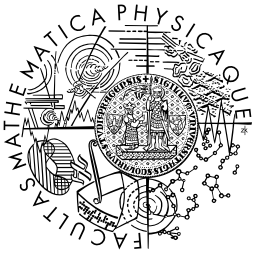


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Vojtěch Tázlar

Procedurální generování stromů

Katedra software a výuky informatiky

Vedoucí bakalářské práce:	doc. Ing. Jaroslav Křivánek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové a datové inženýrství

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 18. 7. 2018

Děkuji vedoucímu mé bakalářské práce, doc. Ing. Jaroslavu Křivánkovi, Ph.D., za veškeré rady, pomoc a trpělivost při jejím vypracování.

Název práce: Procedurální generování stromů

Autor: Vojtěch Tázlar

Katedra: Katedra software a výuky informatiky

Vedoucí bakalářské práce: doc. Ing. Jaroslav Křivánek, Ph.D., Katedra software a výuky informatiky

Abstrakt: Při 3D modelování reálného světa je mnohdy zapotřebí umístit do scény stromy. Konkrétní model ale nemusí vyhovovat představám uživatele a použití dedikovaného software může být nepohodlné - z důvodu složitého ovládání, nemožnosti integrace do používaného systému, kvality modelů či ceny. V době psaní této práce existuje několik profesionálně používaných high-end aplikací, k jejichž použití ale potenciální příležitostný uživatel nepřistoupí a to hlavně z důvodu ceny. Pak mu zbývají pouze volně dostupné aplikace, které jsou mnohdy zastaralé a trpí různými nedostatky. Proto má smysl pokusit se vyvinout low-end aplikaci, právě pro příležitostné uživatele, kde není nejdůležitější maximální síla nástroje, ale hlavně jednoduchost a přirozenost ovládání s uspokojivými výsledky. V rámci práce analyzujeme způsoby modelování stromů za pomoci existujících aplikací a popisujeme známé algoritmy a přístupy pro generování jejich modelů. Na základě toho implementujeme vlastní nástroj schopný generovat různorodé druhy stromů a přizpůsobovat jejich modely případným překážkám ve scéně.

Klíčová slova: 3D počítačová grafika, 3D procedurální modelování

Title: Procedural tree generation

Author: Vojtěch Tázlar

Department: Department of Software and Computer Science Education

Supervisor: doc. Ing. Jaroslav Křivánek, Ph.D., Department of Software and Computer Science Education

Abstract: During 3D modelling of the real world, it is often necessary to place trees into the scene. However particular model may not suit users´ needs and the use of a dedicated software can be inconvenient - due to complex control, impossible integration into the used system, model quality or price. At the time of writing this work, there exist a few professional high-end applications that are not suitable for use by a potential casual user, mainly because of their price. Then there are only freely available applications which are often obsolete and suffer from various shortcomings. Therefore, it makes sense to try to develop a low-end application, targeted for casual users, where the power of the tool is not as important as simplicity and nature of control and satisfactory results. We analyze ways of modelling trees using current applications and describe known algorithms and approaches to the generation of tree models. Based on this, we implement our own tool capable of generation of diverse tree types and adaptation of their models to obstacles in the scene.

Keywords: 3D computer graphics, 3D procedural modeling

Obsah

Úvod

V 3D počítačové grafice je pro tvorbu scény nutné mít vhodné objekty, které po ní rozmisťujeme. Tvorbou modelů všech možných druhů se zabývá 3D modelování, jehož součástí je také modelování stromů či rostlin obecně.

Pokud je proces tvorby modelů automatizován do podoby programu, který je většinou na základě zadaných parametrů generuje, pak hovoříme o tzv. procedurálním generování. V kontextu tvorby modelů či textur můžeme také použít termín procedurální modelování.

Jelikož takřka každá scéna exteriéru nějakou vegetaci obsahuje - a to ve velkém množství, jsou nároky na různorodost (podle typu a specifik scény) a počet modelů vysoké. Tvořit je všechny čistě ručně by bylo prakticky nemožné, a proto také existuje řada aplikací zaměřujících se na usnadnění jejich výroby.

Bohužel na trhu jsou jen dva typy těchto aplikací. Několik profesionálních programů s mnohdy dlouholetou tradicí, které skvěle pokrývají potřeby filmového či herního průmyslu - za adekvátně velkou cenu, a pak řada volně dostupných aplikací, čemuž odpovídá jejich kvalita. Neexistuje tedy žádná „jasná volba“ pro příležitostné uživatele.

Proto v rámci této práce provádíme nezbytné kroky k vývoji právě takového nástroje. Od analýzy dostupných aplikací a známých algoritmů až po výběr a implementaci vhodného řešení.

Cíle implementace

- Jednoduchost a přirozenost ovládání
  - Zavedením co možná nejmenší množiny parametrů, které jsou jasně definovány a mají předvídatelný dopad (např. pomocí parametrů, které by reprezentovaly vlastnosti výsledných stromů).
  - Ideálně musí být implementace více prostředkem pro generování modelů, kde proces jejich tvorby je co možná nejvíce automatizován, nežli modelovacím nástrojem, kde se předpokládá, že uživatel celý proces tvorby modelu stromu (od kořenu po listy) povede sám.
- Netriviální rozsah typů generovatelných tvarů stromů
  - Velké rozdíly nejsou jenom mezi jehličnatými a listnatými stromy, ale i mezi jednotlivými druhy v rámci těchto skupin. Například bříza má velice odlišné rysy od dubu apod.
  - Tato podmínka jde do velké míry proti jednoduchosti ovládání, ale aby výsledná implementace měla praktický smysl, je nutné dosáhnout přijatelného kompromisu.
  - Výsledný tvar je definován strukturou větví stromu - jejich vzájemnou organizací v rámci koruny a obecnými vlastnostmi (např. jak moc přímé či pokroucené jsou). Protože požadujeme, aby nástroj byl co nejvíce automatizován, znamená to, že generované struktury větví

musí být kvalitní natolik, aby nutně nevyžadovaly manuální úpravy a opravy.

- Věrohodnost generovaných modelů
  - Je hlavním předpokladem pro praktickou využitelnost výsledných modelů.
  - Protože při pohledu z dostatečné vzdálenosti na model ve scéně nejsou vlastně žádné větve vidět, bylo by možné případné nepřesnosti ve struktuře větví zakrýt pomocí listů, což je v souladu s realitou. Takové modely jsou ale jen těžko využitelné ve scénách reprezentujících podzim či zimu, kde jejich nedostatky nejsou nijak zakryty.
- Přizpůsobivost modelu vůči předpokládané pozici ve scéně
  - Vzhledem ke snaze omezit nutnost uživatele vygenerovaný model manuálně upravovat, je nutné umožnit generátoru, aby byl schopen se přizpůsobit zadané pozici v konkrétní scéně. Prakticky se jedná o problematiku toho, aby generátor s překážkami ve scéně automaticky interagoval, což může ovlivnit strukturu výsledného modelu stromu. Alternativou, obzvláště u volně dostupných aplikací, je vytvoření modelu stromu bez omezení a pozdější manuální odstranění jeho větví, které s objekty ve scéně kolidují. Proto umožnění generování na základě konkrétní pozice ve scéně nejenom ušetří pozdější práci uživatele, ale mělo by i zlepšit věrohodnost výsledného modelu.
  - Scénu předpokládáme ve formě polygonálního meshe.

Struktura práce

Vzhledem k vytyčeným cílům nejprve provádíme analýzu existujících aplikací (kapitola 1.) a seznámení s možnými algoritmickými postupy (kapitola 2.), ze kterých vybíráme nejvhodnější pro naši implementaci. V kapitole tři detailně vysvětlujeme zvolený algoritmus, komentujeme změny, které jsme v něm provedli a zmiňujeme možné problémy a obecné postřehy k jeho fungování. Ve čtvrté kapitole popisujeme použité technologie, překlad, strukturu a další implementační detaily naší aplikace. V páté kapitole prezentujeme výsledné modely stromů vygenerováné naší aplikací. Nakonec na základě vlastností implementace provádíme hodnocení splnění cílů a nastínění dalšího potenciálního směřování vývoje aplikace (kapitola Závěr). První přílohou práce je uživatelská dokumentace aplikace. V druhé příloze je obsah přiloženého CD.

1. Existující aplikace

V této kapitole prezentujeme stávající nejčastěji používané aplikace pro tvorbu modelů stromů. Analyzujeme výhody a nevýhody jejich postupů výroby modelů, ovladatelnost a další vlastnosti, které mohou ovlivnit rozhodnutí, zda je použít, či nikoliv.

Prezentujeme pět komerčních nástrojů v pořadí od aplikací prakticky čistě modelovacích po generovací. Také komentujeme volně dostupné nástroje. V závěru posuzujeme, jakým směrem, v kontextu existujících aplikací, by se naše implementace měla vydat.

1.1 Společné rysy

Všechny nástroje po zadání parametrů vytváří výsledný model stromu na základě nějaké formy náhodnosti tak, abychom při jednom konkrétním nastavení byli schopni vygenerovat více modelů stejného typu. Tím je navíc umožněno ze znalosti parametrů a počátečního nastavení náhodného generátoru reprodukovat totožný výsledek. V tomto ohledu jsou všechny nástroje automatizované a tedy generátory modelů, a nikoliv modelovacími nástroji. Rozdíly jsou v typu, rozsahu a způsobu zadávání všech nutných parametrů pro tvorbu výsledného modelu. Jak uvidíme u některých nástrojů, zadávání parametrů skutečně připomíná postupné modelovaní stromu od kořene k listům, zatímco jindy se jedná o nastavení několika obecných vlastností, na základě nichž je vygenerován celý strom.

1.2 GrowFX, SpeedTree a PlantFactory

1.2.1 Modelovací systém

Jak GrowFX [1], tak SpeedTree [2] i PlantFactory [3] sdílí fundamentálně shodný systém modelování stromů. Ten spočívá v postupném nastavování objektů reprezentujících jednotlivé úrovně stromové struktury, tj. kmene, větví prvního, druhého až n-tého řádu a listů. Objekty mezi sebou mohou mít vztah 1:n, kde např. na větvích řádu n může být libovolný počet větví řádu n+1 (vztahy mezi všemi objekty tedy tvoří stromovou strukturu - dále nazývanou parametrický strom). Každý z těchto objektů lze nastavit striktně izolovaně vůči všem ostatním, včetně objektů totožného typu. Pokud tedy chceme generovat model, musíme ho nejprve celý (od kmenu po listy) zadat pomocí parametrického stromu.

Nutno zdůraznit, že např. objekt větví skutečně nereprezentuje jedinou větev ve výsledném stromě, ale větve daného řádu ve stromě, tedy potenciálně přesně neznámý počet. Proto je zapotřebí nastavit vztahy mezi synem a otcem v parametrickém stromě - např. distribuci objektu větví po celkové délce kmene. To je příkladem parametru, který nelze reprezentovat jedinou hodnotou, ale většinou je definován pomocí funkce, jež cílovou distribuci popisuje. Uživatel zadává funkce pomocí grafu jejich průběhu.

Generování rozdílných modelů se stejnými parametry je umožněno přinejmenším dvěma způsoby - aplikací šumových funkcí na jednotlivé objekty (to má za

následek např. zprohýbání původně rovného kmene) a parametry zadanými pomocí rozsahů, ze kterých jsou náhodně vybrány konkrétní hodnoty.

Výhodou tohoto systému je možnost zadat libovolný parametrický strom a potenciálně tak reprezentovat takřka neomezené množství druhů stromů. To je hlavní důvod, proč jsou systémy tohoto typu na popředí žebříčku komerčních aplikací. Na druhou stranu vhodné nastavení všech parametrů mnohdy není zcela zřejmé. Navíc parametry často nejsou jen skalární hodnoty, ale jedná se i o rozsahy a funkce popisující vlastnosti, které uživatel musí v systému nastavit. Navíc o to, aby model jako celek vypadal dobře, se musí uživatel postarat sám, neboť mezi interním nastavením jednotlivých objektů v parametrickém stromě není žádný vztah.

U aplikací založených na tomto systému je prakticky nemožné mít všechny potřebné parametry pro nastavení (i jednoduchého) modelu na jedné obrazovce. To může být značně nekomfortní hlavně pro někoho, kdo není zvyklý se v dané aplikaci pohybovat. Právě toho bychom se chtěli vyvarovat vzhledem k cílení na příležitostné uživatele.

1.2.2 GrowFX

Oproti všem ostatním nástrojům v této kapitole je GrowFX [1] (obrázek 1.1) jediný, který není přímo zaměřen na generování modelů stromů, ale vegetace obecně. Proto je v něm výše popsaný postup modelování o úroveň složitější. Můžeme si ho představit tak, že jednotlivé objekty reprezentují křivky (většinou úsečky), jejichž výsledný tvar je upravován jejich nastavením a aplikací různých modifikátorů, což jsou další soustavy parametrů, které můžeme v libovolném počtu aplikovat na každý objekt. Tedy zatímco v obecném parametrickém stromě měl objekt jasnou roli (kmen, větve, listy, ...) a tomu i odpovídající potenciálně biologické parametry, zde se jedná o systém úprav vlastností křivek tak, aby nakonec reprezentovaly strukturu stromu, tedy o čistě matematický model. Tento systém je nutný, aby bylo možné modelovat libovolné rostliny, ale specificky pro modelování stromů je zbytečnou zátěží.

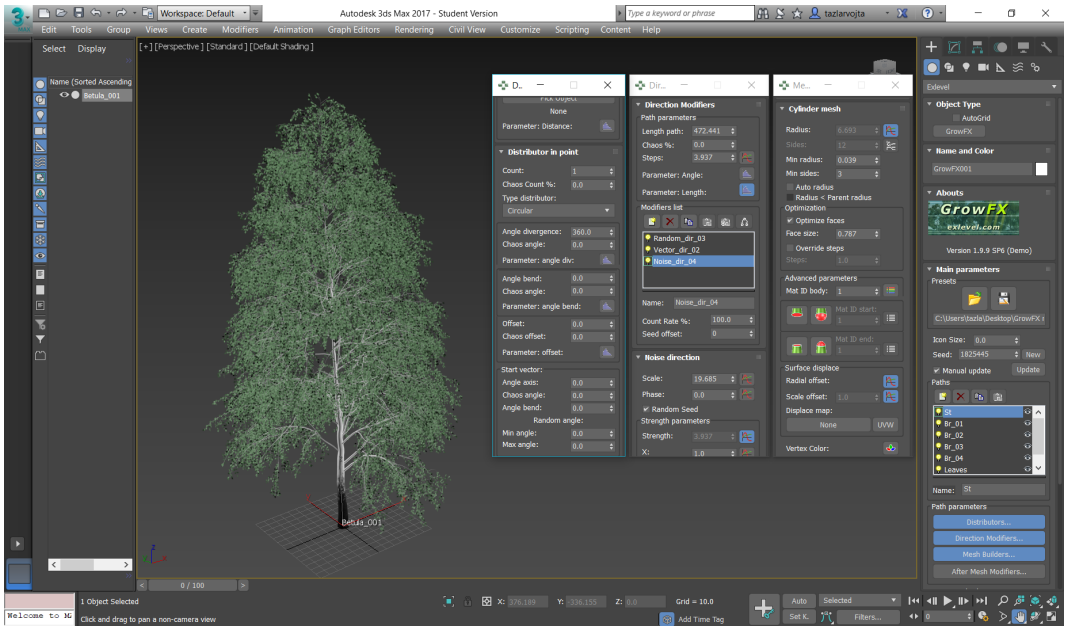
Růst kolem překážek můžeme uskutečnit, ale musíme pro to manuálně aplikovat modifikátory na objekty ve scéně tak, aby buď přitahovaly, nebo odpuzovaly směřování křivek v matematickém modelu.

Nástroj existuje pouze ve formě pluginu pro Autodesk 3ds Max [4], což je dostačují pro to, aby o něm mnoho potenciálních zájemců ani neuvažovalo.

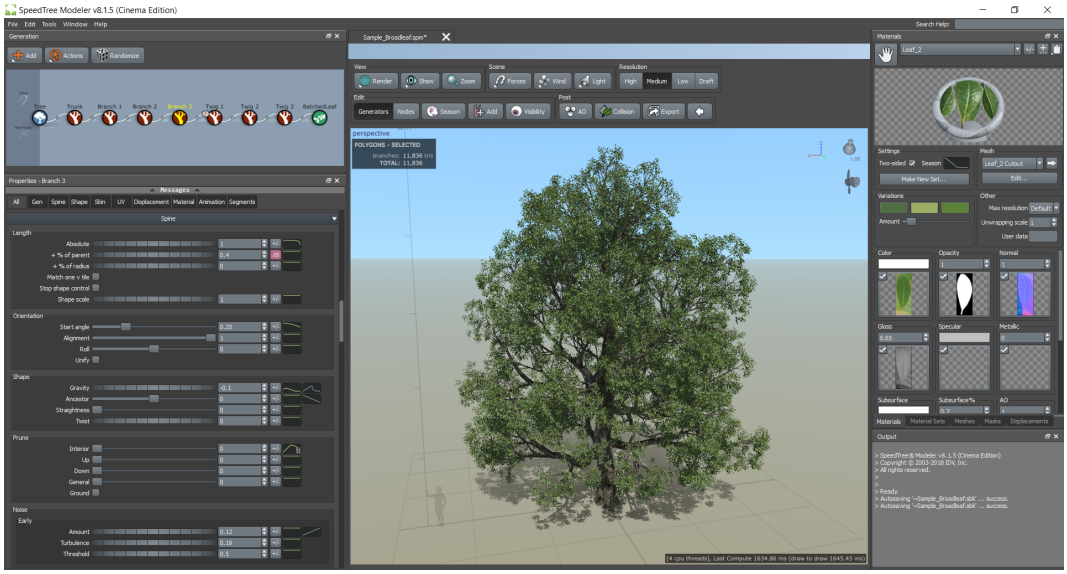
1.2.3 SpeedTree a PlantFactory

Jak SpeedTree [2], tak PlantFactory [3] jsou samostatné aplikace zaměřené na tvorbu stromů na bázi výše popsaného parametrického stromu. Způsob fungování a základní prvky uživatelského rozhraní (viz obrázek 1.2) jsou tedy u nich velmi podobné.

SpeedTree se na trhu pohybuje od roku 2002 a postupně se stal nejznámějším a nejúspěšnějším programem pro modelování stromů pro film a hry, čemuž odpovídají specificky cílené balíčky jako SpeedTree Cinema a SpeedTree Games.



Obrázek 1.1: GrowFX 1.9.9 - Uživatelské rozhraní.



Obrázek 1.2: SpeedTree 8.1.5 - Uživatelské rozhraní.

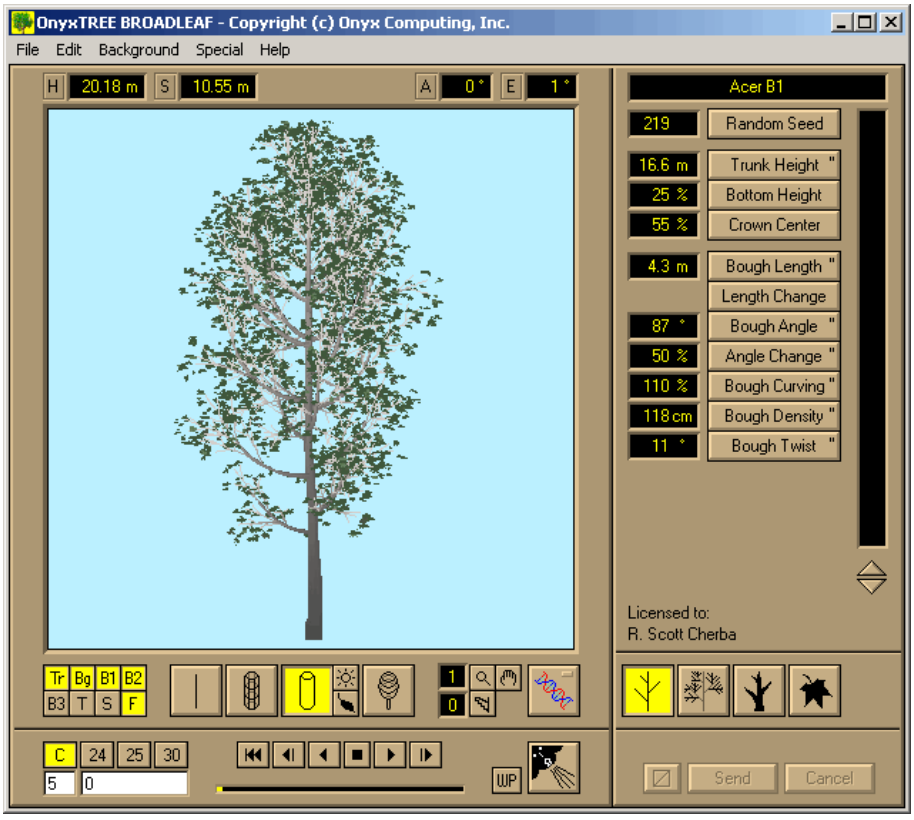
V levém horním rohu můžeme vidět reprezentaci parametrického stromu a pod ní u mnoha parametrů ikonku grafu reprezentující možnost zadat jejich nastavení ve formě grafu funkce.

PlantFactory cílí hlavně na offline modely vysoké kvality (např. pro architektonickou vizualizaci). Produkt ale provází mnoho sporných obchodních rozhodnutí, a proto není zdaleka tak úspěšný, jak by mohl být.

Oba programy umožňují nahrát libovolný objekt ve formě meshe a simulovat růst kolem něj. Navíc mají mnoho přidaných funkcí - např. definování tvaru větví pomocí tahu štětcem, simulaci větru či animaci růstu.

1.3 OnyxTREE

OnyxTREE [5] je trojice velmi starých, samostatných aplikací specificky zaměřených na listnaté stromy, jehličnaté stromy a palmy. Kromě malých změn se jejich vývoj prakticky zastavil v roce 2005, což je hlavně vidět při pohledu na uživatelské rozhraní (viz obrázek 1.3), ale i přesto jsou stále používané.



Obrázek 1.3: OnyxTREE BROADLEAF - Uživatelské rozhraní.
Zdroj: cherba.com [6].

Modelovací systém se zdá být na první pohled jednodušší, neboť množina parametrů je pevně dána. Ve skutečnosti se ale jedná o velmi omezenou část potenciálně nekonečného parametrického stromu, čemuž odpovídá typ a způsob nastavování parametrů. Konkrétně je nám dovoleno nastavit vlastnosti kmene, pěti úrovní větví a listů. Každou úroveň reprezentuje právě jeden objekt a vztah, mezi po sobě jdoucími úrovněmi, je striktně 1:1. Do aplikace nelze nijak nahrát modely cizích objektů, tedy stromy lze modelovat pouze ve volném prostoru. To, že i takové nástroje jsou stále používané, je dobrou reprezentací stavu trhu s generátory stromů.

1.4 The Grove

The Grove [7] je jediným z prezentovaných nástrojů, který skutečně budí dojem čistého generátoru, protože modely vytváří na základě simulace přírodních procesů provázejících růst stromů. Tomu odpovídá i typ parametrů - jedná se o čistě skalární hodnoty popisující konkrétní vlastnosti jako rozmístění pupenů, citlivost na světlo, tíhu větví a listů (pro výpočty ohybu větví), míru opadávání větví a mnohé další. Okamžitě po nastartování aplikace, bez jakéhokoliv nastavení, je možné vytvořit model celého stromu a dopad změn parametrů na něm sledovat. To dělá nástroj velmi uživatelsky přívětivý a možný s úspěchem použít i bez studia návodných materiálů.

Pro interakci s prostředím jsou k dispozici konkrétní objekty, u kterých jde dále nastavit jejich vliv na strom - zda rostoucí větve přitahují, odpuzují nebo čistě blokují.

The Grove existuje pouze ve formě pluginu pro Blender [8], což je velkým omezením pro kohokoliv, kdo používá jiný systém. Přesto je překvapující, jak málo informací a ukázek z tohoto nástroje je k nalezení, vzhledem k tomu, že jeho vývoj aktivně pokračuje dodnes. Pravděpodobným důvodem je, kromě nedostatku informací a neexistence zkušební verze to, že profesionální uživatel má raději naprostou kontrolu nad vznikajícím modelem. To samozřejmě nemá v případě simulátoru, kde neví, jak přesně jsou nastavované parametry uplatňovány.

1.5 Volně dostupné aplikace

Volně dostupných aplikací je celá řada. Většina z nich není žádnou formou udržována a nijak se nevyvíjí. Netriviální množství existuje jen ve formě pluginů do konkrétních systémů. Obecně platí, že kvalita modelů, které jsou schopny vytvářet, je pochopitelně horší než u komerčních aplikací.

Přístupy, které jsou v nich použity ke generování stromů, jsou také daleko různorodější - od stejného přístupu jako OnyxTREE (viz kapitola 1.3), po implementace všech možných druhů algoritmů aplikovatelných na tuto problematiku (viz kapitola 2).

Některé z nám známých volně dostupných nástrojů jsou: ngPlant, Arbaro, Tree[d], ZTree for ZBrush a Tree It.

1.6 Závěr průzkumu

Z průzkumu je zřejmé, že většina komerčních aplikací je založena na nějaké formě parametrického stromu, který zprostředkovává silný nástroj pro definování cílového tvaru stromu. Cenou za to je nutnost manipulovat s velkým množstvím parametrů a dobře se v systému orientovat, což je velkou zátěží obzvláště pro nezkušeného uživatele. Vzhledem k našim cílům je tedy daleko bližší přístup The Groove, kde můžeme celé stromy generovat okamžitě po spuštění aplikace a projevy změn parametrů tak sledovat v kontextu celého výsledného modelu.

Volně dostupné aplikace volí různorodé přístupy ke generování stromů, ale žádná, nám známá v době psaní práce, se typově The Groove nepodobá. Proto si myslíme, že vývoj nové aplikace, s podobným způsobem ovládání, má smysl.

2. Procedurální postupy generování stromů

Na základě průzkumu existujících aplikací (kapitola 1) je nám jasné, že pro splnění našich původních cílů a vybraného směřování uživatelského rozhraní je vhodné zvolit do velké míry generativní postup, který by byl schopný, pomocí limitovaného množství počátečních parametrů, generovat strukturu celého stromu. Neodmítáme možnost uživatele do procesu případně zasahovat, čímž by byl schopen vytvořit větší množství tvarů stromů, ale požadujeme, aby byl i bez toho schopen vytvořit modely přijatelné kvality.

V této kapitole tedy prezentujeme některé z možných přístupů ke generování stromů a konkrétní algoritmy či systémy, které je reprezentují. Ty pak posuzujeme z hlediska našich cílů a vybíráme z nich ten nejvhodnější.

2.1 Přístupy k generování stromů

Je nutno podotknout, že když mluvíme o algoritmech nebo přístupech k generování stromů, jsou tím myšleny postupy jak vytvořit věrohodné uspořádání větví ve stromě. Výslednou strukturu větví pak reprezentujeme soustavou vrcholů a hran mezi nimi (segmenty větví), což je abstrakce, která je na pozadí prakticky všech nástrojů.

Jedním z přístupů, jak takovou reprezentaci vystavět, je považovat strom za rekurzivně se větvící strukturu. To vede na systémy založené hlavně na základě geometrických parametrů (velikost úhlů mezi větvemi, délky segmentů větví, ...). To by ale samo o sobě mělo za následek pravidelnou strukturu, kterou u stromů rozhodně nechceme. Proto je do takových systémů přinejmenším přidán nějaký prvek náhody. Příkladem postupu považujícího strom za rekurzivně se větvící strukturu jsou různé varianty Lindenmayerových systémů (2.2).

To, že nechceme pravidelnou strukturu, neznamená, že by ve stromu žádná nebyla. Například rozmístění listů a pupenů na větvi (tzv. fylotaxe) je striktně pravidelné. Růst stromu je ale ovlivněn mnoha jevy - jak externími, tak interními. V praxi se jedná o vlastnosti prostředí (dostupnost volného prostoru pro růst, množství světla, velikost srážek, množství živin v půdě, síla větru, ale například i vliv škůdců a mnohé další), které se navíc mohou teoreticky při růstu průběžně měnit a každý druh stromu na ně a jejich změny potenciálně reaguje jinak. Mezi interní jevy ve stromě může patřit přerozdělování živin (aby více rostly nějak preferované větve), ohýbání větví pod vlastní vahou či jejich opadávání a různé druhy tropismů - což jsou změny orientace na základě vnějších stimulů (u bylin je např. nejznámější reorientace za světlem - fototropismus).

Mnoho z výše zmíněných jevů se výrazněji projevuje až s vyšším věkem stromů. Jak množství těchto jevů, tak otázka, jak je případně simulovat v rámci programu, jsou důvodem, proč dodnes neexistuje osvědčená metoda, jak proceduralně generovat například stoleté či starší stromy. Naopak, díky tomu, že mladé stromky jimi tolik ovlivněny nejsou, je i jejich struktura daleko pravidelnější a použití rekurzivních systémů často dostačující.

Pochopitelně při snaze dosáhnout co nejvyšší věrohodnosti procedurálně generovaných modelů dochází k pokusům o uplatnění alespoň nejvýraznějších ze zmíněných jevů a to ve většině přístupů, včetně rekurzivních systémů, které původně s přírodními jevy neměly nic společného. Nutno podotknout, že i v případě postupů, které s prostředím nijak neinteragovaly, může být možné aplikovat některé z interních jevů, jako je ohýbání větví, nebo externí jevy ve zjednodušené formě - např. zadání preferovaného směřování větví zastupující blíže nespecifikovaný tropismus.

Mylná by však byla představa, že alespoň nějakou formou je nutné procesy ve stromě nebo mezi stromem a prostředím řešit. Existují postupy pro generování struktury stromu, ve kterých to ani není prakticky možné a i přesto dosahují velmi dobrých výsledků. Jako takový případ popisujeme částicové systémy (2.3).

Na druhé straně spektra jsou algoritmy, které generují strom na základě simulace jeho růstu. Tedy jak procesy uvnitř stromu, tak jeho interakce s okolním prostředím jsou naprosto stěžejní. Mezi ně patří tzv. Sebe-uspořádající modely stromů (2.4).

2.2 Lindenmayerovy systémy (L-systémy)

L-systémy v základní variantě můžeme považovat za variaci na bezkontextové gramatiky, od nichž se ale v mnohém odlišují.

Formálně je definujeme jako trojici:

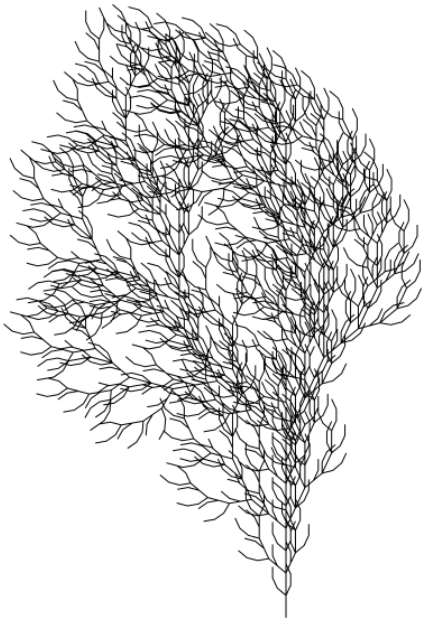
- Abeceda - neprázdná množina symbolů (posloupnost symbolů tvoří slovo),
- Axiom - počáteční stav systému ve formě neprázdného slova,
- Konečná množina přepisovacích pravidel - pravidlo definuje přepis symbolu z abecedy na konkrétní slovo.

Pro symboly bez přepisovacího pravidla je předpokládána implicitní existence identity - přepisu sama na sebe. Už z tohoto pohledu L-systém není klasickou gramatikou, neboť formálně neobsahuje terminální symboly. Dalším rozdílem je to, že axiom je potenciálně tvořen více symboly.

U L-systémů je stěžejní způsob aplikace přepisovacích pravidel. Není to tak, že by se v každém kroku vybral jeden symbol z řetězce reprezentujícího momentální stav, ale pravidla jsou vždy uplatněna na všechny symboly zároveň. Proto L-systémy kategorizujeme jako tzv. paralelní přepisovací systémy.

Opakovaným užitím pravidel tedy získáme řetězec symbolů, který je nutno interpretovat. Častým způsobem, obzvláště v oblasti počítačové grafiky, je tzv. želví grafika. Představa je taková, že želva za sebou vleče štětec a symboly interpretuje jako příkazy ovlivňující její pohyb. Tím, jak se pohybuje, kreslí výsledný obraz (viz obrázek 2.1).

Základní varianta L-systémů je ryze deterministická a je v praxi velmi často používaná např. pro kreslení fraktálů. Pro rostliny a obzvláště stromy je ale nedostatečná, a proto bylo zavedeno mnoho rozšíření, díky nimž se použití L-systémů stalo prakticky nejznámějším přístupem k procedurálnímu generování rostlin.



Obrázek 2.1:
Příklad jednoduchého L-systému.

Axiom: F

Přepisovací pravidlo:

F => FF+F[+F−F−F]−[−F+F+F]

Želva interpretuje F jako pohyb vpřed, + a − jako otočení doprava respektive doleva o 25°, dvojice závorek ⌊ a ⌋ slouží po řadě k uložení momentálního stavu želvy na zásobník a nahrání stavu na jeho vrcholu.

Abychom byli schopni rozhodnout, zda použití L-systémů, jako našeho generativního postupu, je vhodné vzhledem k vytyčeným cílům, uvádíme některá z jejich rozšíření a roli, kterou mají v kontextu generování rostlin.

1. Stochastické L-systémy umožňují existenci více přepisovacích pravidel pro jeden symbol. Volba mezi nimi je uskutečněna náhodně na základě pravděpodobnosti přiřazené každému z nich. Jsou vhodné pro generování malých rozdílů mezi jedinci stejného druhu.

2. Kontextové L-systémy, kde výběr konkrétního přepisovacího pravidla pro znak závisí na jeho okolních symbolech, jsou zapotřebí pro propagování signálů uvnitř rostliny - např. živin či hormonů ovlivňujících, které části mají růst více, nebo méně.

3. Parametrické L-systémy rozšiřují každý symbol o libovolný počet a druh parametrů (obvykle se jedná o čísla). Volba přepisovacího pravidla probíhá v závislosti na parametrech symbolu, které může přepisovací pravidlo měnit. Jsou důležité pro realistickou simulaci růstu rostliny. Například pokud vyžadujeme, aby větev rostla jen do cílové délky, nebo chceme, aby začala růst s nějakým zpožděním, může parametr sloužit jako jednoduché počítadlo.

4. Otevřené L-systémy (viz Měch, Prusinkiewicz, 1996 [9]) prezentují systém, ve kterém dochází k výměně informací mezi generovaným modelem (interně L-systém) a prostředím. Díky tomu jsme schopni potenciálně simulovat vliv externích jevů, ať už se jedná o dostupnost prostoru nebo světla, či čehokoliv jiného.

Výše zmíněná rozšíření nejsou disjunktní, naopak všechna mohou být použita zároveň. To adekvátně zvyšuje obtížnost toho, jak odvodit vhodný axiom, přepisovací pravidla a interpretaci želvy vzhledem k cílovému tvaru, kterého chceme dosáhnout.

2.3 Částicové systémy

Princip částicových systémů na problematiku generování stromů aplikovali Rodkaew a kol., 2003 [10]. Motivace pro vznik systému spočívala v myšlence, že k přesunu vody a živin mezi kořenem, listy a větvemi, by měl být strom vhodně strukturálně přizpůsoben.

Praktická realizace spočívá v jednoduchém algoritmu:

1. Rozmístíme částice na okraji koruny, ideálně na pozicích listů a zvolíme bod na zemi reprezentující kořen.
2. Pro každou částici vypočítáme směr jejího pohybu jako součet normalizovaného směru k nejbližší částici a ke kořeni. Částici v tomto směru posuneme.
3. Pokud jsou dvě, nebo více částic dostatečně blízko u sebe, pak je sloučíme do jedné.
4. Opakujeme kroky 2 a 3 dokud poslední částice není sloučena s bodem reprezentujícím kořen.
5. Dráhy pohybu částic definují stromovou strukturu.

Algoritmus tedy generuje strom ve směru od listů ke kořeni, což i přes jeho motivaci rozhodně přírodní postup není. To pochopitelně činí snahu přidat do systému jakékoliv biologické procesy uvnitř stromu nebo vliv prostředí velmi obtížnou. Výsledný tvar stromové struktury závisí na počátečním rozložení částic, kterým můžeme ovlivnit hustotu větví. To se nám ale zdá jako velmi nepřirozený a hlavně těžko ovladatelný postup. Proto tento algoritmus nevyhovuje našim požadavkům.

2.4 Sebe-uspořádající stromy

Jedná se o pojem, kterým označili svou metodu Pałubicki a kol., 2009 [11]. Ta je založena na myšlence, že výsledný tvar stromu vzniká z sebe-uspořádajícího procesu (růstu), při němž neustále dochází k rozhodování o tom, zda, jak dlouhý a z kterého pupenu má vyrazit nový výhon. Jednotlivá rozhodnutí jsou reakcí na dostupný prostor, světlo či interní vlastnosti stromu. Algoritmus je tedy prakticky simulací přírodního růstu stromu. Metoda je iterativní, kde si každou iteraci můžeme představit jako roční cyklus.

V kontextu tohoto algoritmu se často hovoří o zdrojích. Jejich množství v orgánu stromu určuje, nakolik je výhodný pro jeho další rozvoj. U celé větve to může rozhodovat o tom, zda nemá ze stromu opadnout a u pupenu, jestli z něj má vyrůst nový výhon a případně to, jak bude dlouhý.

Postup v rámci jednoho cyklu simulace je následující:

1. Všechny aktivní pupeny zjistí z prostředí, kolik zdrojů mají (na základě přístupnosti k volnému prostoru, kam by mohly růst, nebo množství světla).

2. Pomocí mechanismů uvnitř stromu dojde k přerozdělení zdrojů mezi pupeny na základě toho, zda strom preferuje růst hlavních nebo vedlejších větví.

3. Podle množství zdrojů v jednotlivých pupenech dochází k jejich růstu. Výsledný směr je určen kombinací orientace pupenu, preferovaného směru vzhledem k prostředí a obecného tropismu nastaveného uživatelem.

4. V případě modelování na základě světelných podmínek můžeme podle množství dopadajícího světla na větev určit, zda je pro strom dostatečně výhodná a ponechat ji, nebo není a pak ji shodit.

Parametry celé simulace jsou nastavení jednotlivých modulů reprezentujících různé části prostředí nebo interních mechanismů stromu - proto jsou buď ve formě přírodních vlastností stromu, nebo alespoň takové, aby jejich dopad na výsledný strom byl předvídatelný.

Na původní práci navázali Longay a kol., 2012 [12], kteří algoritmus vylepšili a přidali nové biologické vlastnosti do simulace. Je nutné poznamenat, že obě práce směřují k generování stromů na základě malování jejich struktury štětcem, což je osvědčená metoda nabízená v komerčních aplikacích (např. v SpeedTree). Toto zaměření ale nijak neomezuje plně automatizované generování modelů, naopak je nástrojem, který lze nad systémem Sebe-uspořádajících stromů lehce implementovat.

2.5 Výběr postupu

Částicové systémy jsme vyřadili, neboť z pohledu všech našich cílů se jeví jako horší než zbývající dvě prezentované alternativy. Jak L-systémy, tak Sebe-uspořádající stromy nabízí možnost potenciálně generovat strukturálně rozdílné tvary stromů a interagovat s prostředím. U L-systémů ale máme obavy z problematiky toho, jak vytvářet vhodná přepisovací pravidla na základě obecných abstraktních vlastností popisujících strom. Na to neexistuje žádný nám známý postup a po uživateli rozhodně nemůžeme požadovat, aby se při nastavování generátoru dostal přímo do styku s L-systémy. Z těchto důvodů volíme Sebe-uspořádající stromy jako vhodnou metodu pro naši implementaci.

3. Generativní algoritmus

V této kapitole detailně popisujeme fungování zvoleného postupu pro generování modelů stromů - tedy fungování simulace na bázi Sebe-uspořádajících stromů (2.4). Články, ve kterých byl simulační systém prezentován, jsou Pałubicki a kol., 2009 [11] a Longay a kol., 2012 [12]. Zde vysvětlujeme interní fungování systému, změny oproti variantě z článků, které jsme provedli a zmiňujeme případné další problémy a postřehy.

Jelikož je simulace založena na rozhodování o tom, z kterých pupenů mají vyrůst nové větve, musíme nejprve vyřešit jejich polohu a orientaci, čímž se zabývá fylotaxe (3.1). K simulaci vlivů prostředí jsou použity adekvátní prostorový (3.2) a světelný model (3.3). Celým simulačním cyklem provází kapitola 3.4. Dále prezentujeme, jak jsme do systémů zavedli interakci se scénou (3.5) a způsob tvorby trojúhelníkového meshe nad vygenerovanou stromovou strukturou (3.6). V kapitole 3.7 komentujeme, co by bylo nutné řešit pro zavedení listů do modelu - ty ale v implementaci obsaženy nejsou. Kapitola 3.8 shrnuje celý prezentovaný algoritmus ve formě pseudokódu.

3.1 Fylotaxe

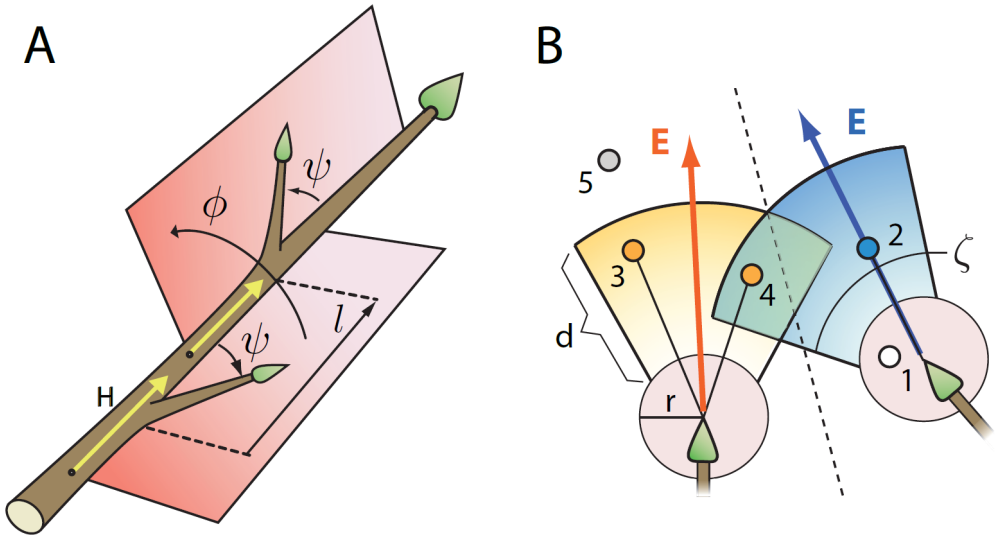
Struktura stromu je interně reprezentovaná systémem vrcholů a hran mezi nimi. Právě v jednotlivých vrcholech se vyskytují pupeny. Vždy definujeme pupen na konci větve ve směru její orientace (vrcholový pupen). Pokud vyroste, pak dochází k prodloužení větve. Pupenu orientovanému mimo osu větve říkáme boční (vedlejší) a v případě jeho růstu vzniká nová větev s počátkem ve vrcholu, kde se pupen nacházel. Fylotaktické uspořádání rozhoduje o počtu a orientaci bočních pupenů ve vrcholech. Je dáno fylotaktickým úhlem, který určuje vzájemné pootočení pupenů v po sobě jdoucích vrcholech vzhledem k větvi. Podle počtu a orientace pupenů v rámci jednoho vrcholu rozlišujeme uspořádání střídavé (jeden pupen), protilehlé (dva pupeny) a přeslenité (větší počet pupenů pravidelně uspořádaných kolem vrcholu).

Pro potřeby simulace je dalším důležitým parametrem úhel mezi orientací pupenu a větvě - tzv. boční či vedlejší úhel. Celé nastavení rozmístění pupenů je tedy vyjádřeno trojicí parametrů: fylotaktickým úhlem, bočním úhlem a typem uspořádání (viz obrázek 3.1A).

Z botanického hlediska je fylotaxe uspořádání listů na větvi. Protože se ale pupeny vyskytují mezi řapíkem listu a stonkem rostliny, je uspořádání listů shodné s uspořádáním pupenů.

3.2 Prostorový model

Prostorový model je založen na algoritmu kolonizace prostoru pro generování stromů (Runions, Lane a Prusinkiewicz, 2007 [13]). Model řeší dotazy pupenů, jakým směrem by z nich měla růst větev, na základě dostupnosti volného prostoru. Sám o sobě bere v potaz pouze vyplnění prostoru stromem samotným, ale přizpůsobujeme ho tak, aby byl schopen reagovat i na překážky ve scéně (3.5).



Obrázek 3.1: Fylotaxe a prostorový model. Zdroj: Longay a kol., 2012 [12].
A) Rozmístění pupenů na větvi je dáno vzdáleností mezi vrcholy l, fylotaktickým úhlem ϕ a bočním úhlem ψ.
B) Souboj pupenů o volný prostor. Tečky reprezentují značky prostoru. Pupeny obsazují sférické oblasti o poloměru r. Barevné oblasti znázorňují oblasti vnímání jednotlivých pupenů. Oblasti jsou definovány vzdáleností $r + d$ a úhlem ζ. Výsledný směr E je průměrem normalizovaných směrů od pupenu k jemu přiřazeným značkám.

Jedná se o postup, kterým každému z množiny pupenů přiřazujeme optimální směr růstu vzhledem k ostatním. To je právě jedním z příkladů sebe-uspořádání, neboť jeho výsledkem je taková stromová struktura, kde by se větve neměly protínat.

Sférickou oblast kolem pupenu prohlašujeme za jím obsazenou. Dále je ve směru orientace pupenu kuželovitá oblast definovaná pomocí úhlu a vzdálenosti (obrázek 3.1B), kterou nazýváme oblastí vnímání pupenu.

Výpočet vhodného směru růstu pupenů vzhledem k prostoru je následující:

1. Na vstupu máme množinu pupenů, ze kterých bude potenciálně růst větev a předpokládáme, že se tak bude dít pro všechny zároveň.

2. Každému pupenu vygenerujeme v jeho oblasti vnímání náhodně umístěné body vyznačující volný prostor - nazýváme je značkami prostoru.

3. Značky, které jsou v jakékoliv oblasti obsazené pupenem, vymažeme.

4. Zbývající značky přiřadíme vždy k nejbližšímu pupenu v jehož oblasti vnímání jsou (obrázek 3.1B).

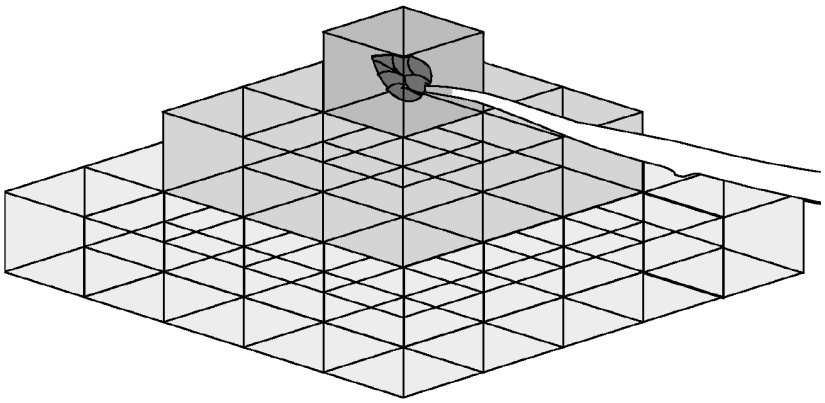
5. Pupeny, které nemají žádné značky, nemají prostor pro růst. U ostatních vypočítáme optimální směr jako součet normalizovaných směrů od pupenu k jemu přiřazeným značkám.

Tento proces velmi silně předchází možnosti toho, že by se některé větve při simulaci protnuly. Také má za následek optimálnější využití prostoru, což je botanicky korektní, neboť lepší pokrytí prostoru větvemi znamená více světla pro listy na nich. Abychom oba jevy posílili, ukládáme do prostorového modelu pozice všech vrcholů ve stromové struktuře a v kroku 2 výpočtu vymažeme i značky, které jsou od libovolného z nich ve vzdálenosti menší, než je poloměr sféry obsazení pupenu.

Je důležité uvědomit si, že vzhledem k náhodnému generování značek prostoru může být výsledný optimální směr orientován kamkoliv do prostoru vnímání pupenu. To v praxi znamená, že prostorový model neřeší jen vhodné uspořádání budoucích větví, ale plní roli randomizace směru růstu. To má za následek pokroucení všech větví včetně těch, které nebyly nijak omezeny v rámci prostoru.

3.3 Světelný model

Světelný model musí zajišťovat odpověď na dotaz, jaké množství světla se dostane k pupenu. K tomu reprezentuje prostor jako trojrozměrnou mřížku voxelů. V každém z nich ukládá hodnotu zastínění s (při žádném stínu je nulová). Každý pupen pod sebe vrhá stín ve tvaru pyramidy, kde tato hodnota s přibývající vzdáleností od něj klesá (obrázek 3.2).



Obrázek 3.2: Světelný model. Zdroj: Pałubicki a kol., 2009 [11].

Model nastavujeme čtyřmi parametry: maximálním množstvím světla M, velikostí stínu pupenu ve vrcholu pyramidy b (tj. ve voxelu, kde se pupen vyskytuje), množstvím stínu (v procentech), které propagujeme na nižší úroveň pyramidy a samotnou výškou pyramidy. Množství dostupného světla pro pupen je vypočítáno jako:

$$max(M - s + b,\, 0),$$

kde přičítáme b z předpokladu, že pupen nevrhá stín sám na sebe.

Výše popsaný světelný model jsme oproti původnímu článku upravili, protože počet pupenů v jednom vrcholu může být odlišný podle typu fylotaxe, což by při její změně znamenalo změnu počtu pupenů ve voxelu a tedy i světelných podmínek. Navíc se nám nejeví jako logické, aby vrcholový a boční pupen jednoho

vrcholu stromové struktury (kdy oba jsou na stejné pozici, jen s odlišnou orientací) vrhaly na sebe navzájem stín. Proto u nás stín nevrhají pupeny, ale každý vrchol struktury stromu. Navíc, poněvadž v rámci voxelu nevíme nic o vzájemné poloze vrcholů, rozhodli jsme se, aby ani ty na sebe navzájem nevrhaly stín. V rámci výpočtu dostupného světla je proto hodnota b odečtena tolikrát, kolik je vrcholů ve voxelu.

Autoři původního článku (Pałubicki a kol., 2009 [11]) zvolili délku hrany voxelu tak, aby přibližně odpovídala vzdálenosti mezi vrcholy v generované struktuře. Jedná se o prakticky nejmenší délku, která má smysl, protože při růstu větve je pak pozice nového navazujícího vrcholu ve stejném nebo sousedním voxelu. To zajišťuje spojitost větví v rámci mřížky voxelů.

3.4 Simulační cyklus

Na počátku cyklu máme množinu pupenů, ze kterých mohou vyrazit nové výhony. Většinou se jedná o pupeny ve vrcholech stromové struktury, které vznikly v minulé iteraci simulace. Teoreticky můžeme použít všechny pupeny ze starších částí stromu, ale protože nebyly zvoleny pro růst v minulosti (neměly dostatek zdrojů, nebo prohrály boj o prostor), šance, že se tak stane nyní, je mizivá. Kořen stromu je reprezentován jediným pupenem.

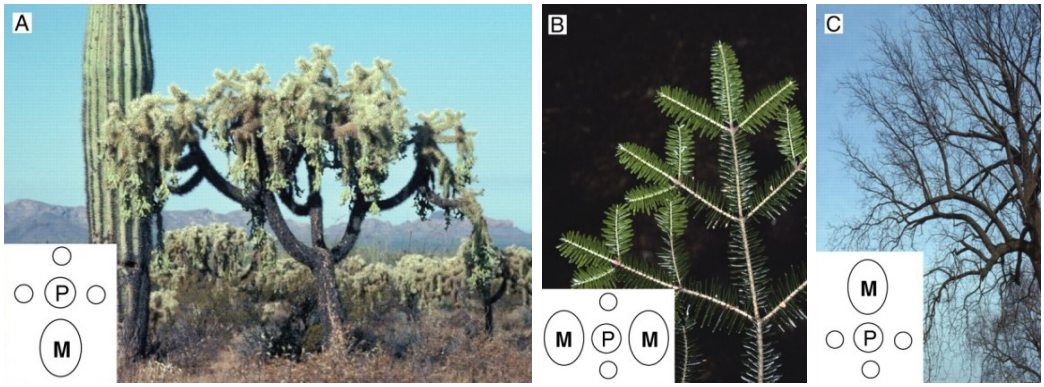
3.4.1 Inicializace zdrojů

Počáteční hodnotou zdrojů pupenu je množství světla, které se k němu dostává. To zjistíme pomocí světelného modelu (3.3), kterým je zajištěno, že pupeny v nižších, zakrytých částech stromu mají méně zdrojů a tedy menší šanci dalšího růstu.

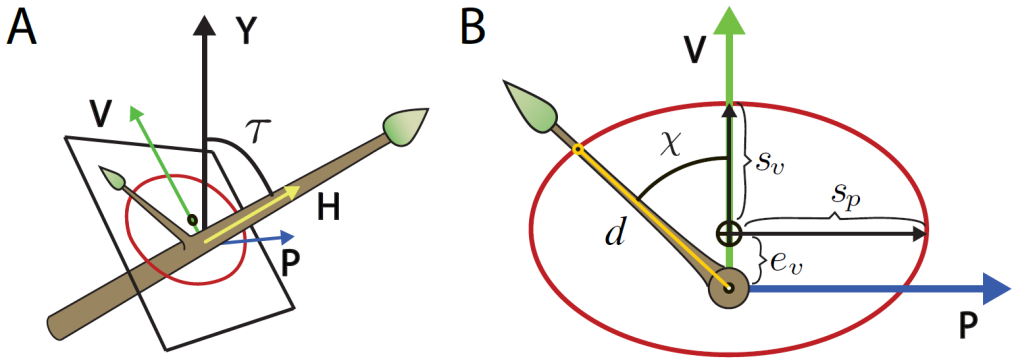
3.4.2 Gravimorfismus

Pokud je více pupenů v jednom vrcholu, pak po inicializaci mají stejnou velikost zdrojů. V přírodě ale pozorujeme jev, kdy strom může preferovat růst bočních pupenů na základě jejich orientace vůči větvi (Barthélémy a Caraglio, 2007 [14]). Rozlišujeme tři základní typy preference - pro pupeny směřující vzhůru, dolů, nebo v horizontální rovině vzhledem k ose větve (obrázek 3.3).

Pro simulaci tohoto jevu je použita rovina kolmá na směřování větve ve vrcholu, kde se pupen nachází. Na ni promítneme orientaci pupenu, čímž problém převedeme do dvourozměrné roviny. Zde můžeme analyzovat horizontální a vertikální složku směřování pupenu vůči větvi (osa větve je bodem počátku promítnutého vektoru orientace pupenu). V této rovině definujeme elipsu pomocí délek poloos v horizontální a vertikální orientaci a posunem jejího středu nahoru, nebo dolů, vůči větvi. Vzdálenost d mezi osou větve a bodem, kde se protne promítnutá orientace pupenu spolu s elipsou, udává gravimorfickou preferenci pro růst pupenu (obrázek 3.4).



Obrázek 3.3: Gravimorfismus. Zdroj: Barthélémy a Caraglio, 2007 [14]. Preference růstu pupenů na spodní straně větví se silně projevuje u některých druhů opuncií (A). Růst převážně horizontálně orientovaných pupenů je zřejmý u mnoha druhů jehličnatých stromů (B - jedle). Zvýhodnění pupenů na vrchu větví sledujeme například u ořešáku černého (C).



Obrázek 3.4: Výpočet gravimorfismu. Zdroj: Longay a kol., 2012 [12].
A) Promítnutí do kolmé roviny na směřování větve.
B) Výpočet v rovině.
Vysvětlivky: H - směřování větve, Y - vertikální směr, P - horizontální orientace vůči větvi, V - vertikální orientace vůči větvi. S_p - horizontální poloosa, S_v - vertikální poloosa, e_v - posun středu elipsy vůči větvi, d - vzdálenost mezi větví a elipsou v orientaci pupenu.

Pojmy horizontální či vertikální vůči větvi ztrácejí svůj význam, čím je směřování větve vertikálnější. Proto je výpočet rozšířen o úhel τ mezi směřováním větve a vertikální orientací. Celý vzorec pro preferenci růstu bočního pupenu p pak je:

$$p = (\cos\tau)^2 + d(\sin\tau)^2.$$

Jeho efekt je největší u horizontálních větví a nulový u vertikálních. Uplatnění hodnoty preference p pro boční pupeny je popsáno v kapitole 3.4.5. Ukázkou toho, co gravimorfismus umožňuje při souhře se zbytkem systému je obrázek 5.1.

3.4.3 Rozšířený Borchert-Hondův model

V rámci simulace se jedná o systém určený k přerozdělování zdrojů ve stromě. Umožňuje zavést preferenci růstu hlavních, nebo vedlejších větví pomocí přerozdělení zdrojů pupenů, které jim odpovídají. Je tedy simulací přírodního jevu nazývaného apikální dominance.

Algoritmus funguje ve dvou krocích. Nejprve je množství zdrojů ve všech pupenech propagováno směrem do kořene tak, že v každém vrcholu stromu je uložen součet všech zdrojů, které se nachází v stromové struktuře nad ním. Pro potřeby druhého kroku si představme vrcholové pupeny jako hlavní větve a boční pupeny jako větve vedlejší. V něm celkové množství zdrojů (nyní uložené v kořeni) rozvádíme po stromě směrem vzhůru do původních pupenů. Zdroje Z ve vrcholu rozdělujeme mezi hlavní větev (či vrcholový pupen) Z_v a vedlejší větev (případně boční pupen) Z_b podle vzorců:

$$Z_v = Z\frac{bP_v}{bP_v + (1-b)P_{bt}},$$

$$Z_b = Z\frac{(1-b)P_b}{bP_v + (1-b)P_{bt}}.$$

P_v, P_b a P_{bt} jsou hodnoty zdrojů, které byly uloženy při propagaci směrem dolů v navazujících vrcholech. Konkrétně P_v pro hlavní větev (vrcholový pupen), P_b pro jednu konkrétní vedlejší větev (boční pupen) a P_{bt} pro součet přes všechny navazující boční větve (a pupeny). Argument b z intervalu [0, 1] pak udává, zda preferujeme hlavní větve ($b > 0{,}5$), boční ($b < 0{,}5$), nebo žádné ($b = 0{,}5$).

Zatímco pomocí gravimorfismu (3.4.2) nastavujeme preferenci v rámci bočních pupenů, pomocí tohoto systému můžeme zvýhodnit pupeny vrcholové. To je naprostou nutností pro modelování jehličnatých stromů a silným nástrojem, jak pomocí jednoho parametru dosáhnout velmi rozdílných tvarů listnatých stromů (obrázek 5.2).

3.4.4 Určení délek výhonů

Po přiřazení zdrojů musíme na základě nich rozhodnout o tom, které pupeny vyraší a jak dlouhé výhony z nich vyrostou. Označme z jako zdroje pupenu, Z_{max} hodnotu největšího množství zdrojů, které má pupen v iteraci a D_{max} parametr (zadaný uživatelem), vyjadřující maximální délku výhonu. Na základě nich Longay a kol., 2012 [12] navrhují výpočet délky d pro výhon pupenu ve formě:

$$d = \left\lfloor \frac{z}{Z_{max}} D_{max} \right\rfloor.$$

Tento vzorec ale shledáváme za pochybný vzhledem k uživatelem zadanému parametru D_{max}. Při jeho celočíselné hodnotě může dosáhnout maximální délky potenciálně výhon jediného pupenu. To je problémem u malých hodnot, obzvláště v situaci $D_{max} = 1$, kdy nepředpokládáme, že by uživateli dávalo smysl, proč roste jediný pupen v celé iteraci.

Proto zachováváme základní myšlenku, ale vzorec upravujeme do podoby

$$d = \left\lfloor \frac{z}{Z_{max} + \varepsilon} (D_{max} + 1) \right\rfloor.$$

Kde je ε nejmenší taková hodnota, aby pro žádné z neplatilo $(z/(Z_{max}+\varepsilon)) = 1$. Jen s touto úpravou by ale žádný výhon nemohl mít nikdy délku D_{max}, proto také $D_{max}+1$ v upraveném vzorci. Výsledkem je tedy rozdělení rozsahu $[0, Z_{max}]$ (bez újmy na obecnosti) na stejně velké intervaly hodnot zdrojů, kterým připadnou délky výhonů v rozsahu 0 až D_{max}.

Ze způsobu, jak jsou výsledné zdroje přepočítány na délky výhonů, je vidět, že závisí na hodnotě Z_{max} a poměru z každého pupenu vůči ní. Pokud bychom dokázali posunout rozložení zdrojů pupenů po intervalu $[0, Z_{max}]$, ovlivnili bychom tím množství a délky výsledných výhonů. Proto Longay a kol., 2012 [12] zavedli parametr citlivosti pupenu na světlo c, který implementovali jako exponent, na nějž jsou všechny hodnoty zdrojů v pupenech umocněny. Při hodnotě $c = 1$ nemá žádný vliv. Čím je c větší než 1, tím kratší většina výhonů bude, nebo nevyroste vůbec. Pro c jdoucí od 1 k nule je tomu naopak. Až při $c = 0$ mají všechny pupeny zdroje rovné 1 - tedy stejný předpoklad pro růst. Aplikace tohoto parametru by ale byla nevhodná těsně před určováním délek výhonů, neboť by přebíjela veškeré vlastnosti, jež jsou pomocí velikosti zdrojů v pupenech simulovány. Více o parametru c hovoříme v následující kapitole. Projev zvýšení citlivosti pupenů na světlo ukazuje obrázek 5.3.

3.4.5 Aplikace zdrojů

Po inicializaci (3.4.1) zdrojů a před výpočtem délky výhonů (3.4.4) potřebujeme uplatnit gravimorfismus (3.4.2), Rozšířený Borchert-Hondův model (3.4.3) a parametr citlivost pupenů na světlo (3.4.4).

Nejprve provádíme výpočty v rámci jednotlivých pupenů a až poté v rámci celého stromu. Provedení v tomto pořadí je výhodné v případě vícevláknové implementace, kde tím nemícháme nezávislou práci v rámci pupenů s výpočtem vyžadujícím synchronizaci.

Označme hodnotu gravimorfismu g, citlivosti pupenu na světlo c a velikost zdrojů v pupeni po inicializaci z. Pokud se jedná o vrcholový pupen, kde gravimorfismus nemá vliv, pak $g = 1$. Novou hodnotou zdrojů pro každý pupen z_n je:

$$z_n = gz^c.$$

Poté uplatníme Rozšířený Borchert-Hondův model a nakonec vypočítáme výsledné délky výhonů. Do dalšího procesu růstu větví (3.4.7) pokračují pupeny s přiřazenou nenulovou délkou růstu.

3.4.6 Tropismus

Jak již bylo zmíněno, tropismus je změna orientace rostliny na základě vnějšího stimulu. Zatímco u bylin je velmi výrazná orientace za světelným zdrojem, u stromů je nejdůležitější jeho růst vůči směru působení gravitace, což je projev gravitropismu. Zatímco kmen stromu většinou směřuje rámcově vzhůru, orientace větví v koruně může být různá - u jehličnatých stromů v horizontální rovině a u listnatých v různé úrovni sklonu.

Orientaci tropismu nastavujeme pomocí úhlu mezi směrem vzhůru (opačný ku gravitační síle) a preferovanou orientací větví vůči němu. Proto 0° znamená

vzhůru, 90° horizontálně a 180° dolů. Pochopitelně možné jsou všechny hodnoty v rozsahu [0°, 180°].

V rámci simulace závisí výsledný tropismus na původní orientaci orgánu, ze kterého větev vyroste. V našem případě se jedná o pupen. Tropismus pro něj vypočítáme tak, že vyrotujeme vektor vzhůru o zadaný úhel ve vertikální rovině obsahující vektor orientace pupenu. V případě, že pupen směřuje přímo vzhůru, vybíráme vertikální rovinu pro rotaci náhodně.

Tropismus většinou nechceme uplatňovat při růstu kmene. Například v případě tropismu ve směru dolů by se kmen ohnul do země a simulace skončila. Na druhou stranu jeho aplikaci na kmen nechceme zakázat úplně, neboť jsou tvary stromů (obrázek 5.4), které bez ní simulace nedokáže vytvořit. Proto, kromě ovládání samotného tropismu, umožníme uživateli specificky zvolit, zda ho chce aplikovat na kmen, či nikoliv.

3.4.7 Růst nových větví

Po aplikaci zdrojů (3.4.5) máme množinu pupenů, u kterých víme, jak dlouhé výhony z nich mají vyrůst. Aplikujeme na ně prostorový model, čímž zjistíme jejich preferovaný směr růstu P vzhledem k prostoru. Spolu s orientací pupenu O a reorientací na základě tropismu T máme trojici vektorů, které udávají potenciální směry pro růst - každý na základě jiného vlivu. Výsledný směr S vypočítáme jako jejich vážený součet:

$$S = aO + bP + cT \qquad a, b, c \in \mathbb{R}.$$

Před výpočtem vektory O, P a T normalizujeme, čímž poměr $a : b : c$ přesně reprezentuje poměr zastoupení jednotlivých složek na směru růstu.

Protože P z prostorového modelu je částečně náhodným směrem, může se jeho vliv na výsledném směru růstu v případě kmene projevit jeho ohnutím. Uživatel by tento jev nemohl napravit jinak, než zvýšením zastoupení orientace pupenu na výpočtu směru, což by mělo za následek nejen přímější kmen, ale přímější všechny větve ve stromě. Tento problém řešíme zavedením parametru určujícím přímost kmene. Ten interně implementujeme jako tropismus pod úhlem 0° aplikovaný pouze na vrcholové pupeny kmene T_k. Výsledný směr růstu kmene S_k je tedy dán vzorcem:

$$S_k = aO + bP + cT + dT_k \qquad a, b, c, d \in \mathbb{R}.$$

Zde platí to samé jako u výpočtu pro větve, tedy také T_k je normalizované a $a : b : c : d$ odpovídá poměrům zastoupení jednotlivých složek.

Pokud má výhon délku větší než 1, musíme celý proces určení směru růstu opakovat po celé jeho délce, kde kromě počátečního pupenu je zbytek růstu uskutečněn za pomoci vrcholových pupenů v nově vznikajících vrcholech výhonu. Růst musí probíhat paralelně v celém stromě, čímž je na základě aplikace prostorového modelu zajištěno vhodné uspořádání nových větví.

Alternativně je možné vypočítat preferovaný směr v prostoru pouze pro počáteční pupen a aplikovat ho v průběhu celého růstu výhonu. To má za následek přímější větve a ušetření výpočetních prostředků nutných pro simulaci za cenu potenciálního snížení kvality uspořádání větví v koruně.

3.4.8 Opadávání větví

Opadávání větví je přirozený proces v životě stromu se silným vlivem na organizaci jeho koruny. Určení, které větve by měli opadnout, můžeme uskutečnit porovnáním celkového množství světla dopadajícího na větev s její velikostí. Takový postup je odůvodnitelný představou, že po délce větve jsou listy a čím více světla na ně dopadá, tím lepší jsou předpoklady pro fotosyntézu, čímž je větev pro strom výhodnější. V simulaci je větev reprezentována množinou pospojovaných vrcholů. Porovnáváme tedy součet světla v nich (získaného ze světelného modelu) a jejich počet. Pokud poměr mezi nimi klesne pod uživatelem zadaný práh, větev opadne, její vrcholy jsou odstraněny z modelu prostoru a světla a v simulaci nadále nijak nefiguruje. Opadávání větví provádíme v každé iteraci simulace po růstu nových větví (3.4.7).

My tento systém konkrétně implementujeme tak, že jako větev považujeme celou část koruny, která leží nad vrcholem ve stromové struktuře. Proto při testování na opad větví začínáme od těch nejmenších, na nichž nejsou žádné další navazující boční větve. Množství světla a velikost větví propagujeme ve stromě dolů se snahou uskutečnit opadnutí větve co nejníže ve stromě. Tento mechanismus již při nízkých hodnotách prahu dokáže zbavit model krátkých větviček v centru koruny, což v důsledku zvyšuje jeho realističnost.

Opadávání větví má problém vyplývající z jeho silné závislosti na světelném modelu. V něm je stín propagován jen do omezené vzdálenosti pod vrcholem větve, což znamená, že pokud nastavíme takový práh, aby opadly všechny větve u země, budou ve skutečnosti opadávat ty, které jsou ve výšce nižší než vrchol stromu mínus vzdálenost propagace ve světelném modelu. Proto narozdíl od Pałubicki a kol., 2009 [11] nepovažujeme použití modelu opadávání větví za tímto účelem za vhodné. Namísto toho zavádíme jednoduchý parametr délky kmene bez větví, jehož aplikace a dopad jsou naprosto zřejmé. Opadávání větví ponecháváme jako případný prostředek, pomocí něhož lze při nízkých hodnotách prahu odstranit menší větve uvnitř koruny (viz obrázek 5.3).

3.5 Interakce se scénou

Jedním z cílů, který jsme si vytyčili, byla interakce generátoru se scénou zadanou ve formě trojúhelníkové sítě. Použití světelného modelu, jenž reprezentuje prostor ve formě mřížky voxelů, jsme zavrhli, neboť pro větší scény by se nám mřížka reprezentující celou scénu nemusela vejít do paměti.

K zajištění veškeré interakce jsme použili metodu vrhání paprsku, pomocí níž jsme testovali viditelnost mezi dvěma body v prostoru. Využili jsme ji ve všech částech simulačního procesu, které by mohly být případnou překážkou pro růst stromu ovlivněny.

1. Po vypočítání výsledného směru růstu pupenu a před jeho uskutečněním testujeme, zda se mezi pozicí pupenu a cílovou pozicí růstu nevyskytuje překážka. Pokud ano, pak růst neuskutečníme nehledě na jakékoliv další okolnosti.

2. Při vytváření značek prostoru v oblasti vnímání před pupenem, testujeme vzájemnou viditelnost mezi pupenem a značkou. Značky zakryté překáž-

kou pak do prostorového modelu nevkládáme. Je tedy možné, že pupen nevygeneruje žádnou validní značku prostoru.

3. Při přiřazování značky prostoru konkrétnímu pupenu ověřujeme jejich vzájemnou viditelnost. Tím předcházíme tomu, že značka vytvořená pupenem bude přiřazena jinému, z jehož pohledu je schovaná za překážkou.

Bod 1. dostačuje k tomu, aby generované větve stromu neprotnuly objekt ve scéně. Sám o sobě ale nijak nesimuluje snahu větví překážku obrůst. K tomu slouží 2. a 3. bod. Představa je, že značky prostoru jsou z pohledu pupenu jen v dostupných místech scény, což zajišťuje, že doporučený směr získaný z prostorového modelu by neměl kolidovat s překážkou ve scéně. Pokud je ve scéně například plocha s mnoha malými štěrbinami, může se stát, že i výsledný doporučený směr s ní koliduje. To nevadí, protože na orientaci rostoucí větve se částečně podílí orientace pupenu a tropismus, tedy samotné body 2 a 3 by kolizi se scénou stejně předejít úplně nedokázaly.

3.6 Tvorba meshe

Před vytvořením trojúhelníkových meshů větví z vygenerované stromové struktury musíme určit jejich poloměry. Ty v rámci stromu definujeme pomocí tzv. trubicového modelu, který udává vztah mezi poloměry větví $r_1, \ldots, r_n$ nad bodem větvení a poloměrem větve r pod ním rovnicí:

$$r^n = r_1^n + r_2^n + \cdots + r_n^n,$$

kde n je parametr trubicového modelu. Čím je n větší, tím menší je poloměr větve pod bodem větvení. Díky němu je zajištěn přirozený nárůst poloměru od vrchních větví v koruně až po kmen.

Výpočet poloměrů začíná od špiček větví směrem do kořene. Všechny špičky mají stejný poloměr, který může uživatel nastavit. Pomocí něj můžeme manipulovat s obecnou tloušťkou větví. Aby byly tloušťky korektní, vzhledem k průběhu simulace růstu stromu, je zapotřebí do výpočtu zahrnout i opadnuté větve. Samotné větve modelujeme pomocí generalizovaných válců (Bloomenthal, 1985 [15]), kterými obalujeme stromovou strukturu ze simulačního algoritmu, kde kvůli lepšímu vizuálnímu dojmu z tvarů větví provádíme interpolaci vrcholů každé větve pomocí vhodné křivky. My jsme pro interpolaci zvolili přirozený kubický spline a to z několika důvodů. Výsledná křivka prochází všemi interpolovanými body a jejich souřadnice postačují pro její výpočet. Také má spojitou druhou derivaci v koncových bodech rovnou nule, což znamená, že má spojitou změnu zakřivení po celé její délce, čímž působí opticky velmi přirozeně. Vychýlení vrcholu větve vůči okolním se na kubické spline projevuje jen lokálně, díky čemuž poměrně dobře kopíruje původní stromovou strukturu, která je ekvivalentní lineární interpolaci. To je důležitou vlastností u větví blízko překážek ve scéně, kde tím minimalizujeme možnost jejich průniku s interpolační křivkou.

Mesh větve je reprezentován kružnicemi se středy v interpolační křivce. Jejich orientace je dána tečnou (první derivací) kubické spline v daném bodě. Kružnice jsou reprezentovány nastavitelným počtem vrcholů rozmístěných rovnoměrně po jejich obvodu. Spojením vrcholů po sobě následujících kruhů vytváříme válcovitý

segment reprezentující část větve. Kružnice jsou vždy v interpolovaných bodech, ale jejich počet mezi dvojicí vrcholů stromové struktury můžeme navýšit. Aby bylo zabráněno kroucení válců (vznikající nevhodnou volbou vrcholů vytvářejících trojúhelník meshe na po sobě následujících kruzích), nejsou kružnice vrcholů generovány samostatně, ale od počátku větve vždy vrcholy předchozí kružnice vyrotujeme do orientace kružnice následující. Po délce větve pak hranu trojúhelníku tvoří reprezentace stejného vyrotovaného vrcholu. Aplikovaná rotace na kružnici je totožná rotaci nutné pro přeorientování tečny v počátečním bodě do tečny v bodě cílovém.

Tento postup nebere nijak v potaz vzdálenost mezi vrcholy po sobě jdoucích kružnic ani jejich poloměr. To má za následek, že na vnější straně ohybu větve jsou vrcholy kružnic dále od sebe, než na straně vnitřní. Pokud je ohyb příliš prudký, nebo nárůst poloměru v jeho průběhu velký, pak se kružnice na jeho vnitřní straně dokonce mohou překrývat. Tento jev na námi generovaných modelech není tak častý a zjevný, ale před případnou distribucí aplikace bude vhodné ho odstranit, buď pomocí redukce geometrie v problematických místech, nebo volbou odlišné metody pro tvorbu trojúhelníkového meshe.

Pro lepší optický dojem nepoužíváme větve tak, jak je vytváří generátor, tedy větev není segment od počátku (bodu křížení nebo kořenu) do posledního vrcholu (kde jeho vrcholový pupen nevyrostl). Za větev pro účely tvorby generalizovaných válců považujeme cestu ve stromové struktuře, kde dochází k nejmenší změně obvodu. To je důležité, protože válcové reprezentace jednotlivých větví se v bodech větvení překrývají, ale nová definice větví pro tvorbu meshe zajišťuje to, že užší větev vždy vrůstá do širší a nikdy naopak.

UV mapování provádíme na základě rozměrů textury a jejího škálování po délce, které může uživatel nastavit. Souřadnice textury na obvodu větve odvozujeme z jejího počátku, kde je její poloměr největší.

3.7 Listy

Pro realistickou implementaci listů je nutné určit pro ně vhodné pozice a orientace. Jednoduchým a přirozeným postupem je jejich umístění na všechny větve slabší než uživatelem zadaný průměr. V takovém případě by listy nebyly umístěny jen ve vrcholech vygenerované stromové struktury, ale nezávisle na ní v zadané hustotě po povrchu větví.

Konkrétní pozice a počáteční orientace řapíku by měla být dána fylotaktickým uspořádáním. Strom může řapík využít k tomu, aby změnil orientaci listu - například, aby ho nastavil plochou směrem ke světlu. Různé druhy stromů mohou orientovat listy odlišným způsobem, což se projevuje na výsledném vizuálním dojmu. Proto by pro realistické rozmisťování listů bylo zapotřebí vytvořit systém popisující jejich možné reorientace.

Implementaci listů jsme neuskutečnili v rámci této práce, ale byla by jejím nejvhodnějším pokračováním.

3.8 Pseudokód algoritmu

```
/* Předpokládáme, že scéna je nahrána, stromy přidány
   a požadované parametry nastaveny.                          */
```

Input: Počet iterací generativního algoritmu It_{cnt}.
Result: Trojúhelníkové meshe vygenerovaných stromů.
Data: P = množina aktivních pupenů, ze kterých mohou růst nové výhony. S = množina stromů.

foreach *strom* $s \in S$ **do**
  if s *je nový* **then** přidej pupen kořene do P

for $i \leftarrow 1$ **to** It_{cnt} **do**
  foreach *pupen* $p \in P$ **do**
    // $Z_p =$ zdroje pupenu p
    $Z_p \leftarrow$ dostupné světlo pupenu (3.4.1)
    $g \leftarrow$ gravimorfická preference pupenu (3.4.2)
    $c \leftarrow$ citlivost pupenu na světlo (3.4.4)
    $Z_p \leftarrow gZ_p^c$ (3.4.5)
  end
  foreach *strom* $s \in S$ **do**
    aplikuj Rozšířený Borchert-Hondův model (3.4.3)
    foreach *pupen* $p \in s$ **do** /* Každý pupen ve stromě */
      // $D_p =$ délka výhonu, který má vyrůst z pupenu p
      $D_p \leftarrow$ délka výhonu na základě Z_p (3.4.4)
    end
  end
  odstraň všechny $p \in P$, kde $D_p = 0$
  while $P \neq \emptyset$ **do**
    $P \leftarrow$ běh prostorového modelu na P (3.2)
    foreach *pupen* $p \in P$ **do**
      spočítej směr působení tropismu (3.4.6)
      spočítej výsledný směr růstu (3.4.7)
      $v \leftarrow$ nový vrchol stromu ve směru růstu, jeho vrcholový pupen v_p dostává přiřazenu délku výhonu: $D_{vp} \leftarrow D_p - 1$
      odstraň p z P
      if $D_{vp} \neq 0$ **then** přidej v_p do P
    end
  end
  foreach *strom* $s \in S$ **do** aplikuj opadávání větví (3.4.8)
  validní pupeny na neopadnutých vrcholech stromů vzniklých v této iteraci přidej do P
end
foreach *strom* $s \in S$ **do**
  vytvoř trojúhelníkový mesh větví (3.6)
end

4. Implementace

V rámci této kapitoly představujeme použité externí knihovny, moduly a technologie (4.1). Náležitosti pro kompilaci kódu naší aplikace naleznete v kapitole 4.2. Strukturu generátoru stromů, který je implementací výše popsaného generativního algoritmu a další, s ním spjaté zajímavosti, jsou obsahem kapitoly 4.3. Stejně tak je prezentováno uživatelské rozhraní aplikace(4.4).

K implementaci jsme zvolili jazyk C++, jenž je vhodný pro výpočetně náročné aplikace, jako právě náš procedurální generátor stromů. Vývoj probíhal v prostředí Visual Studia 2017. V něm je aplikace rozdělena na dva projekty:

1. Generator - generátor stromů ve formě statické nebo dynamické knihovny.
2. GUI - uživatelské prostředí, jako spustitelná aplikace.

4.1 Použité knihovny, moduly a technologie

V obou projektech byla použita:

- OpenGL Mathematics (GLM) - verze 0.9.9.0 [16]
  - MIT licence.
  - Header-only knihovna pro matematické výpočty se syntaxí podobnou GLSL (OpenGL Shading Language).

V rámci projektu Generator byly navíc použity:

- Cubic spline interpolation C++ [17]
  - GNU GPL 2 nebo 3.
  - Header s implementací kubické spline interpolace na základě LU rozkladu pásové matice.
  - Jiné knihovny nabízí pod stejnou licencí daleko více funkcí, které bychom nevyužili.
  - Provedli jsme na ní optimalizace pro naše konkrétní použití (interpolace vrcholů větví ve stejné vzdálenosti od sebe).
- Intel Embree 3.2.0 [18]
  - Apache 2.0 licence.
  - Vysoce výkonná knihovna pro vrhání či sledování paprsků.
- Intel Threading Building Blocks 2018 Update 4 [19]
  - Apache 2.0 licence.
  - Template knihovna pro task paralelismus.
  - Využívaná v rámci Embree a tedy přirozenou volbou pro implementaci paralelismu v generátoru samotném.

V rámci projektu GUI byly navíc použity:

- TinyObjLoader [20]
  - MIT licence.
  - Knihovna ve formě header souboru určená k nahrávání .obj souborů.
- Qt 5.10.1 [21]
  - GNU LGPL 3.0.
  - Multiplatformní framework zprostředkovávající tvorbu grafického uživatelského rozhraní.
  - Jedná se o jeden z nejznámějších a komerčně nejvyužívanějších C++ frameworků tohoto druhu. Pro naši aplikaci se nabízely i další možnosti, ale Qt nemá z našeho pohledu, kromě nutnosti distribuovat více dll knihoven, žádné nevýhody. Navíc, kromě klasického řešení událostí (pohyb myši v okně, stisk klávesy apod.) pomocí zpětných volání, využívá systém signálů a slotů pro komunikaci v rámci aplikace. Implementace naší aplikace tedy byla dobrou příležitostí vyzkoušet si specifika Qt v praxi.
- OpenGL (3.3 Core) [22]
  - Multiplatformní API pro vykreslování 2D a 3D grafiky.
  - Qt jakožto multiplatformní framework podporuje použití OpenGL.
  - Core verze 3.3 byla zvolena, protože s ní již máme zkušenosti a zakazuje použití zastaralých součástí OpenGL (fixed-pipeline apod.), které není vhodné používat.

4.2 Kompilace kódu

Jak již bylo zmíněno, aplikace se skládá ze dvou projektů Visual Studia 2017. Z tohoto důvodu je cílovou platformou 64 bitový Windows 10. V praxi by nebyl problém překládat kód pro 32 bitové systémy, nebo jiné platformy jako Unix, protože všechny použité knihovny jsou multiplatformní.

Solution nabízí konfigurace pro použití knihovny generátoru jak ve statické, tak v dynamické podobě. V případě použití projektu Generator ve formě dynamické knihovny je vyžadováno definování makra TREEGEN_AS_DLL (u nás je nastaveno). Varujeme, že knihovna je spíše statického rázu, neboť komunikaci s ní provádíme za pomoci komplexních datových typů a vyhazujeme z ní výjimky. Proto dbejte na to, aby byla knihovna, nehledě na formu (statická/dynamická) kompilována se stejnou CRT (C Run-time library) a nastavením vlastností šíření výjimek (kompilační direktivy) jako zbytek vaší aplikace.

Projekt *Generator* je samostatnou knihovnou generátoru stromů. Všechny potřebné knihovny, nutné pro jeho překlad a linkování, jsou dodány spolu s ním.

Projekt *GUI* nelze kompilovat bez nainstalování Qt frameworku. Zapotřebí nejsou jen rozsáhlé knihovny Qt, ale i speciální kompilátory zprostředkovávající překlad souborů popisujících uživatelské rozhraní, napojující Qt signály a sloty

a zařizující další nezbytnou funkcionalitu Qt. Pro nainstalování Qt a jeho nastavení v rámci Visual Studia 2017 je nutné vykonat tyto kroky:

1. Z webu Qt [21] stáhnout instalátor.
2. V rámci něj přidat variantu Qt 5.10.1 - MSVC 2017 64-Bit. (Velikost celé instalace dosahuje skoro 4 GB).
3. Nainstalovat Qt VS Tools [23].
4. Spustit Visual studio a v QT VS Tools přidat cestu k Qt 5.10.1.
5. Projekt *GUI* je odteď přeložitelný a spustitelný (korektní volání překladačů Qt v kompilačním procesu zařídí QT VS Tools).

4.3 Generátor stromů

Celý generátor je silně modulární, kde každý modul je prvkem simulačního cyklu, nebo reprezentací části prostředí. Jako rozhraní pro vnější komunikaci slouží abstraktní třídy *Generator* a *Tree* a výstupní struktury obsahující data, jako je mesh stromu nebo jeho struktura.

Zamýšlený způsob použití generátoru je ryze sekvenčního rázu:

1. Nahrání parametrů.
2. Běh generativního algoritmu.
3. Vytvoření meshe.

S použitím generátoru přes jeho externí rozhraní seznamují následující dvě podkapitoly. Ve zbytku kapitoly se zaměřujeme na jeho interní fungování.

4.3.1 Generator

Třída *Generator* spravuje vytváření či rušení stromů, nastavení vlastností prostředí a běh simulačního procesu.

Generátor umožňuje generovat více stromů současně v jedné scéně. Manipulace s nimi (jejich přidání nebo smazání) je bezpečná jen ve chvíli, kdy simulace neběží. To znamená, že můžeme na jednom stromě provést několik iterací simulace, poté přidat další stromy do scény a pokračovat se všemi v iteracích následujících.

Vlastnosti prostředí je možné změnit jen předtím, než proběhne první iterace simulace. Stromy je možné navrátit do stavu před simulací pomocí metody *clearCurrentModels*, po jejímž použití můžeme prostředí znovu nastavovat. Výjimkou z tohoto pravidla je škálování modelu, které můžeme měnit bez ohledu na proběhlé iterace.

Simulace je spouštěna metodou *runIterations* a může být kdykoliv v jejím průběhu zrušena pomocí *cancelIterations*. Na stav generátoru je možné se kdykoliv dotázat pomocí trojice metod *isRunning*, *isCanceled* a *isEnvironmentLocked*. Paralelní volání těchto pěti metod je synchronizováno.

Nahrání scény ve formě trojúhelníkového meshe (metoda *loadScene*) nebo prázdné scény (metoda *defaultEmptyScene*) vymaže všechny stromy z generátoru. Protože reprezentace prostředí uvnitř generátoru je, co se velikosti týče, limitována (viz 4.3.4), je umístěna rovnoměrně kolem pozice prvního přidaného stromu do generátoru. Rozsah prostoru ve scéně, který generátor reprezentuje v danou chvíli, lze zjistit pomocí metody *boundingBoxBounds*.

Pomocná funkce *tryGetRootPosition* provádí vrhání paprsku ze souřadnic ve scéně v daném směru a analyzuje pozici, kde se protne se scénou. Uplatníme ji ve chvíli, kdy nemáme v aplikaci vlastní reprezentaci scény a chceme určit pozici pro růst stromu - například na základě souřadnic ukazatele myši ve vykreslovacím okně. V případě, že vržený paprsek mine scénu, je funkcí vrácena pozice odpovídající jeho protnutí s rovinou na úrovni nejnižšího bodu scény. Pokud dopadá paprsek na plochu, která má sklon větší než 90°, přidává doporučení, že na dané pozici nemá smysl strom umisťovat, neboť simulace není schopna korektně realizovat růst směrem dolů.

Kromě výše zmíněných, vzájemně synchronizovaných metod, je možné kdykoliv volat všechny konstantní metody za předpokladu, že ve stejnou dobu neběží jiná (nekonstantní) metoda, kromě běhu simulace.

4.3.2 Tree

Třída *Tree* zprostředkovává nastavení vlastností jednotlivých stromů nezávisle na sobě a metody pro načtení struktury stromu (*getTreeStructure*, *getShedBranches*) nebo vytvoření meshe stromu (*makeTreeMesh*).

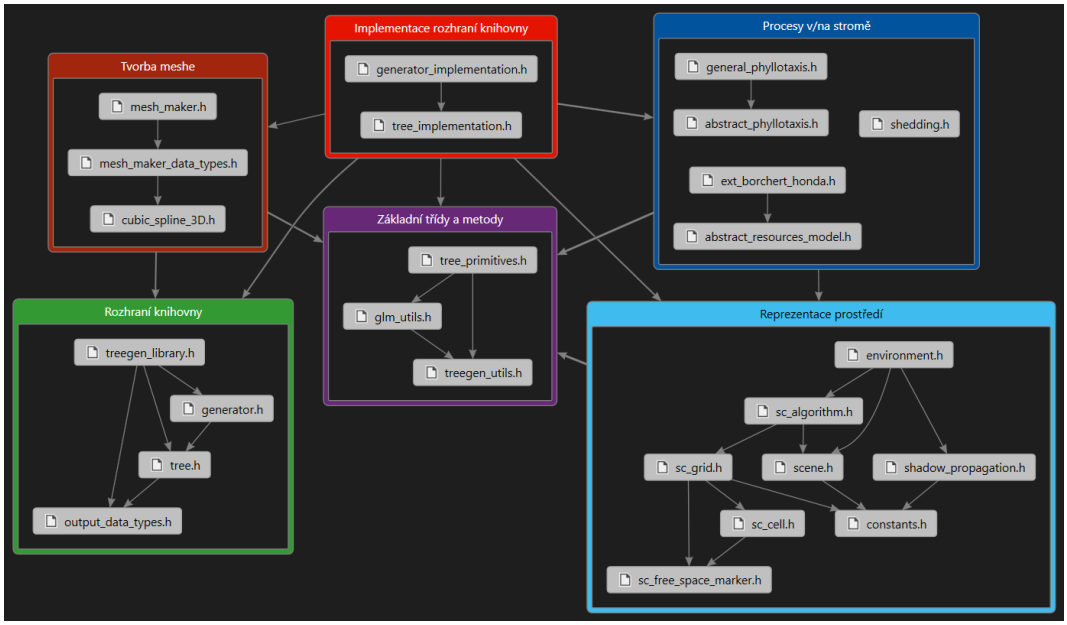
Tvorba meshe u větších modelů může být časově náročnější, a proto ji lze zrušit pomocí *cancelMakeMesh*. Ostatní metody této třídy nejsou nijak synchronizovány. Konstantní funkce lze volat kdykoliv. Všechny nekonstantní metody mají nedefinované chování při jejich paralelním volání nebo při jejich volání v průběhu simulace (v takovém případě by mohlo dojít k pádu programu).

4.3.3 Vnitřní struktura generátoru

Jádrem generátoru je implementace abstraktního rozhraní knihovny ve formě tříd *GeneratorImplementation* a *TreeImplementation*. Ty spolu obstarávají korektní běh simulace.

GeneratorImplementation je správcem generativního procesu popsaného v kapitole 3. Volá metody na jednotlivých stromech, které o sobě vzájemně nevědí tak, aby stejný krok simulace byl proveden na všech stromech, než je přistoupeno k dalšímu. Navíc, aby boj o prostor mezi pupeny dával korektní výsledky, je nutné ho realizovat s pupeny všech stromů zároveň, což lze snadno zařídit právě s objektem organizujícím celý simulační cyklus.

TreeImplementation zprostředkovává volání jednotlivých kroků simulace, jež nevyžadují přímou vzájemnou interakci mezi stromy. Konkrétně se jedná například o metody reprezentující: odvození pupenů pro růst na základě dostupného světla, nahrání pupenů do prostorového modelu, růst jednoho vrcholu výhonů, nebo opadávání větví stromu. V rámci implementace těchto metod jsou volány další moduly reprezentující jednotlivé operace nutné pro uskutečnění simulace (uplatnění fylotaxe, Rozšířený Borchert-Hondův model atp.).



Obrázek 4.1: Struktura generátoru.

Struktura celého systému je dobře vidět ze zjednodušeného diagramu vzájemného zahrnutí header souborů (obrázek 4.1). Na něm jednotlivé soubory obsahují:

- Rozhraní knihovny a jeho implementace
  - Výše, detailně popsané stejnojmenné třídy.
  - Je vidět, že vnější rozhraní neumožňuje únik žádným datovým typům používaným uvnitř knihovny.
- Základní třídy a metody
  - tree_primitives.h - třídy pro vrcholy stromu, pupeny a větve. Ty jsou vzájemně provázány ukazateli, takže případný pohyb po stromové struktuře je možné uskutečnit odkudkoliv.
  - glm_utils.h a treegen_utils.h - pomocné metody využívané v různých částech generátoru.
- Tvorba meshe
  - cubic_spline_3d.h - upravená knihovna kubické interpolace pro naše konkrétní použití.
  - mesh_maker_data_types.h - datové typy použité pro reprezentaci stromu uvnitř třídy pro tvorbu jeho meshe.
  - mesh_maker.h - třída pro tvorbu meshe (každý strom má vlastní instanci).
- Reprezentace prostředí
  - constants_h - konstanty pro nastavení velikosti modelů prostředí (viz kapitola 7.3.4).

  - scene_h - reprezentace scény, zprostředkovává vrhání paprsků přes Embree.
  - sc_algorithm.h - algoritmus prostorového modelu.
  - shadow_propagation.h - světelný model.
  - environment.h - zastřešení všech součástí prostředí pod jedinou třídu. Ta existuje v rámci simulace v jediné instanci, ke které mají přístup všechny stromy.

- Procesy v/na stromě
  - abstract_phyllotaxis.h - abstraktní třída pro potenciálně libovolný druh fylotaxe.
  - general_phyllotaxis.h - konkrétní implementace fylotaxe umožňující nastavení libovolných úhlů a střídavého nebo protilehlého typu.
  - abstract_resource_model.h - abstraktní třída pro libovolný typ přerozdělení zdrojů v rámci stromu.
  - ext_borchert_honda.h - implementace Rozšířeného Borchert-Hondova modelu.
  - shedding_h. - implementace opadávání větví.

4.3.4 Konstanty

Vzhledem k tomu, že světelný model je ve formě trojrozměrné mřížky voxelů, zabíral by při větších velikostech neúnosně mnoho paměti. Proto jsou rozměry mřížky omezeny tak, aby plně postačovala pro generování osamoceného stromu. Světelný model nepodporuje růst větví směrem dolů (oblasti pod větvemi jsou zastíněny). Proto je pozice prvního přidaného stromu mapována doprostřed nejnižší úrovně mřížky (tím je maximalizován prostor, který ve světelném modelu může strom vyplnit). I když knihovna umožňuje generování více stromů zároveň, může se stát, že velikost mřížky bude omezením pro jejich vygenerování (obzvláště při jejich umístění k okrajům mřížky).

Je přirozené, že prostorový model je pak reprezentován stejně - tedy trojrozměrnou mřížkou buněk totožné velikosti jako ve světelném modelu. Vzhledem k typu operací, které je při jeho běhu nutné vykonávat (vyhledávání nejbližších pupenů pro prostorové body), je mřížka poměrně vhodnou datovou strukturou. Při vhodně zvolené velikosti jejích buněk (menší, nebo rovné velikosti vzdálenosti vnímání pupenu) stačí v průběhu výpočtu analyzovat buňky, jež v mřížce sousedí.

4.3.5 Vícevláknový běh simulace

Při převedení simulace do podoby vícevláknové implementace bylo zapotřebí zachovat reproduktibilitu generovaných modelů na základně totožných nastavení parametrů pro strom a náhodných generátorů.

Použití globálních náhodných generátorů se synchronizací přístupu není možné, protože je zapotřebí, aby náhodná čísla v simulaci byla deterministická vzhledem k semínku pro strom. Ukázalo se, že veškerou náhodnost v generátoru lze převést do podoby, kde stačí, aby každý pupen byl schopen generovat náhodná

čísla. Proto má každý pupen vlastní náhodný generátor. Pupen reprezentující kořen je inicializován semínkem stromu. Ve chvíli, kdy je vytvořen nový vrchol stromu, jsou v něm generátory pupenů nastaveny pomocí generátoru pupenu, ze kterého vrchol vyrostl.

Druhým problémem se ukázalo být zaokrouhlování reálných čísel v hardwaru. V závislosti na pořadí operandů se totiž může výsledek lišit. Simulační proces je velice náchylný na libovolné změny - například nepatrné vychýlení větve může změnit voxel ve světelném modelu nebo pozice, kde budou vytvořeny značky volného prostoru v prostorovém modelu. Řešili jsme dva problémy tohoto typu:

1. Přiřazené značky volného prostoru pupenu mohou být v libovolném pořadí. Výsledný preferovaný směr se odvíjí od součtu směrových vektorů, kde se zaokrouhlovací chyby projevují ve třech reálných číslech zároveň.

2. Voxel světelného modelu je reálné číslo reprezentující velikost stínu. Jakákoliv jeho úprava je tedy náchylná na zaokrouhlovací chyby.

Oba problémy je možné vyřešit zavedením deterministického pořadí operací. Značky volného prostoru třídíme před výpočtem na základě unikátního identifikačního čísla, které jim přidělujeme. Druhý problém považujeme za neřešitelný, pokud vyžadujeme vícevláknový přístup, při použití reálných čísel. Proto úpravy hodnot v něm provádíme sekvenčně v deterministickém pořadí. Pokud bychom našli mapování mezi celými čísly a původním modelem založeným na reálných parametrech, pak bychom se úplně zaokrouhlovacímu problému vyhnuli. Protože úpravy hodnot ve světelnému modelu nejsou úzkým hrdlem simulace, na výsledném výkonu se tento problém projevil jen minimálně.

Paralelizovali jsme všechny části simulace, které přirozeně paralelizovat šly. Hlavně se jedná o: prostorový model, růst nových výhonů, lokální výpočty na pupenech a tvorbu meshe jednotlivých větví stromu. Naopak se nám nepodařilo paralelizovat Rozšířený Borchert-Hondův model a opadávání větví, kde cena za synchronizaci byla příliš vysoká.

4.4 Uživatelské rozhraní

Uživatelské rozhraní je vystavěno nad knihovnou generátoru a zprostředkovává její veškerou funkcionalitu. Simulační proces je z něj spouštěn asynchronně a po jeho dokončení, včetně nahrání modelů do bufferů OpenGL, zobrazí nové stromy ve vykreslovací části okna. Tím je umožněno prohlížet předchozí model do poslední možné chvíle, kdy je přepnuto na vykreslování nového - což činí porovnávání rozdílů mezi stromy při změně parametrů snazší. Ostatní volání knihovny jsou blokující, ale na uživatelském dojmu se to neprojevuje.

Parametry generátoru jsou přenositelné ve formě XML souboru. Jeho načítání a ukládání probíhá prostřednictvím DOM (Document object model) stromu. Pro načítání scény jsme umožnili použití .obj souborů obsahujících normálové vektory vrcholů. Ty je možné vypočítat z trojúhelníků meshe, ale při exportu scény z libovolného modelovacího nástroje by neměl být problém si je vyžádat. Proto jsme to nepovažovali v rámci této implementace za důležité. Do budoucna bude muset být umožněno použití dalších formátů, kde při jejich implementaci může být tento problém adresován.

4.4.1 Struktura projektu

Projekt uživatelského rozhraní je rozdělen do dvou hlavních tříd: TreegenApp a TreeRenderGLWidget a pomocné třídy pro ovládání kamery.

TreegenApp zajišťuje veškerou interakci uživatele s oknem aplikace, tedy události vzniklé klikáním na jednotlivá tlačítka. Mezi ně hlavně patří: přidávání a mazání stromů, asynchronní spouštění a rušení generačního procesu, ukládání a načítání parametrů, ukládání modelů stromů a načítání scény.

TreeRenderGLWidget představuje okno pro vykreslování OpenGL. Spravuje všechny úkony a datové struktury nutné pro použití OpenGL, takže nikde jinde v rámci projektu uživatelského rozhraní se s nimi nesetkáme. Protože používáme OpenGL 3.3 Core, potřebné texty shaderů jsou v projektu ve složce *shaders*. Jejich obsahem je Blinn-Phongův světelný model.

4.4.2 Z-buffer pro větší vzdálenosti

Z-buffer mapuje polovinu svého rozsahu do dvojnásobku vzdálenosti blízké plochy, což limituje jeho rozsah pro větší vzdálenosti. Následkem toho může být jev nazývaný Z-fighting, kdy fragmenty v rozdílné vzdálenosti připadnou na stejnou hodnotu Z-bufferu - takže to, který je nakonec vykreslen na výstupu, není možné předvídat. Aby tento jev nenastal, je ideální zvolit vzdálenou plochu co nejblíže ploše blízké. To by v našem případě omezovalo viditelnost ve scénách s většími vzdálenostmi, a proto byla použita alternativa z článku Maximizing Depth Buffer Range and Precision [24], která tomuto problému předchází. Její myšlenka spočívá v použití vhodnějšího mapování rozmezí mezi blízkou a vzdálenou plochou na rozsah hodnot Z-bufferu. Díky ní jsme napevno nastavili vzdálenosti blízké a vzdálené plochy na 0,001 a 1 000 000, což jsou normálně nemyslitelné hodnoty, které považujeme za dostatečné pro prezentovanou implementaci.

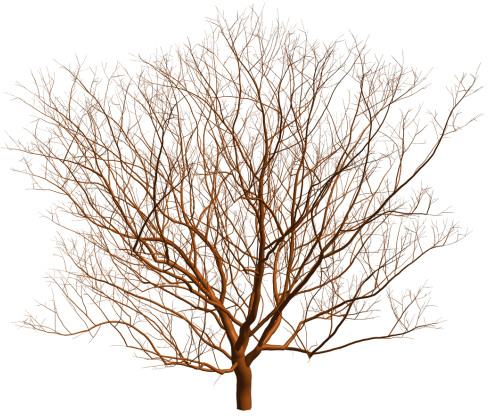
5. Generované modely

Tato kapitola je galerií různých modelů stromů, které je naše implementace schopna generovat. Na obrázcích 5.1 až 5.3 je možné pozorovat změny výsledných modelů v závislosti na změnách některých parametrů. Obrázek 5.4 prezentuje stromovité struktury vzniklé aplikací tropismu na kmen. Obrázky 5.5 až 5.7 demonstrují interakci stromu se scénou a vzájemné působení více stromů rostoucích zároveň. Na obrázku 5.8 je zobrazen postupný růst stromu v pravidelném počtu iterací algoritmu. Na obrázcích 5.9 až 5.11 jsou znázorněny některé z různých druhů reálných stromů, které se nám podařilo napodobit. Pro ně na konci kapitoly uvádíme tabulky s jejich vlastnostmi (5.1) a naměřenou dobou nutnou k jejich vygenerování (5.2).

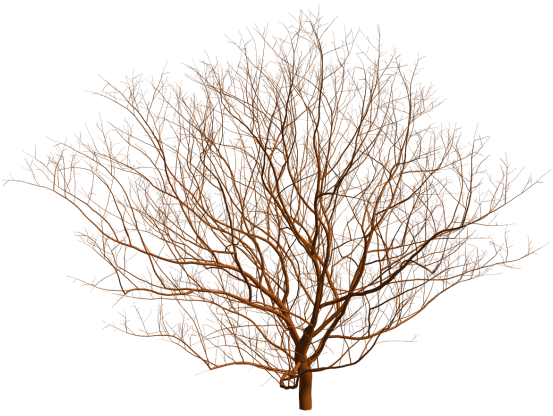
Je zapotřebí zdůraznit, že všechny modely samostatných stromů prezentovaných v této kapitole byly vytvořeny nad stejným semínkem náhodného generátoru. Proto je možné při detailním zkoumáním naleznout mezi některými z nich podobné rysy, ale prokazujeme tím velkou variabilitu modelů v závislosti na parametrech stromu, nikoliv náhodných generátorech.

Specifikace rodového názvu stromů na obrázcích 5.9 až 5.11 je nutné brát s nadhledem. Stromy stejného rodu se mohou velmi lišit nejen v rámci jednotlivých druhů, ale i v závislosti na jejich stáří. Obzvláště projevy stáří nejsme schopni tak dobře reprodukovat, protože generativní proces nesimuluje ohýbání větví pod jejich vahou.

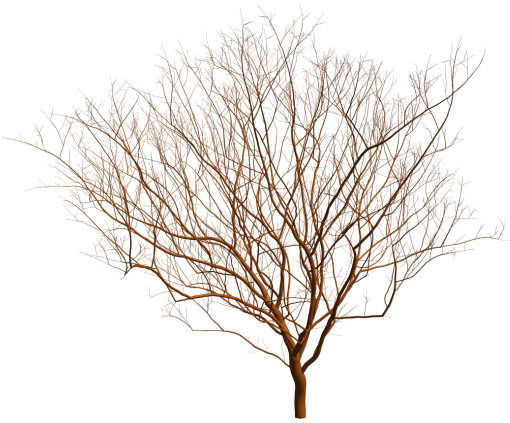
Parametry samostatných stromů jsou součástí instalace naší aplikace na přiloženém CD a jednotlivé modely je na jejich základě možné jednoduše reprodukovat.



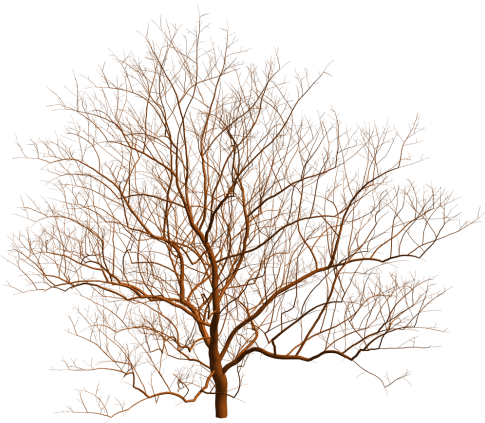
(a) Výchozí



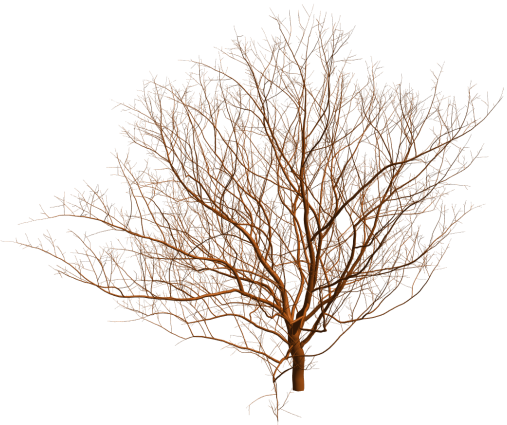
(b) Vertikální pupeny



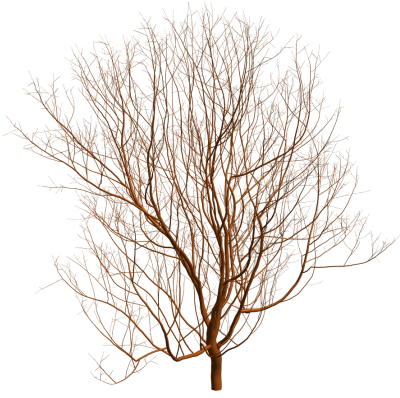
(c) Pupeny vzhůru



(d) Pupeny vzhůru, tropismus dolů

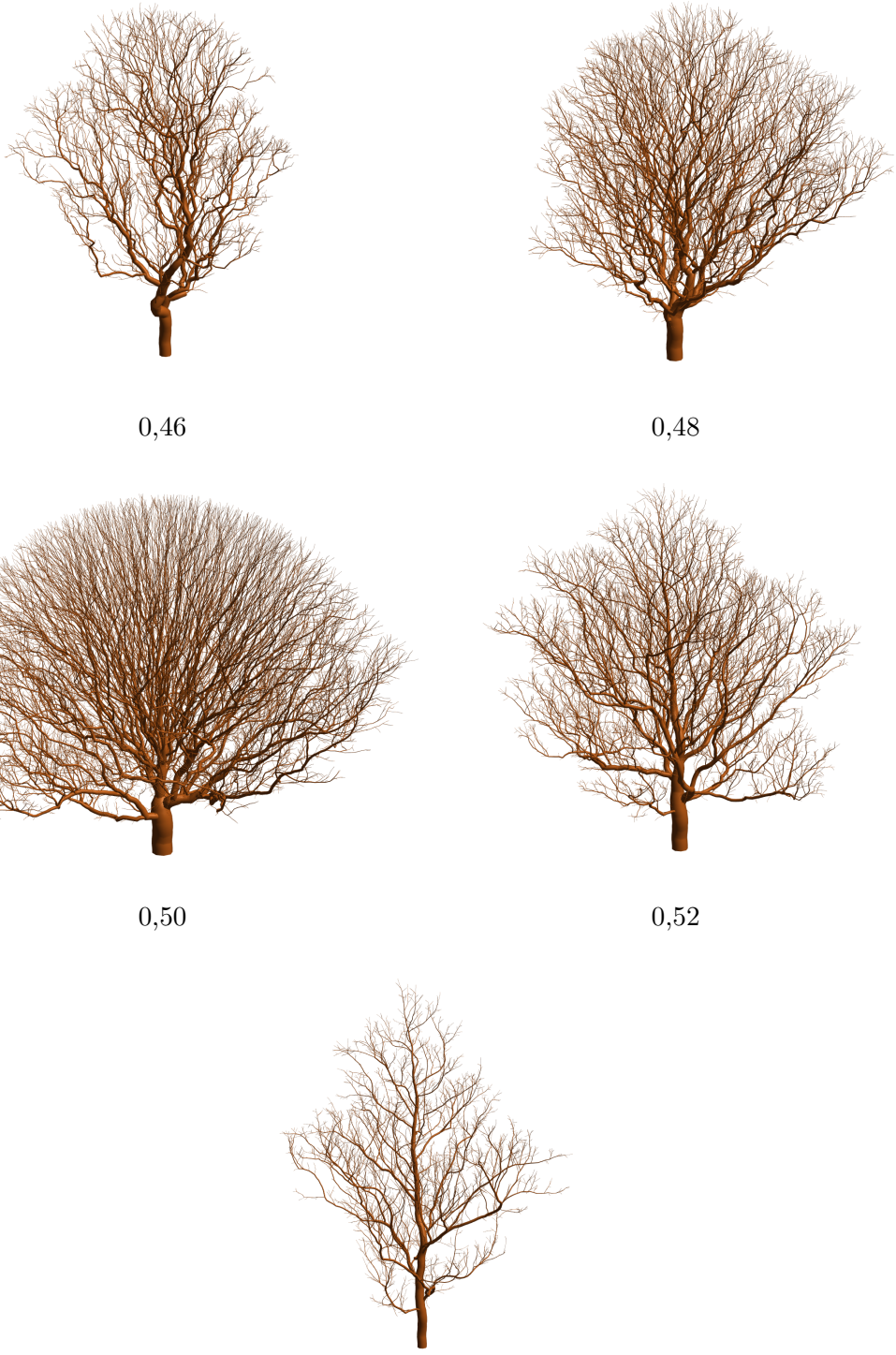


(e) Pupeny dolů

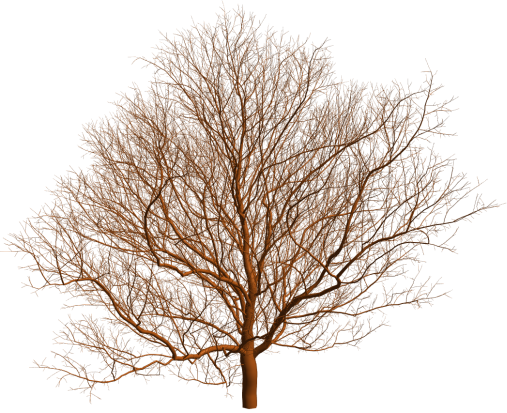


(f) Pupeny dolů, tropismus vzhůru

Obrázek 5.1: Projevy gravimorfismu.
Podle preference specifických typů bočních pupenů se mění vlastnosti výsledného stromu. Ve výchozím modelu (a) mají všechny pupeny stejnou prioritu. Při preferenci všech vertikálně směřujících pupenů (vzhůru i dolů) je projev poměrně neurčitý (b). Preference vzhůru či dolů ovlivňuje tvar koruny stromu: zda je užší (c), nebo košatější (e). Při souhře gravimorfismu se silou ovlivňující směr růstu (tropismem) v opačném směru vznikají specifické tvary větví, které nelze jinak reprodukovat (d, f).

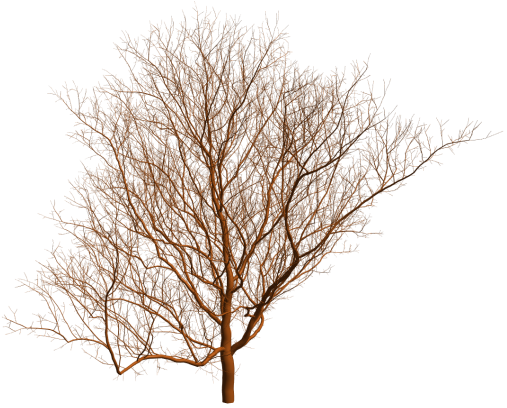


Obrázek 5.2: Projev preference růstu vrcholových pupenů.
Preference vrcholových pupenů (parametr b v Rozšířeném Borchert-Hondově modelu) má silný dopad na výsledný tvar stromu. Pro hodnoty menší než 0,5 nevyroste žádná větev do velké délky, neboť růst vždy převezmou nové boční větve. Pro hodnoty větší než 0,5 jsou hlavní větve delší a výraznější než boční, které jsou potlačovány. Při 0,5 jsou preference vyrovnané a všechny větve rostou rovnoměrně, díky čemuž je výsledný strom nejhustší.

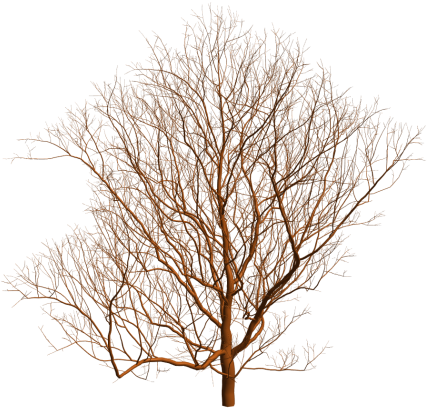


(a) Výchozí - bez opadávání větví

(b) Opadávání větví

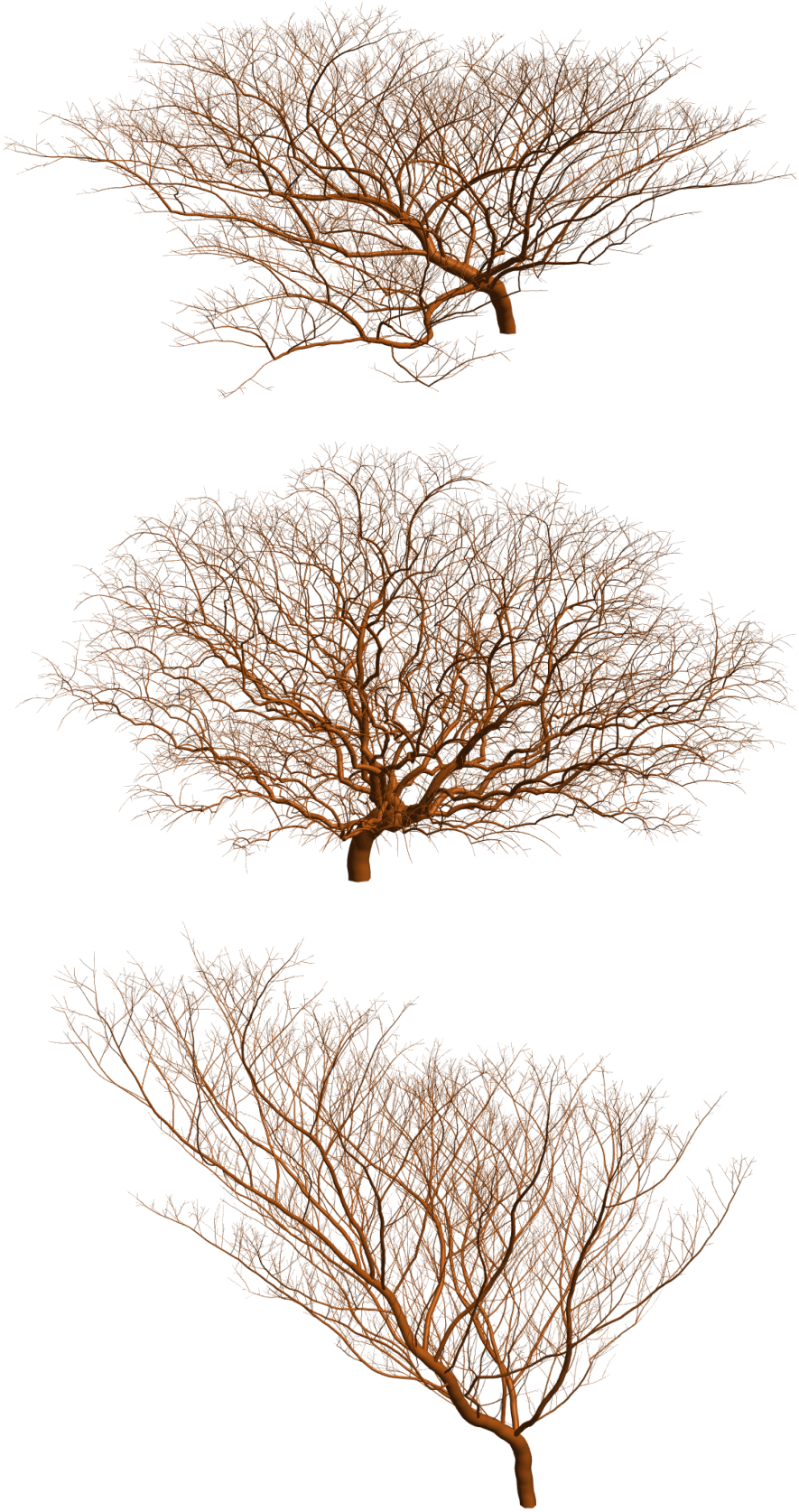


(c) Vyšší citlivost na světlo



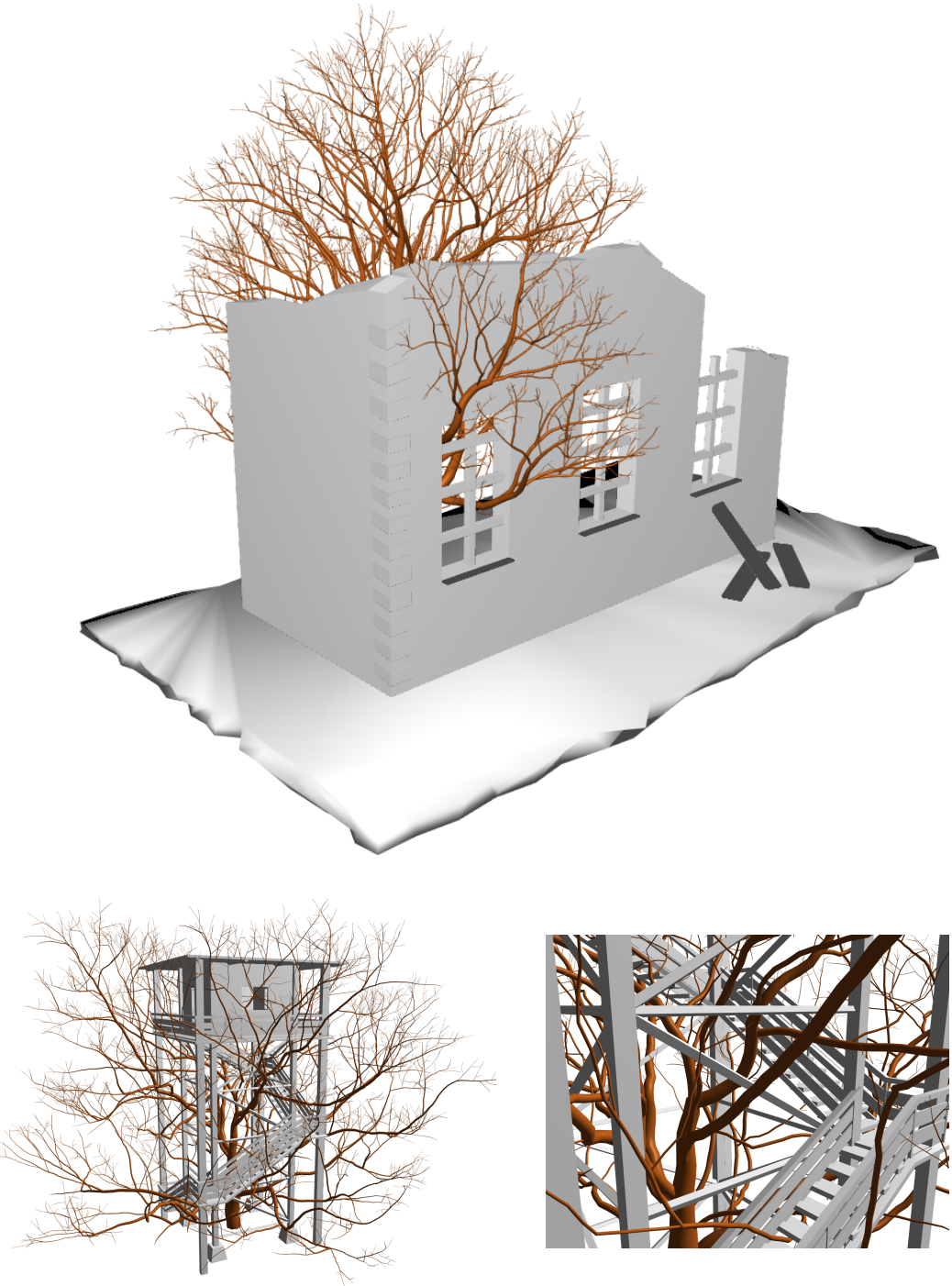
(d) Vyšší citlivost a opadávání větví

Obrázek 5.3: Projev citlivosti na světlo a opadávání větví.
Koruna výchozího modelu (a) je velmi hustá. Při aplikaci opadávání větví je rozložení hlavních větví poměrně zachováno a malé větvičky zahušťující hlavně centrum koruny jsou odstraněny (b). Alternativně navýšením citlivosti pupenů na světlo je také dosaženo zmenšení hustoty větví, ale dochází k větším změnám tvaru koruny (c). Může za to změna volby pupenů, z nichž vyráží nové výhony, kde větší prioritu mají ty, které jsou méně zastíněny (na okrajích a hlavně na vrchu koruny). Při aplikaci opadávání větví na strom citlivější na světlo (d) zůstávají hlavní rysy shodné, ale množství shozených větví není tak vysoké, protože už v původním stromě (c) jich je méně.

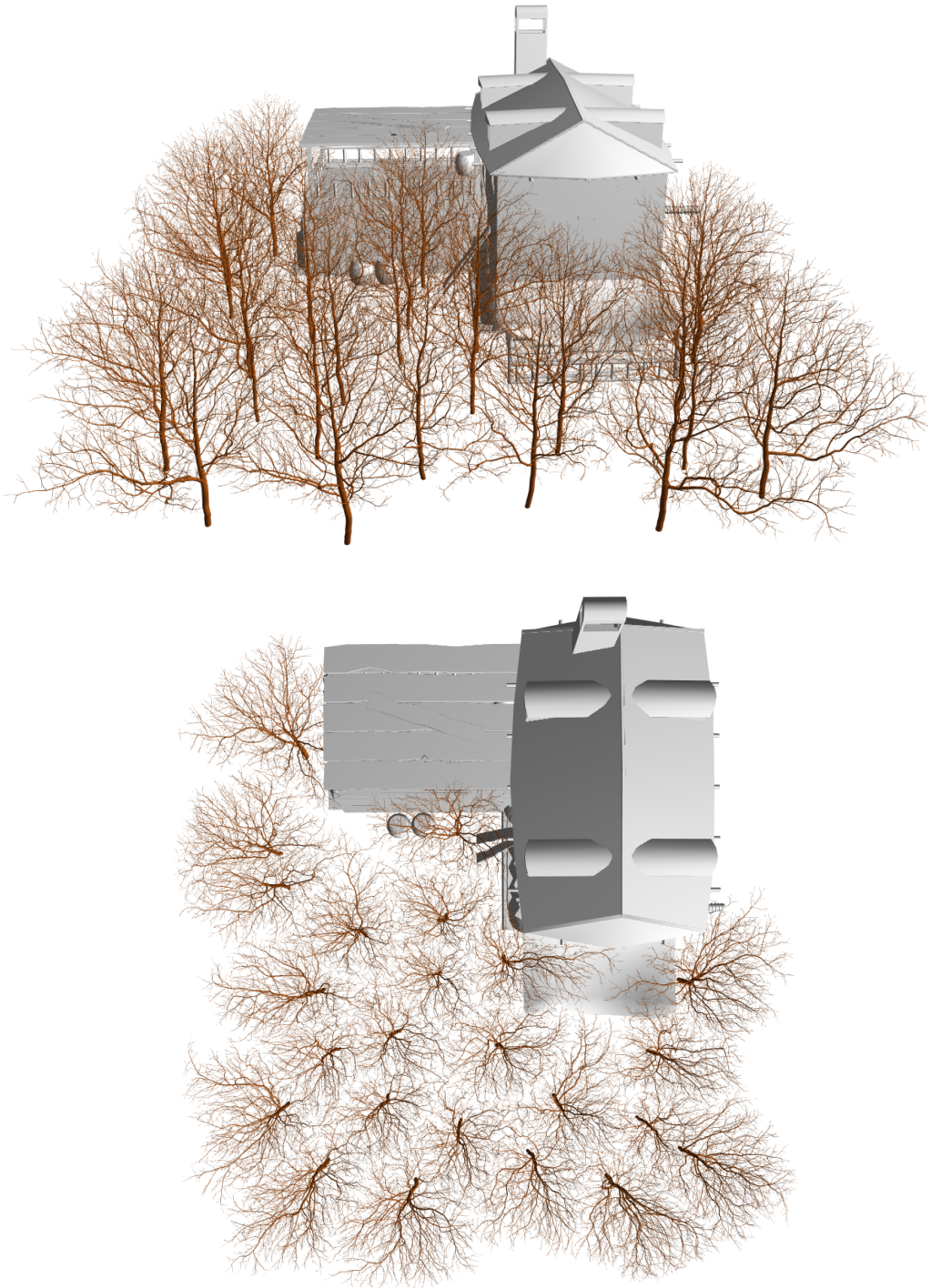


Obrázek 5.4: Stromovité struktury s aplikovaným tropismem na kmen. Tyto tvary připomínají stromy na prudkých svazích a skalách, nebo keře.

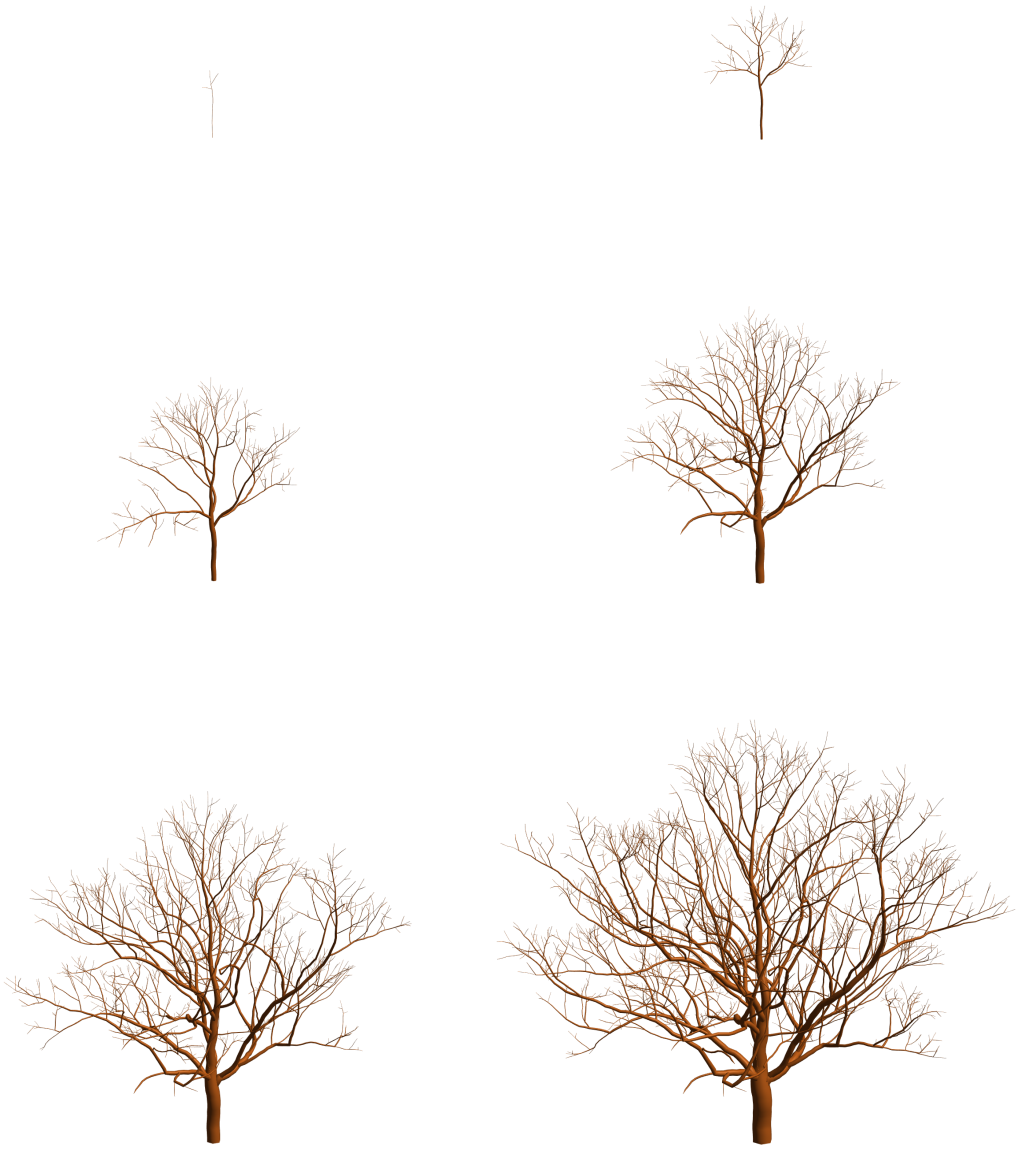
Obrázek 5.5: Jednoduchá interakce stromu se scénou.
Vrchní obrázek prezentuje předpokládané nejjednodušší využití možnosti interakce rostoucího stromu se scénou. Jedná se o interakci stromu s objektem (zde dům), který v sobě nemá žádné otvory. Na spodních obrázcích jsou pohledy podél zdí, z kterých je zřejmé, že větve do domu neprorůstají. Naopak na větvích u zdí je možné pozorovat, jak se ohýbají, aby se domu vyhnuly.



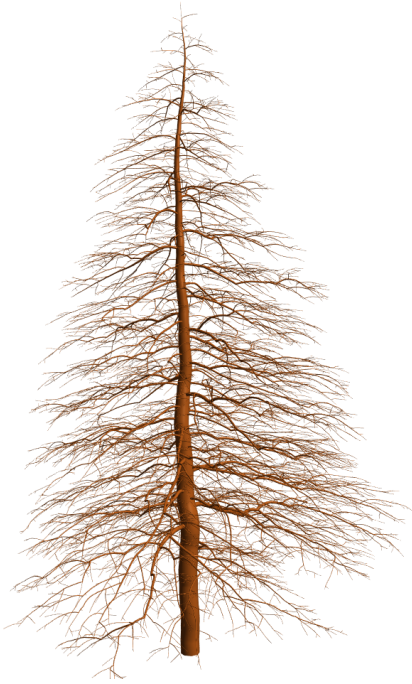
Obrázek 5.6: Komplikovaná interakce stromu se scénou.
Realistický případ komplikovanější interakce stromu s objektem znázorňuje vrchní obrázek, kde některé větve stromu rostou úzkými otvory ve zdi. Model na spodních obrázcích není příliš přirozený, ale slouží jako zátěžový test pro růst v extrémnějších podmínkách. Nalevo je celý model zobrazen z dálky, zatímco napravo je vidět detail koruny. Je nutné podotknout, že interakce se scénou nebere v potaz tloušťku větví (generovaná stromová struktura je soustava vrcholů a hran mezi nimi), takže větev může být (nebo spolu s dalším růstem se stát) širší, než otvor, kterým prorostla.



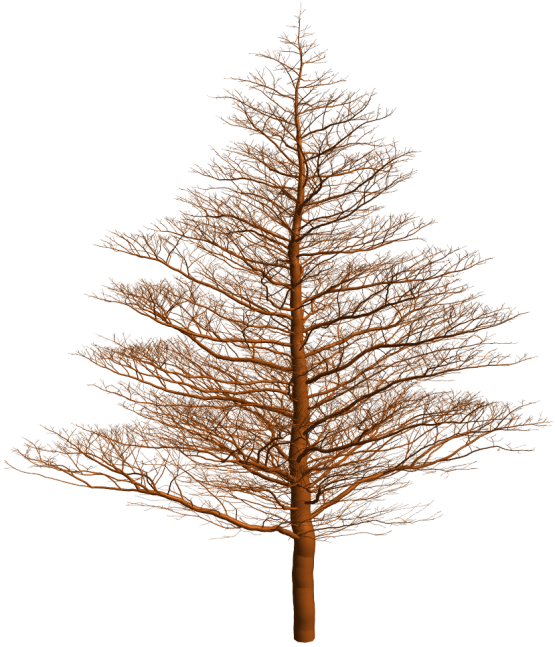
Obrázek 5.7: Více stromů ve scéně.
Reprezentace prostoru a dostupného světla v generativním algoritmu nijak nerozlišují mezi jednotlivými stromy, a proto se v rámci nich může více stromů vzájemně ovlivňovat. Při růstu stromů vedle sebe tak nedochází k jejich prolínání, ale naopak, svůj růst vzájemně silně omezují (dolní obrázek).



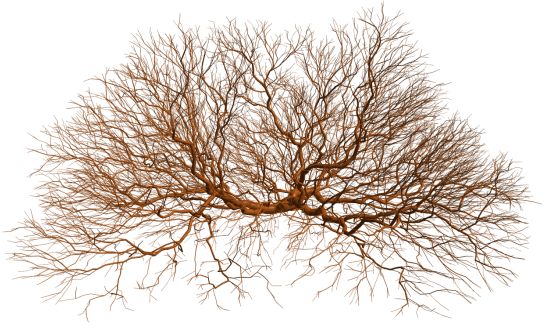
Obrázek 5.8: Postupný růst stromu.
Na obrázcích je zachycen růst stromu v intervalu pěti iterací algoritmu. Je zřejmé, že množství generovaných větví za jednu iteraci stoupá s přibývajícím stářím stromu. Také je možné pozorovat průběžné opadávání některých menších větví.



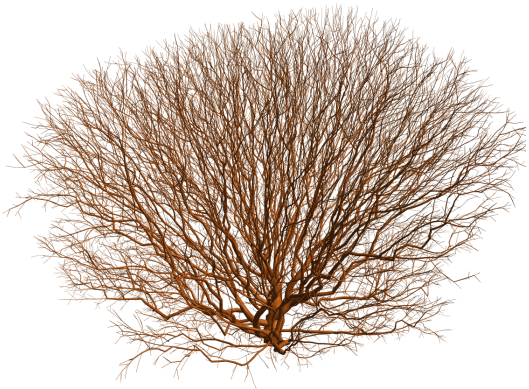
Smrk



Borovice

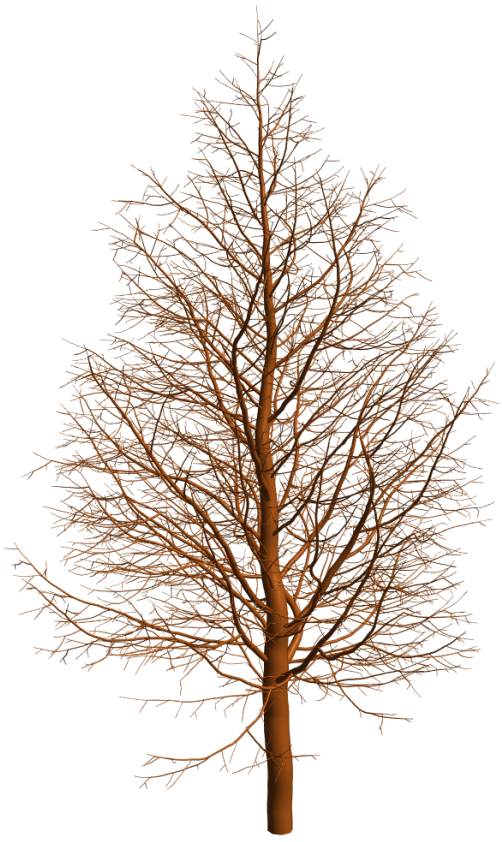


Kosodřevina

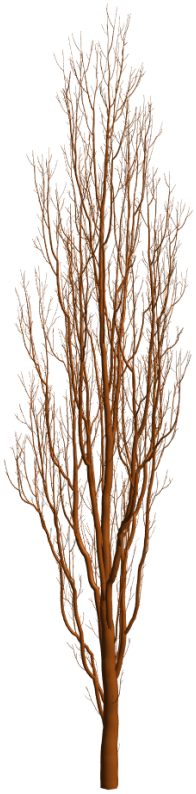


Keř

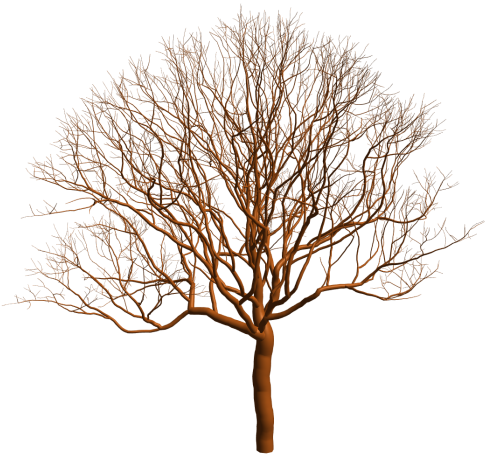
Obrázek 5.9: Jehličnaté stromy a další dřeviny.



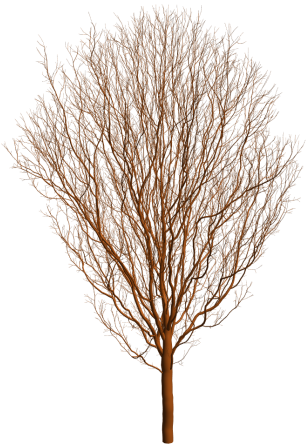
Bříza



Topol

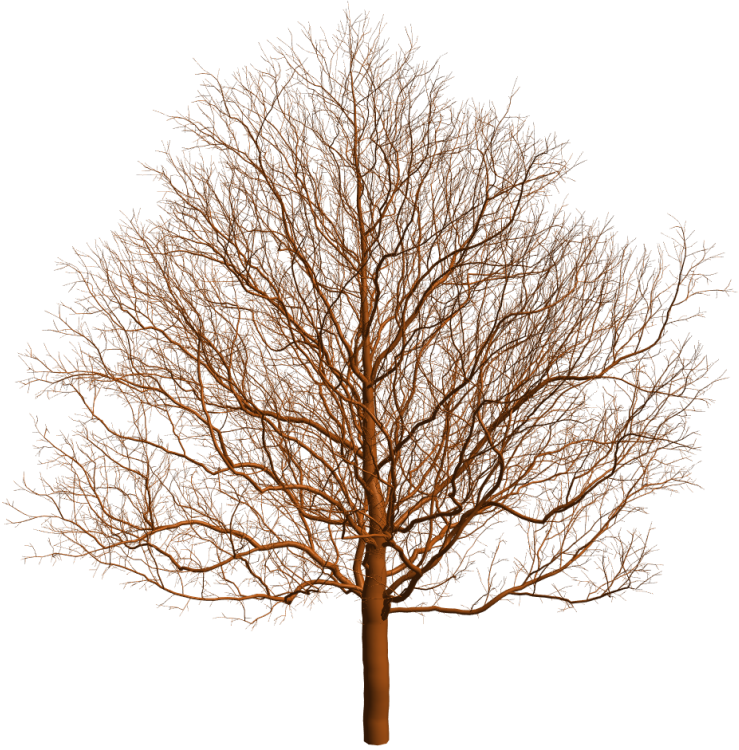


Ovocný strom



Švestka

Obrázek 5.10: Listnaté a ovocné stromy.



Lípa

Starší dub/buk

Obrázek 5.11: Listnaté stromy s mohutnější korunou.

Pro prezentované modely stromů s deklarovaným rodovým názvem uvádíme jejich vlastnosti (tabulka 5.1) a časy, které generátor strávil v jednotlivých částech algoritmu, aby je vytvořil (tabulka 5.2).

Model	Vrcholy stromu	Vrcholy meshe	Trojúhelníky meshe
Smrk	31 633	302 430	549 360
Borovice	59 070	572 319	1 033 128
Kosodřevina	22 828	187 425	317 178
Keř	34 956	286 555	485 310
Bříza	15 296	147 439	265 688
Topol	8 559	67 210	116 292
Ovocný strom	14 748	135 316	251 072
Švestka	23 910	259 042	442 784
Lípa	41 806	463 032	783 184
Starší dub/buk	34 127	256 939	453 102

Tabulka 5.1: Vlastnosti modelů.
Počet vrcholů meshe není násobkem počtu vrcholů stromu, protože při tvorbě meshe má každá začínající větev v místě větvení svůj počáteční vrchol (tedy původní vrchol větvení ve stromové struktuře je pro každou novou větev zduplikován).

Model	Zdroje	Prostor	Růst	Opadávání	Struktura	Mesh
Smrk	114,87	68,98	77,80	52,07	411,25	62,52
Borovice	460,68	227,35	235,00	239,57	1300,97	149,667
Kosodřevina	26,93	50,90	23,83	0	145,32	40,88
Keř	56,12	79,65	51,05	0	248,82	65,48
Bříza	36,62	28,12	28,95	20,05	184,22	30,35
Topol	10,55	1,87	0,15	0	64,53	11,95
Ovocný strom	70,12	118,77	114,42	107,65	507,95	60,20
Švestka	43,98	36,95	28,10	0	175,3	46,78
Lípa	85,88	83,47	69,77	0	313,70	84,95
Starší dub/buk	113,57	107,58	93,92	90,43	505,15	76,42

Tabulka 5.2: Časová náročnost generování modelů.
Hodnoty udávají časy v jednotkách milisekund. Zdroje - veškeré výpočty nutné k zvolení pupenů pro růst výhonů (3.4.1 až 3.4.5). Prostor - výpočty v rámci prostorového modelu (3.2). Růst - určení směru a uskutečnění růstu v něm (3.4.6 a 3.4.7). Opadávání - případná aplikace opadávání větví (3.4.8). Struktura - celková doba běhu generátoru. Mesh - čas nutný pro vytvoření meshe nad vygenerovanou stromovou strukturou (3.6).

Všechna měření byla uskutečněna na stolním počítači s:

- Microsoft Windows 10 Pro 64-bit,
- Intel Core i5-3570K (4 fyzická jádra @ 3,4 GHz - bez hyper-threadingu),
- 8 GB RAM, 2 x 4 GB, 1333 MHz.

Kód byl přeložen pomocí Microsoft Visual Studia 2017 Enterprise s optimalizačním nastavením /O2 a /fp:fast. Měření byla uskutečněna pomocí hodin ve standardní knihovně C++11, konkrétně pomocí std::chrono::steady_clock. Pro každý strom bylo provedeno šedesát měření. Hodnoty uvedené v tabulce byly získány jako aritmetický průměr z naměřených hodnot.

Z tabulky 5.2 je možné vidět, že zastoupení jednotlivých částí simulačního procesu na celkovém času generování se nedá zobecnit. Nezávisí jen na velikosti výsledného modelu, ale na specifických parametrech každého stromu. Například, lípa má o třetinu více vrcholů než smrk, ale je vygenerována rychleji.

Za zmínku stojí to, že přes 90 % času stráveného řešením zdrojů zabírá Rozšířený Borchert-Hondův model, který spolu s opadáváním větví nelze jednoduše paralelizovat. Naopak prostorový model zabírá nejvíce procesorového času, ale je nejlépe paralelizován, což se silně projevilo na naměřených hodnotách. Celková doba běhu generátoru je vyšší než součet jednotlivých částí, protože v sobě obsahuje další operace jako inicializaci, správu běhu a synchronizaci simulace.

Závěr

V rámci této práce jsme započali vývoj aplikace pro generování trojrozměrných modelů stromů cílený spíše na příležitostné uživatele, než na zkušené profesionály. Našimi hlavními požadavky pro implementaci byly: jednoduchost ovládání, velký rozsah a věrohodnost vytvářených modelů stromů a schopnost nástroje při tvorbě modelu interagovat se scénou. Nároky byly také kladeny na efektivitu implementovaného řešení, aby byl výsledný nástroj co nejvíce interaktivní, což usnadňuje jeho použití.

Dosažené cíle

Jako algoritmus pro tvorbu stromů jsme zvolili Sebe-uspořádající stromy, jež představují simulaci růstu založenou na vnitřních přírodních procesech uvnitř stromu a jeho interakci s prostředím. To umožnilo zavedení malé množiny nastavitelných parametrů (celkem 16), které dostačují k možnosti generovat širokou škálu tvarů dřevin - od listnatých stromů přes jehličnaté stromy po keře a další nízké dřeviny. Uvědomujeme si, že v důsledku toho, že parametry popisují různé biologické vlastnosti stromu, nemusí být jejich projev na výsledný strom vždy snadno předvídatelný. Domníváme se, že tento nedostatek může být redukován vylepšením uživatelského rozhraní výsledné aplikace - například zadáváním parametrů pomocí grafických ovládacích prvků namísto přímo číselných hodnot.

Myslíme si, že věrohodnost generovaných modelů je velmi vysoká, obzvlášť bereme-li v potaz to, že uživatel proces v průběhu nijak neovlivňuje. Největším nedostatkem je chybějící simulace působení gravitace na strom, která by měla za následek ohýbání a případné další opadávání větví, což by velmi podpořilo možnost vytvářet modely starších stromů.

Implementovaná interakce generátoru se scénou se ukázala jako velmi robustní. Reaguje nejen na plochy bez děr (např. stěny budov), kde jimi výsledný strom neprorůstá, ale je schopná pro růst korektně využít i menší volné otvory v objektech.

I přesto, že se nám nepovedlo paralelizovat algoritmus úplně, dokázali jsme udržet rychlost generování modelů stromů na velmi přívětivých hodnotách, které pro většinu modelů na běžném stolním počítači nepřesáhnou jednu vteřinu.

Budoucí práce

I když modely generované naší implementací jsou prakticky plně použitelné, před distribucí samotné aplikace by bylo vhodné rozšířit její funkcionalitu o:

- Listy stromu
  - Ačkoliv listy je možné dodatečně přidat v 3D modelovacích nástrojích, bylo by přirozené, aby tak učinil generátor stromů sám.

- Detaily kmene a větví
  - Spojitý trojúhelníkový mesh v místech větvení (obecněji odstranění případné překrývající se geometrie).
  - Rozšíření kmene u základny s přechodem do rozbíhajících se kořenů.
- Level of detail (LOD)
  - Generované modely stromů obsahují nadbytečné množství geometrie (obzvláště na velkém množství tenčích větví), kterou by bylo vhodné odstranit.
  - Obecně nejlepším řešením by byl nastavitelný LOD, který by umožnil pro strukturu stromu vytvořit mesh v rozmezí od tisíců až po miliony trojúhelníků.
- Zlepšení uživatelského rozhraní
  - Ať už pro samostatnou aplikaci, nebo při integraci generátoru ve formě pluginu do jiného nástroje.

Další potenciální volně navazující témata jsou:

- Vystavět na myšlenkách Sebe-uspořádajících stromů nový, vylepšený algoritmus.
  - Původní algoritmus vyžaduje vymezení oblasti pro tvorbu modelu (z důvodu nutnosti použití trojrozměrné mřížky pro výpočty). Bylo by ale vhodné, aby žádné omezení tohoto typu neexistovalo. Myslíme si, že řešením by mohl být postup založený na vrhání paprsků, které jsme již využili pro interakci se scénou.
- Doplnění algoritmu o simulaci dalších jevů
  - Gravitace způsobující ohýbání a opadávání větví.
  - Povětrnostní podmínky ovlivňující výsledný tvar celého stromu.
- Podpora generování více stromů zároveň
  - I když naše implementace umožňuje generování více stromů zároveň, uživatelské rozhraní k tomu není přizpůsobeno.
  - Obzvláště zajímavé by bylo vytváření celých úseků lesa založených na zadané distribuci jednotlivých druhů stromů v něm.
- Procedurální generování dalších částí stromu
  - Tvary listů, textury listů a kůry, plody a květy stromů.
- Mapování reálných stromů na parametry algoritmu
  - Naučit neuronovou síť mapovat fotografie stromů na parametry v algoritmu, pomocí nichž by bylo možné reprodukovat stromy se stejnými rysy jako má předloha.

Seznam použité literatury

[1] Exlevel. GrowFX. [online]. [cit. 10.07.2018]. Dostupné z: https://exlevel.com/.

[2] IDV, Inc. SpeedTree Vegetation Modeling - SpeedTree. [online]. [cit. 10.07.2018]. Dostupné z: https://store.speedtree.com/.

[3] e-on software. PlantFactory. [online]. [cit. 10.07.2018]. Dostupné z: https://info.e-onsoftware.com/plantfactory.

[4] Autodesk. 3ds Max. [online]. [cit. 10.07.2018]. Dostupné z: https://www.autodesk.com/products/3ds-max/overview.

[5] Onyx Computing. Onyx Tree. [online]. [cit. 10.07.2018]. Dostupné z: http://www.onyxtree.com/.

[6] Scott Cherba. Tree Modeling Software. [online]. [cit. 10.07.2018]. Dostupné z: http://www.cherba.com/wcs/tutorials/vns/090304/index.html.

[7] The Grove. Wybren van Keulen. 3D Tree Growing Software - The Grove. [online]. [cit. 10.07.2018]. Dostupné z: https://www.thegrove3d.com/.

[8] The Blender Foundation. Home of the Blender project - Free and Open 3D Creation Software. [online]. [cit. 10.07.2018]. Dostupné z: https://www.blender.org/.

[9] Radomír Měch and Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with Their Environment. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 397–410, New York, NY, USA, 1996. ACM.

[10] Yodthong Rodkaew, Prabhas Chongstitvatana, Suchada Siripant, and Chidchanok Lursinsap. Particle Systems for Plant Modeling. In *Plant Growth Modeling and Applications. Proceedings of PMA03*, 01 2003.

[11] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing Tree Models for Image Synthesis. *ACM Trans. Graph.*, 28(3):58:1–58:10, July 2009.

[12] Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. TreeSketch: Interactive Procedural Modeling of Trees on a Tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 107–120, Goslar Germany, Germany, 2012. Eurographics Association.

[13] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling Trees with a Space Colonization Algorithm. In *Proceedings of the Third Eurographics Conference on Natural Phenomena*, NPH'07, pages 63–70, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[14] Daniel Barthélémy and Yves Caraglio. Plant Architecture: A Dynamic, Multilevel and Comprehensive Approach to Plant Form, Structure and Ontogeny. *Annals of Botany*, 99(3):375–407, 2007.

[15] Jules Bloomenthal. Modeling the Mighty Maple. *SIGGRAPH Comput. Graph.*, 19(3):305–311, July 1985.

[16] G-Truc Creation. OpenGL Mathematics. [online]. [cit. 10.07.2018]. Dostupné z: https://github.com/g-truc/glm.

[17] Tino Kluge. Cubic Spline interpolation in C++. [online]. [cit. 10.07.2018]. Dostupné z: https://github.com/ttk592/spline/.

[18] Intel Corporation. Embree. [online]. [cit. 10.07.2018]. Dostupné z: https://embree.github.io/.

[19] Intel Corporation. Threading Building Blocks. [online]. [cit. 10.07.2018]. Dostupné z: https://www.threadingbuildingblocks.org/.

[20] Syoyo Fujita. TinyObjLoader. [online]. [cit. 10.07.2018]. Dostupné z: https://github.com/syoyo/tinyobjloader.

[21] The Qt Company. Qt Cross-platform software development for embedded & desktop. [online]. [cit. 10.07.2018]. Dostupné z: https://www.qt.io/.

[22] Khronos Group Inc. OpenGL - The Industry Standard for High Performance Graphics. [online]. [cit. 10.07.2018]. Dostupné z: https://www.opengl.org/.

[23] The Qt Company. Qt Visual Studio Tools - Visual Studio Marketplace. [online]. [cit. 10.07.2018]. Dostupné z: https://marketplace.visualstudio.com/items?itemName=TheQtCompany.QtVisualStudioTools-19123.

[24] Outerra. Maximizing Depth Buffer Range and Precision. [online]. Dostupné z: http://outerra.blogspot.com/2012/11/maximizing-depth-buffer-range-and.html.

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Příloha I - Uživatelská dokumentace

Obsahem tohoto dokumentu jsou informace o požadavcích pro spuštění, instalaci a ovládání aplikace.

Požadavky pro spuštění

- Procesor podporující SSE2 instrukce, což je v podstatě libovolný procesor z posledních čtrnácti let. Aplikace spuštěná na starším hardware neproběhne korektní inicializací a uživatel je na tento problém upozorněn.
- Grafická karta podporující OpenGL 3.3. Dedikované grafické karty by neměly mít s tímto požadavkem žádný problém. Varujeme, že nejstarší integrované karty nemusí být schopné podpory OpenGL 3.3. Pokud není OpenGL 3.3 dostupná, uživatel je varován a aplikaci (za splnění ostatních podmínek) je možné použít, ale modely stromů nebudou vykreslovány.
- 64-bitový Windows 10.
- Microsoft Visual C++ 2017 Redistributable (x64) 10.0.16299.0, nebo vyšší.

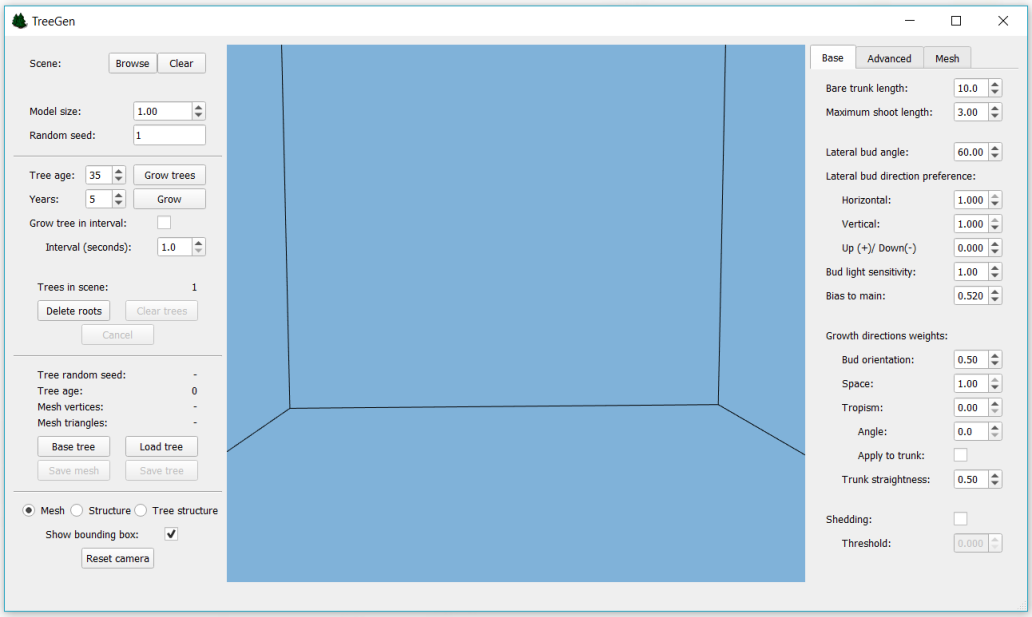
Instalace

Celá aplikace je ve složce *Treegen Application* a její spuštění je možné přímo z přenosného média, i když doporučujeme její nahrání na disk kvůli lepšímu výkonu. Pokud se aplikaci nepovede vůbec spustit a vyskytne se zpráva oznamující chybějící msvcp140.dll, nebo jí podobná, pak velice pravděpodobně není nainstalována dostatečná verze Microsoft Visual C++ 2017 Redistributable (x64). Na CD s aplikací tedy dodáváme instalátor aktuální verze: VC_redist.x64.exe. Alternativně lze nechat proběhnout aktualizaci Windows 10, která by měla Visual C++ 2017 Redistributable (x64) nainstalovat. Pokud aplikace nastartuje, ale je hlášena chyba, že systém nepodporuje OpenGL 3.3, doporučujeme aktualizovat ovladače grafické karty. I když v takovém případě je pravděpodobnější, že grafická karta vůbec není schopná běhu OpenGL 3.3.

Ovládání aplikace

Uživatelské prostředí aplikace je rozděleno na tři základní části. V modře podbarveném okně je vykreslována scéna a vygenerované stromy. Na levém panelu jsou základní ovládací prvky, jako základní nastavení a spouštění generátoru, ukládání a nahrávání parametrů stromu, ukládání vygenerovaných modelů, nahrávání scény a ovládání kamery. Pravý panel obsahuje veškeré parametry stromu, na jejichž základě probíhá generování výsledných modelů.

Aplikace je ovládána pomocí myši a klávesnice. Pro nastavování číselných hodnot je silně doporučeno používat kolečko myši. Ukazatel myši stačí přesunout



Uživatelské rozhraní aplikace.

na nastavitelnou hodnotu (s šipkami nahoru a dolů) a rotací kolečka ji upravit. Ve většině případů tak není nutné psát konkrétní čísla nebo opakovaně klikat na šipky u jednotlivých hodnot.

Ovládání zobrazení

Pohled a pozici kamery vykreslovacího okna je možné měnit. Tahem myši po okně při stisknutém levém tlačítku myši dochází k rotaci kolem bodu před kamerou. Použitím kolečka myši lze změnit vzdálenost kamery od tohoto bodu, čímž můžeme rotovat po kratších (bližších) nebo delších (vzdálenějších) kružnicích. Pokud provádíme tah myši při stisknutém tlačítku Alt, pak dochází k pohybu kamery v jejích lokálních souřadnicích x a y (pohyb nahoru, dolů, doleva a doprava). Pokud je Alt stisknut při rotaci kolečka myši, pak se kamera pohybuje dopředu nebo dozadu vzhledem k její stávající orientaci. Obě varianty pohybu při použití tlačítka Alt mění stejným způsobem jak pozici kamery ve scéně, tak zmíněný bod před ní. Při nahrání scény jsou vlastnosti kamery automaticky nastaveny tak, aby uživatel viděl celou scénu pod úhlem z boku a případným tahem myši rotoval kolem jejího středu. Do tohoto základního nastavení kamery se lze kdykoliv vrátit použitím tlačítka *Reset camera* na levém ovládacím panelu.

Ve spodku levého panelu můžeme zaškrtnutím *Show bounding box* vybrat, zda chceme zobrazovat krychli reprezentující část scény, v níž aplikace generuje výsledný strom. Velikost této oblasti je možné změnit (viz Správa a růst stromů). Výběrem z *Mesh*, *Structure* a *Tree structure* udáváme způsob, kterým je vykreslován vygenerovaný strom:

- *Mesh* - vykreslení plných trojúhelníků meshe.
- *Structure* - vykreslení hran trojúhelníku meshe.

- *Tree Structure* - vykreslení soustav hran a vrcholů reprezentujících strukturu stromu. V tomto módu jsou vidět i pozice nevzrostlých stromů.

Nahrávání scény

Je umožněno nahrát libovolnou scénu ve formátu .obj souboru obsahujícím normálové vektory. Pokud je některé objekty v něm neobsahují, pak nebudou načteny. V případě, že soubor neobsahuje žádnou validní geometrii, není scéna nahrána. V obou případech je na problém uživatel upozorněn. Vertikální osou scény musí být osa y, aby byla scéna stejně orientována jako generovaný model v aplikaci. Pro nahrání scény a její vymazání (tj. načtení prázdné scény) jsou určena tlačítka *Browse* a *Clear* v horní části levého panelu.

Správa a růst stromů

Pro přidání pozice kořenu stromu je nutné namířit myší na požadované místo ve scéně a stisknout klávesu N. První strom definuje oblast scény, ve které bude program generovat výsledné stromy. Proto další následně přidané pozice stromů mohou být jen v ní. Velikost této oblasti lze upravit pomocí parametru *Model size*, který lze měnit, pokud je ve scéně nejvýše jeden strom. Tento parametr tak slouží ke škálování velikosti generovacího prostoru (a tedy i stromů v něm) vůči scéně. V prázdné scéně je vždy (automaticky) definována pozice stromu, tedy prázdná scéna odpovídá generovacímu prostoru.

Pro smazání všech pozic stromů slouží tlačítko *Delete roots*. Pro situaci, kdy stromy nechceme úplně odstranit (třeba pro opakování jejich postupného růstu), je zamýšleno použití *Clear trees*. To vrátí všechny stromy ve scéně do stavu, kdy ještě nezačaly růst.

Generovat modely stromů je možné třemi způsoby:

1. Tlačítko *Grow trees* - všechny stromy vyrostou do zadaného stáří.

2. Tlačítko *Grow* - všechny stromy povyrostou o specifikovaný počet let. Jeho opakovaným použitím můžeme sledovat postupný růst stromů.

3. Zvolení *Grow trees in interval* a nastavení hodnoty *Interval*. Jedná se o opakované spouštění *Grow trees*, kde po vygenerování stromu a před spuštěním tvorby dalšího, proběhne zadaný interval. Díky němu můžeme měnit parametry stromů a sledovat jejich dopad na průběžně generovaných modelech. Dalším použitím je nechat generátor vytvářet modely a ukládat si jen ty, které se nám líbí.

Generování modelu může být kdykoliv za běhu zrušeno pomocí tlačítka *Cancel*. Aby k tomu nebylo nutné přistoupit, je doporučeno zvolit výchozí věk stromu v málo desítkách let a až pak ho případně navyšovat.

Pro jedno konkrétní nastavení vlastností můžeme generovat mnoho podobných stromů na základě hodnoty *Random seed*. Při hledání vhodných parametrů stromu je doporučeno ponechat tuto hodnotu zafixovanou, aby byl pozorován pouze projev jejich změn a nikoliv interní nastavení náhodných generátorů. Naopak, ve chvíli, kdy chceme generovat více stromů stejného typu, stačí ponechat

položku *Random seed* prázdnou - pro každý generovaný strom je pak dosazena náhodně.

Jakmile je strom vygenerován, jsou zobrazeny jeho základní parametry, jako semínko náhodného generátoru, věk stromu a počet vrcholů a trojúhelníků meshe. Takový model můžeme uložit - buď přímo jeho mesh ve formě .obj souborů (*Save mesh*) nebo momentální parametry viditelné v uživatelském prostředí (*Save tree*), na jejichž základě je možné ho znovu v aplikaci vytvořit. V případě, že ve scéně je více stromů, uloženy do .obj souboru jsou všechny jako samostatné objekty. Parametry lze kdykoliv načíst (*Load tree*), nebo vrátit do podoby při spuštění aplikace (*Base tree*). K aplikaci je přiloženo několik ukázkových sad parametrů pro některé z různých druhů stromů.

Je důležité upozornit, že libovolná změna parametrů se projeví až při spuštění generátoru. Proto je právě vhodný mód *Grow trees in interval*, který použije případné nové parametry pro růst celého stromu daného stáří. Jinak je možné nechat růst strom postupně (*Grow*) a parametry měnit průběžně. K tomu ale není aplikace uzpůsobena.

Základní parametry

První záložka pravého panelu (*Base*) obsahuje všechny parametry vhodné pro ovlivnění tvaru výsledného stromu:

- *Bare trunk length*
  - Určuje délku kmene od kořene vzhůru, kde nebudou žádné boční větve.
  - Má smysl nastavit alespoň na nízkou hodnotu pro všechny listnaté stromy, aby nevznikaly větve těsně nad zemí.
- *Maximum shoot length*
  - Určuje, jak dlouhé výhony větví vyrostou každý rok.
  - Pokud je upravována priorita růstu různých typů pupenů, pak je vhodné nastavit maximální délku výhonů na větší hodnotu. To umožní, aby rostly výhony různé délky na základě priority jednotlivých pupenů.
  - Při vyšších hodnotách má za následek řidší korunu.
  - Při zvýšení délky výhonů je nutné adekvátně snížit věk stromu.
- *Lateral bud angle*
  - Úhel mezi větví a bočním pupenem.
  - Čím vyšší hodnota, tím košatější strom je.
- *Lateral bud direction preference*
  - Výběr preferovaných bočních pupenů v závislosti na jejich orientaci na větvi.

  - Hodnoty *Horizontal* a *Vertical* určují preferenci pupenů ve stejnojmenných směrech. Růst vrcholových pupenů tímto parametrem nijak ovlivněn není, takže při snížení hodnot *Horizontal* a *Vertical* obecně omezíme růst bočních pupenů. Doporučené použití je zachovat jeden z preferovaných směrů na maximu a druhý snížit. Pro pupeny ve vertikální orientaci je vhodné specifikovat, zda preferujeme orientaci vzhůru nebo dolů (parametr *Up(+)/Down(−)*).

- *Bud light sensitivity*
  - Citlivost pupenů na světlo ovlivňuje, jak dobré světelné podmínky pupen potřebuje k tomu, aby z něj vyrostl výhon.
  - Čím je hodnota vyšší, tím méně a kratších výhonů roste, neboť jsou vyžadovány lepší podmínky, aby růst započal. To má za následek řidší strukturu větví. Jedná se o nejlepší a nejjednodušší způsob, jak hustotu větví stromu ovlivnit.
  - Při nízkých hodnotách může růst takřka každý pupen, což adekvátně prodlužuje čas nutný pro generování stromu. Proto je doporučeno hodnotu parametru spíše zvyšovat, než snižovat, vůči jejímu počátečnímu nastavení.

- *Bias to main*
  - Určuje, zda preferujeme růst vrcholových, nebo vedlejších pupenů stromu (růst hlavních, nebo vedlejších větví).
  - Čím vyšší hodnota, tím větší preference vrcholových pupenů. Pro stromy je nejlepší zvolit alespoň mírné zvýhodnění vrcholových pupenů (hodnota větší než 0,5). Naopak pro tvorbu keřovitých struktur je vhodnější růst bočních pupenů (hodnota menší než 0,5).

- *Growth direction weights*
  - Udává zastoupení jednotlivých složek na rozhodnutí o výsledném směru růstu větve.
  - *Bud orientation* - původní orientace pupenu. Čím vyšší zastoupení, tím přímější větve jsou.
  - *Space* - orientace ve směru volného prostoru. Čím vyšší zastoupení, tím pokroucenější větve jsou.
  - *Tropism* - růst v orientaci dané úhlem (*Angle*). Úhel je vzhledem ke směru vzhůru, tedy 90° je v horizontálním směru a 180° ve směru dolů.
    - *Apply to trunk* - je nutné zvolit, pokud chceme směr tropismu použít i pro růst kmene stromu, což většinou není vhodné.
  - *Trunk straightness* - míra udávající nakolik přímý je kmen stromu.

- *Shedding*
  - Zapíná opadávání větví, jehož projev je tím silnější, čím více větví je ve stromě.

  - *Threshold* - nastavuje míru opadávání větví. Pro základní aplikaci stačí ponechat na nule, případně jen nepatrně navýšit. Opadávání větví je vhodné k vyčištění centra koruny od malých nežádoucích větviček.

Pokročilé parametry

Druhá záložka pravého panelu (*Advanced*) obsahuje parametry, které nejsou určeny pro uživatele bez znalosti použitých algoritmů pro generování stromů. Počáteční nastavení jejich hodnot bylo odvozeno na základě testování tak, aby se parametry v panelu *Base* chovaly dobře.

Parametry meshe

Třetí záložka pravého panelu (*Mesh*) obsahuje parametry pro nastavení vlastností generovaného meshe větví:

- *Branch thickness*
  - Šířka vrcholů větví. Definuje obecnou šířku větví ve stromě.
- *Branching thinness*
  - Čím vyšší hodnota, tím méně se šířka větví propaguje směrem od nejmenších větví do kmene. Tedy chceme-li v celém stromě obdobně široké větve a méně výrazný kmen (většina jehličnanů, bříza apod.), nastavíme vyšší hodnotu parametru. Pro stromy se silnějším výrazným kmenem (dub, buk) učiníme opak.
- *Cyllinders per internode*
  - Určuje míru podrozdělení větví (počet válcovitých segmentů meshe reprezentujících větev), čímž přímo ovlivňuje množství trojúhelníků meshe modelu.
  - Má smysl zvýšit, pokud se nám zdá, že ohyb větví je nedostatečně plynulý.
- *Circumference vertices*
  - Udává počet vrcholů meshe po obvodu větve.
- *Texture*
  - Umožňuje nahrát (*Browse*) a případně vymazat (*Clear*) texturu pro vykreslování meshe stromové struktury.
  - Ve složce aplikace je přiloženo několik použitelných textur.
- *Texture size*
  - Nastavení škálování nahrané textury po délce kmene. Poměr mezi délkou a šířkou textury je zachován automaticky.

Klávesové zkratky

Pro některé z nejdůležitějších funkcí a tlačítek v aplikačním okně jsou dostupné klávesové zkratky:

- Escape - vypnutí aplikace
- N - nový strom na pozici myši ve scéně
- F - *Grow trees*
- G - *Grow*
- J - *Clear trees*
- K - *Delete roots*
- L - *Cancel*
- A - *Save mesh*
- S - *Save tree*
- Q - *Mesh*
- W - *Structure*
- E - *Tree structure*
- R - *Reset camera*
- T - *Show bounding box*

Příloha II - Obsah CD

Přiložené CD obsahuje:

- Procedurální generování stromů.pdf - PDF verze textu této práce.
- Složku Application obsahující:
  - TreeGen - složka s aplikací generátoru stromů pro Windows 10 x64.
  - VC_redist.x64.exe - instalační soubor pro Microsoft Visual C++ 2017 Redistributable (x64).
- Složku Code obsahující:
  - TreeGen - Visual Studio 2017 solution aplikace. Nutné kroky pro její kompilaci je možné nalézt v kapitole 4.2.
  - Doxygen Documentation - HTML dokumentace kódu vytvořená nástrojem DoxyGen.