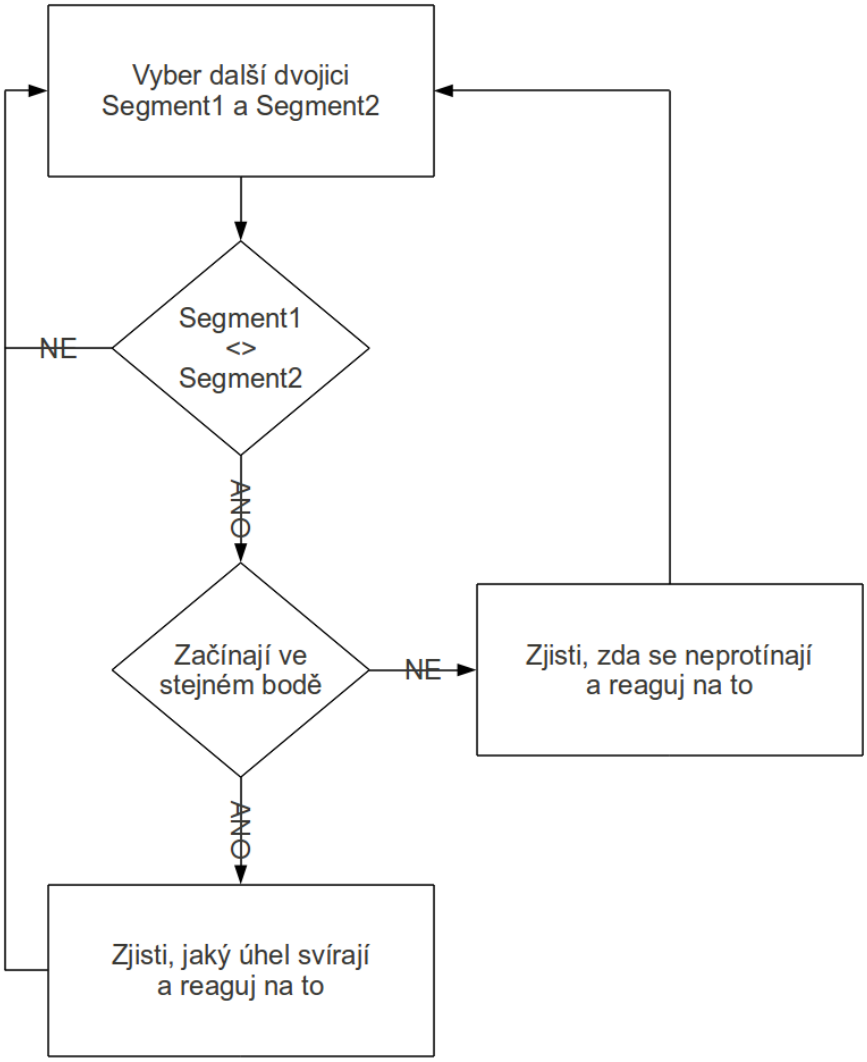
[9] FABIÁN, Petr: *Generátory vegetace*
Liberec 2009

7 Přílohy

Příloha A: Schéma běhu programu



Příloha B: Schéma řešení kolizí mezi dvojicemi segmentů



Příloha C: Počet segmentů kmenu ve scéně

krok	počet objektů		
	minimální	průměrný	maximální
1	1	1,0	1
2	5	6,0	7
3	10	12,2	15
4	16	20,6	27
5	24	33,4	47
6	36	54,0	79
7	55	87,4	130
8	85	139,8	207
9	130	221,0	325
10	198	345,8	505
11	297	534,2	772
12	444	816,6	1 168
13	661	1 236,6	1 752
14	981	1 854,2	2 615
15	1 452	2 765,4	3 869
16	2 138	4 051,6	5 692
17	3 127	5 804,0	7 941
18	4 555	8 108,0	10 190
19	6 626	1 0540,6	12 439
20	8 697	12 973,2	14 688