Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Petr Tovaryš

Modelování a zobrazování rostlin

Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Josef Pelikán

Studijní program: Informatika

Děkuji RNDr. Josefu Pelikánovi za cenné připomínky a rady při tvorbě této práce. Děkuji své rodině a přátelům za jejich podporu a porozumění.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 13. prosince 2004 Petr Tovaryš

Obsah

1 Modelování rostlin v počítačové grafice **7**

1.1 Úvod . . . 7
1.2 Přehled přístupů . . . 8
1.2.1 Metoda billboardů . . . 9
1.2.2 Metoda směrových billboardů . . . 9
1.2.3 Metoda řezů . . . 10
1.2.4 Ručně vytvářené modely . . . 11
1.2.5 Procedurální modely . . . 12
1.2.6 L-systémy . . . 12
1.2.7 Interaktivní modely . . . 15
1.2.8 Komerční systémy . . . 17
1.2.9 Kombinace přístupů . . . 18
1.3 Rose systém . . . 18

2 Popis Rose systému **19**

2.1 Komponenty . . . 19
2.1.1 Idea komponent . . . 19
2.1.2 Modelování rostliny . . . 19
2.1.3 Graf komponent . . . 21
2.1.4 Prototypy . . . 21
2.1.5 Souřadný systém . . . 22
2.1.6 Geometrické spoje a sloty . . . 22
2.1.7 Úrovně komponent . . . 23
2.1.8 Transformace . . . 24
2.2 Parametrický subsystém . . . 24
2.2.1 Parametry a parametrické prostory . . . 24
2.2.2 Typy parametrů . . . 25
2.2.3 Číselné parametry s desetinnou čárkou . . . 26
2.2.4 Správa parametrů . . . 26
2.3 Simulace růstu . . . 27
2.3.1 Simulace a dL-systémy . . . 27
2.3.2 Řízení událostmi . . . 27

3 Generování geometrie **29**

3.1 Základní přehled . . . 29
3.1.1 Kontext . . . 29

3.1.2 Materiály . . . 29
3.1.3 Databáze materiálů . . . 30
3.1.4 Fáze generování geometrie . . . 31
3.1.5 Stem objekt . . . 31
3.1.6 Průchod grafu komponent . . . 32
3.2 Jak fungují moderní karty . . . 32
3.2.1 Grafická rozhraní OpenGL a DirectX . . . 33
3.2.2 Fungování současných grafických karet . . . 33
3.2.3 Velikost přenášených dat . . . 34
3.2.4 Frekvence přesunu dat . . . 35
3.2.5 Množství geometrických primitiv . . . 36
3.3 Generování geometrie pro OpenGL . . . 36
3.3.1 Fragment . . . 38
3.3.2 Stem . . . 38
3.3.3 Uzel . . . 38
3.3.4 GeometryTurtle . . . 39
3.3.5 OpenGLGeometry . . . 40
3.3.6 Dávka . . . 40
3.3.7 Vykreslování dávek v OpenGL . . . 41

4 Model stromu **43**
4.1 Přehled . . . 43
4.2 Model stromu . . . 43
4.2.1 Větev . . . 44
4.2.2 Dceřinné větve . . . 45
4.2.3 Poloměr větví . . . 49
4.2.4 Listy . . . 49
4.2.5 Vývoj v čase . . . 50

5 Aplikace Viewer **52**
5.1 Zobrazení modelu stromu . . . 52
5.2 Animace vývoje modelu stromu . . . 52
5.3 Parametry programu . . . 53
5.4 Využití OpenGL . . . 53
5.5 Animace vlnění větví ve větru . . . 54
5.6 Formát konfiguračního souboru . . . 55
5.7 Formát databáze materiálů . . . 56
5.8 Srovnání . . . 57

6 Dokumentace Rose systému **58**
6.1 Moduly . . . 58
6.2 Jádro systému . . . 59
6.2.1 Komponenty . . . 59
6.2.2 Sloty . . . 60
6.2.3 Simulace vývoje . . . 60
6.2.4 Události . . . 61
6.2.5 Správa materiálů . . . 62

6.3 Parametrický subsystém 62
6.3.1 Parametry 63
6.4 Generování geometrie 64

7 Závěr **66**

Název práce: Modelování a zobrazování rostlin
Autor: Petr Tovaryš
Katedra: Kabinet software a výuky informatiky
Vedoucí diplomové práce: RNDr. Josef Pelikán
e-mail vedoucího: Josef.Pelikan@mff.cuni.cz

Abstrakt: Práce se zabývá návrhem systému pro modelování a zobrazování stromů a rostlin. Důraz je kladen na to, aby vytvořené modely bylo možno co nejefektivněji zobrazovat na současných grafických akcelerátorech. Úvodem jsou shrnuty nejpoužívanější metody pro modelování a zobrazování rostlin. Dále následuje popis vytvořeného systému. Systém je založen na komponentovém přístupu, kdy je rostlina modelována jako skupina propojených komponent reprezentujících jednotlivé části rostliny. To umožňuje nejen dívat se na modelování rostliny jako na skládanku z mnoha samostatných celků, ale díky zakomponování pojmu času a vývoje jako nedílné součástí systému lze vyjít vstříc modelování biologických pochodů, které v rostlinách probíhají. Součástí práce je také aplikace, která umožňuje generovat modely stromů a v reálném čase je zobrazovat pomocí rozhraní OpenGL. Aplikace umožňuje vytvářet plynulé animace vývoje modelovaných stromů.
Klíčová slova: modelování, zobrazování, rostliny, stromy

Title: Modeling and rendering of plants
Author: Petr Tovaryš
Department: Department of Software and Computer Science Education
Supervisor: RNDr. Josef Pelikán
Supervisor's e-mail address: Josef.Pelikan@mff.cuni.cz

Abstract: The thesis proposes the system for tree and plant modeling and graphic representation of generated models. The work consists of the description of the proposed system and an application for modeling and displaying of created models. The thesis is focused on effective representation and display using current graphic accelerators. The system proposed is based on the compound approach where the plant is modeled as interrelated groups representing its particular parts. Such an approach allows apprehending of the modeled plant not only as a mere sum of its parts. The model then allows including of such essential factors as time, development and evolvement of modeled structures which makes it suitable for precise modeling of biological processes. The application included in second part of the thesis is to model the tree in the real time and to display the result using the OpenGL interface. This application generates the data for the tree model and enables smooth animation of modeled trees in the real time.
Keywords: modeling, rendering, plants, trees

Kapitola 1

Modelování rostlin v počítačové grafice

1.1 Úvod

Snahou počítačové grafiky vždy bylo vytváření fotorealistických obrázků nejrůznějších objektů a scén. Od modelování předmětů, živočichů a lidských postav, až po vyobrazení přírodních scén zahrnující rostliny, stromy a keře. Jelikož mnoho přírodních objektů je velmi komplexních a detailních, jejich modelování a zobrazování bylo vždy spojeno s obtížemi jak zpracovat a efektivně vykreslit jejich složitou geometrii. Zvláště v poslední době, kdy je kladen stále větší důraz na co nejkvalitnější a nejrychlejší vykreslování geometrie v interaktivních aplikacích.

Ve většině prostředích, která obklopují lidskou společnost, tvoří flora jednu z dominantních složek vytvářející vzhled a ráz okolí. Rostliny, stromy a keře můžeme vidět kdekoli v našem okolí, nejen v přirozených venkovních prostředích jako jsou lesy či louky, ale také nemalou měrou v našich moderních městech.

Přesto, že se lidé setkávají s rostlinami každý den, kreslení a modelování realistické flory na počítačích bylo dlouhou dobu velkou výzvou a i v dnešní době existuje jen několik vyhovujících modelů.

Základním problémem modelování stromů a rostlin na počítačích je vysoká geometrická a strukturální složitost. Strom může být tvořen ze stovek a tisíců listů a větví. Detailní model jednoho stromu i v dnešní době znamená výzvu, nemluvě o situaci, kdy je potřeba vykreslit pohled na krajinu, kde se můžou nacházet stovky takovýchto stromů.

V době, kdy počítače měly malou kapacitu paměti a malý výkon, bylo základní snahou vůbec nějaké realisticky vypadající stromy či rostliny vymodelovat. Vzhledem k tomu, že rychlé vykreslování nějakého modelu (o více modelech nemluvě) v měřítku několik snímků za sekundu nepřipadalo v té době v úvahu, většina kreslení probíhala v tzv. *offline* módu, kdy se nějakým způsobem vygeneroval určitý model, který poté počítač mohl kreslit i několik hodin.

Později přišel obrovský výkonnostní nárůst související s rozšířením počítačů

do všech odvětví lidské činnosti, umocněný konkurenčním prostředím výrobců procesorů. Díky grafickým kartám, které dnes dokážou vykreslit miliony grafických primitiv během jediné sekundy a podporující mnoho dalších rozšíření, mohou být scény, jejichž vykreslení dříve trvalo hodiny, vykresleny několikrát za sekundu. Celkový trend se označuje jako *real-time* rendering (kreslení v reálném čase). Dnes již není problém provádět v reálném čase efekty jako jsou stíny, mlha, zrcadlové odrazy, kreslení srsti či deformace obrazu, a to vše v desítkách snímku za sekund.

Přes tyto kvality dnešních grafických karet, zobrazování realistických stromů a keřů v reálném čase zůstává těžko dosažitelným cílem. Existuje velice málo programů, které se zaměřují čistě na zobrazení stromů v reálném čase a jsou použitelné pro opravdu interaktivní aplikace, jakými jsou např. počítačové hry či virtuální realita. Jeden strom může obsahovat tisíce listů, což již dnes není problém při vykreslování jediného či několika stromů, ale při větším počtu (stovky či tisíce) ani dnešní hrubá síla grafických karet nestačí, a je nutno vytvořit speciální algoritmy pro tyto účely.

Proto se dnes stále používají ty nejjednodušší metody pro kreslení stromů a flory, a zvětšuje se hlavně počet takovýchto objektů viditelných v jediné scéně. Jedním z problémů při zobrazování komplexních scén v reálném čase je to, aby se využila výpočetní síla moderních grafických karet, což vyžaduje geometrická data v takovém formátu, která vyhovuje požadavkům těchto karet. Proto sebelepší vylepšení kreslení stromů a rostlin může narazit na problém, že vyžaduje data v takovém formátu, který není vhodný nebo příliš náročný pro grafické karty a proto přináší nepřijatelnou degradaci výkonu.

Cílem této práce bylo vytvořit systém pro tvorbu realisticky vypadajících stromu a rostlin s důrazem na to, aby vytvořené modely bylo možno zobrazovat co nejrychleji na současných grafických kartách.

1.2 Přehled přístupů

V posledních několika letech se objevilo mnoho rozličných přístupů zabývajícími se problematikou generování rostlin a stromů. Z obecného hlediska můžeme přístupy rozdělit do dvou hlavních oblastí. První oblast tvoří formální techniky zaměřené na zachycení vnitřní struktury rostlin a na simulaci biologických pochodů, které se v nich odehrávají a určují výslednou podobu rostliny. Autoři se snaží porozumět procesům probíhajícím uvnitř či v okolí rostlin a tyto procesy pak vyjádřit pravidly a podmínkami, které se aplikují při simulaci vývoje modelované rostliny.

Druhou oblast tvoří přístupy zaměřené více celkovou geometrickou podobu modelované rostliny bez vztahu na botanické principy a často se zaměřují jen na generování specifického typu či skupiny podobných rostlin a jejich geometrie. Autoři se nechtějí omezovat reálnými pochody probíhajícími v přírodě, ale jde jim pouze o výslednou vizuální podobu rostliny, ať se už k ní dospělo libovolnou cestou.

Samostatným problémem je pak vygenerovaná data nějakým způsobem vykreslit. Pokud máme celkový model popisující strukturu a složení modelo-

vaného stromu či rostliny, je nutno převést data do formátu vhodného pro vykreslování. V současné době se v počítačové grafice zaměřené na interaktivní zobrazování v reálném čase pracuje nejčastěji s polygonovými geometrickými primitivy. Dále je potřeba přiřadit odpovídající materiály popisující barvy a textury jednotlivých části modelu, a případně spočítat osvětlení a stíny, pokud tyto nejsou zpracovány až při samotném vykreslování. Pokud cílem není interaktivní zobrazování rostlin, je zpracování vygenerované geometrie často jednodušší, protože v dnešní době nejsme limitováni rychlostními ani paměťovými omezeními jako v minulosti.

Při modelování rostlin je také důležité, jak je systém vstřícný k uživatelům, jaké předpoklady a požadavky na ně klade a jaké parametry jim umožňuje měnit. Obecně lze specifikovat geometrické parametry (jako jsou barva či tvar listů, tloušťka kmene a větví, či celkový tvar stromu nebo rostliny), parametry popisující růst modelu (rychlost prodlužování větví nebo množství větví, které může rodičovská větev během života vytvořit), až po parametry přidávající náhodný faktor do simulace.

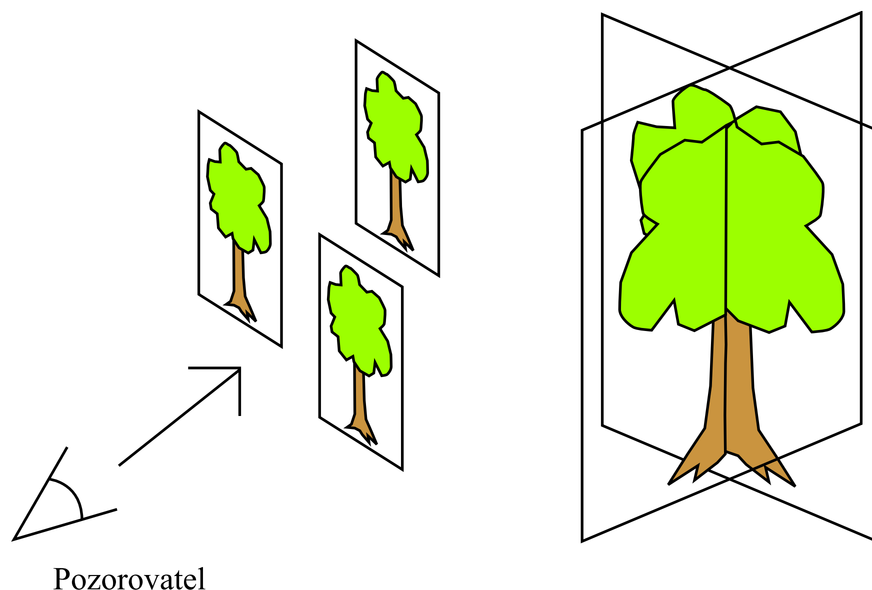
1.2.1 Metoda billboardů

Metoda "billboardů" je jednoduchá metoda používaná po dlouhou dobu pro vykreslování mnoha stromů v reálném čase. Nejjednodušší verze spočívá v namapování obrázku celého stromu na jednoduchý obdélníkový polygon připomínající plakátovou tabuli (odtud název billboard). Silueta je v obrázku ohraničena speciálně označenými body, které jsou při vykreslování ignorovány. Billboard je orientován vždy tak, aby byl natočen k pozorovateli (viz. obrázek 1.1), proto je tato metoda limitována na situace, kdy se pozorovatel pohybuje nízko u země. Čím blíže je pozorovatel stromu, tím více je vidět rozlišení textury a chybějící objem stromu. Nicméně pro zobrazování stromů vzdálených stovky metrů od pozorovatele je to velice dobrá a rychlá metoda, proto se často používá i dnes jako doplněk jiných metod. Tato metoda se v počítačové grafice používá pro mnoho jiných účelů, zvláště pro speciální efekty jako je vykreslování ohně, deště nebo mraků, a proto se lze o ni dočíst v mnoha publikacích a článcích. Jako příklad můžeme uvést článek [1] z knihy, jejímiž autory jsou Akenine-Möller a Heines.

Existuje několik variant, jak může billboard vypadat. Strom nemusí být reprezentován jedním, ale více billboardy, které jsou zkříženy a již se neorientují k pozorovateli. Tím vzniká iluze objemu a strom vypadá i na bližší vzdálenost velmi dobře. Vizuální dojem lze vylepšit přidáním stínu, čímž se zdůrazní spojení stromu se zemí a předejde se dojmu, že strom visí nízko na povrchem.

1.2.2 Metoda směrových billboardů

V této metodě je pro jeden strom vygenerováno množina obrázků představujících strom z různých úhlů pohledu. Při zobrazování je vybrán vždy jeden obrázek podle úhlu pohledu pozorovatele a strom se vykresluje jako standardní billboard. Aby bylo přepínání mezi billboardy co nejméně vidět, je potřeba

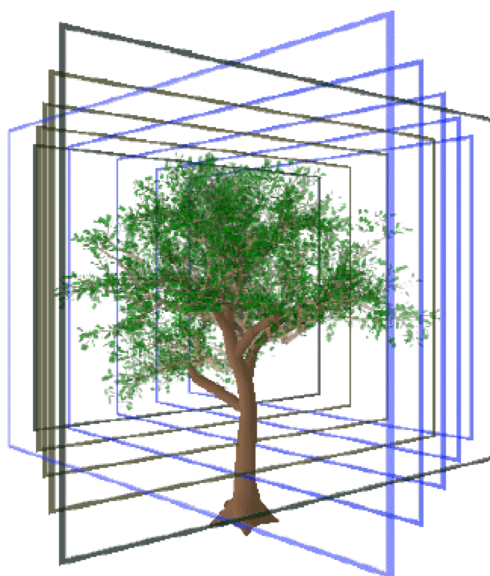


Obrázek 1.1: Metoda billboardů — billboardy jsou natočeny směrem k pozorovateli. Křížení více billboardů pro lepší iluzi objemu.

vytvořit mnoho obrázků stromu z různých stran, což je nejen náročné samo o sobě, ale rostou paměťové nároky na uložení obrázku při běhu programu, díky čemuž místo této metody využívají jiná vylepšení klasické metody billboardů. Rozšířením metody jsou pak tzv. billboardové mraky (*billboard clouds*), které ve své práci prezentoval Décoret a kol. [2], kdy je geometricky složitý model nahrazen skupinou billboardů tak, aby co nejvěrněji aproximovala model z různých úhlů pohledu. Rekonstrukcí obrazu stromu z předpočtených obrázků s uloženou hloubkou pro každý bod se snaží ve své práci Max a Ohsaki [3], metoda je však příliš náročná pro reálné využití v interaktivních aplikacích. Zde je důležité si uvědomit, že přestože kvalita vykreslování stromů a rostlin je důležitá, tvoří typicky jen malou část celého produktu a existující zdroje je nutno sdílet mezi mnoho systémů, které mohou být také velice časově a prostorově náročné.

1.2.3 Metoda řezů

Metodu řezů (v literatuře označována jako *slices*) prezentoval ve své práci Aleks Jakulin [4]. Tato metoda se snaží vylepšit metodu billboardů tím, že strom je rozdělen na skupinu obrázků, reprezentující řezy stromu z různých pohledů. Při vykreslováni jsou zvoleny dvě skupiny podle úhlu pozorovatele a všechny řezy z těchto skupiny jsou vykresleny po sobě. Pro co nejméně viditelné rozdělení na skupiny při pohybu pozorovatele slouží správné nastavení mixování barev při vykreslování řezů. Rozdělení na řezy způsobuje efekt, kdy při pohybu kolmém ke stromu se řezy na obrazovce pohybují různou rychlostí, čímž vzniká dojem objemu stromu. Tento efekt se označuje v literatuře anglickým slovem *parallax*. Tohoto efektu standardní metodou billboardů nelze docílit. Jelikož nepřesnosti



Obrázek 1.2: Rozdělení stromu metodou řezů.

ve tvaru kmene stromu člověk vnímá daleko více než nepřesnosti v zobrazování listů, je výhodné reprezentovat kmen stromu jako geometrii, a řezy použít jen pro zobrazení listí.

Stejně jako metoda směrových billboardů má tato metoda obrovské paměťové nároky na uložení obrázků všech řezů, zvlášť pokud by se povolilo pozorování stromů z výšky, a proto není v praxi příliš používaná.

1.2.4 Ručně vytvářené modely

Interaktivní aplikace, pro které byla metoda billboardů nedostačující a zobrazování plných geometrických modelů stromů příliš náročné, často využívaly ručně modelované stromy. Kmen stromu a jednoduchá hierarchie větví byly typicky modelovány desítkami či stovkami polygonů, skupiny malých větví a listů pak byly připojeny jako malé billboardy. Díky ideálnímu poměru mezi složitostí geometrie a vizuálním dojmem je tento přístup v hojné míře využíván i dnes.

Hlavní nevýhodou je fakt, že celý strom musí vytvářet člověk s grafickým nadáním a schopností využívat modelovací programy. Jelikož jednoduchým způsobem nelze proces zautomatizovat, vymodelování jednoho stromu může trvat dlouhou dobu. Stejně tak jsou komplikované případné úpravy či rozšíření modelu. Současným trendem je generování takovýchto modelů procedurálními metodami. To umožňuje změnou několika parametrů nejenom vytvářet mnoho variací stromů v krátkém čase, ale typicky i škálovat komplexnost výsledné geometrie.

1.2.5 Procedurální modely

Předchozí metody se zabývaly převážně způsobem, jak modely stromů co nejrychleji zobrazovat na dostupných grafických zařízeních, a nezabývaly se způsobem, jak samotné modely vytvářet. Motivací pro procedurální modelování rostlin a stromů bylo nejenom zachycení přírodních procesů a generování realistických a tudíž náročných geometrií, ale také jednoduché vytváření modelů pro interaktivní zobrazování. Procedurální modely typicky umožňují změnou několika málo parametrů změnit jak celkové vzezření modelu, tak za využití stochastickým metod poskytnou mnoho instancí jediného druhu stromu.

De Reffye a kol. [5] vytvořil biologicky motivovaný procedurální model založený na vzniku a zániku pupenů, který umožňoval řídit růst stromu pomocí mnoha parametrů. Model vychází z botanických znalostí struktury stromů: jejich růstu, rozložení větví v prostoru, kdy a za jakých podmínek vznikají listy, pozice květů a plodů a pod. Pokryto je velké množství druhů stromů. Model také implementuje čas, což lze využít k vytváření animací vývoje stromu v různých fázích života. Na základě této práce vznikla komerční technologie *AMAP*.

Jiné práce se zaměřily na vlastní strukturu a geometrii rostlin bez nutnosti být vázány biologickými pravidly. Weber a Penn [6] vytvořili model, ve kterém kladli důraz na celkový vzhled a tvar stromu. Formou textově editovaných parametrů umožňuje specifikovat geometrii pro jednotlivé úrovně stromu, od kmene přes větve až po listy.

1.2.6 L-systémy

Mezi nejznámější systémy pro modelování botanických struktur patří jednoznačně Lindenmayerovy systémy (zkráceně L-systémy), pojmenované po svém zakladateli. Aristid Lindenmayer použil L-systémy v roce 1968 ve své práci o modelování vývoje jednoduchých vícebuněčných organizmů [7]. L-systémy byly později využity pro modelování složitějších struktur, až po modelování stromů a keřů [11].

L-systémy jsou definovány formálním jazykem jako skupina obsahující abecedu symbolů, počáteční řetězec a množinu přepisovacích pravidel. Z matematického hlediska lze na L-systémy nahlížet jako na gramatiky, což umožňuje L-systémy studovat a charakterizovat za pomoci teorie automatů a gramatik. Z biologického hlediska symboly z abecedy reprezentují části rostliny (jako jsou články stonku, pupeny, listy nebo květy), počáteční řetězec (axiom) označuje výchozí stav rostliny a přepisovací pravidla zachycují vývoj částí rostliny po pevných časových úsecích.

Počáteční řetězec se aplikací přepisovacích pravidel neustále mění, což představuje vývoj rostliny v čase. Z výsledného řetězce se poté generuje výstupní geometrie, kdy je každému symbolu přiřazen model reprezentující danou část rostliny. Pro interpretaci výsledného řetězce se vžil přístup založený na využití želvičky z jazyka LOGO. Řetězec je považován za posloupnost příkazů pro tuto želvičku. Průchodem řetězce a postupným vykonáváním příkazů želvička vytváří obrazec odpovídající vygenerovanému řetězci.

Pokud by výstupem L-systémů byl pouze řetězec, výsledným obrazcem by byla pouze lomená čára. Aby bylo možno modelovat rostliny a stromy, byl postup rozšířen o možnost vytvářet obecné stromové struktury v matematickém smyslu slova. Aby nebylo potřeba reprezentovat model datovými strukturami spojenými pomocí ukazatelů, byly zavedeny závorkové 0L-systémy. V řetězci je podřetězec reprezentující část rostliny vyrůstající z hlavní osy uzavřen v hranatých závorkách. Závorky lze aplikovat rekurzivně na části rostliny hlouběji v hierarchii. Při průchodu řetězcem je při levé závorce uložen aktuální stav želvičky na zásobník, při pravé závorce je stav ze zásobníku obnoven.

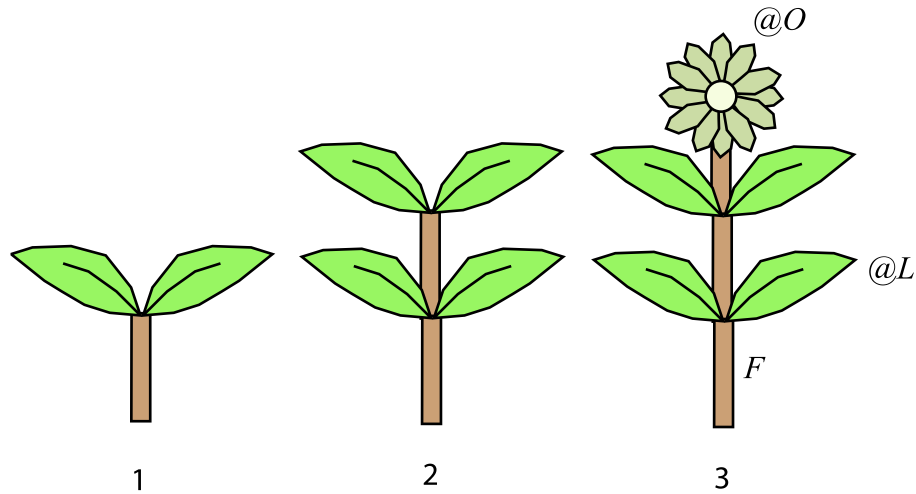
L-systémy se staly předmětem studia pro modelování biologických pochodů a dočkaly se mnoha obměn, které rozšiřují základní možnosti L-systémů, od kontextově závislých L-systémů pro přenos informací mezi částmi rostliny, přes stochastické L-systémy pro začlenění náhodné variability do modelu, až po modelování vlivu okolního prostředí na rostliny.

Stochastické L-systémy rozšiřují L-systémy o náhodné prvky. Tak lze z jediné skupiny pravidel vygenerovat různé variace téže rostliny. Každé pravidlo je ohodnoceno číslem, reprezentující pravděpodobnost aplikace tohoto pravidla. Při výběru následujícího pravidla se pak náhodně vybralo jedno z možných pravidel dle jejich pravděpodobností. Takto lze ovlivnit nejenom délku segmentů či velikosti úhlů mezi větvemi, ale i strukturu celé rostliny.

Pro možnost šíření informací v rostlině vznikly kontextové L-systémy, kdy o aplikaci přepisovacího pravidla nerozhoduje pouze přepisovaný symbol, ale také symboly v jeho okolí (kontext). Pravidla v 1L-systémech obsahují buď pouze levý nebo pravý kontext velikosti jednoho symbolu. 2L-systémy umožňují zadat levý i pravý kontext současně. Zobecněním jsou pak IL-systémy, kde levý kontext představuje slovo délky k a pravý kontext slovo délky l. V praxi se pracuje s L-systémy, kdy v jednom systém jsou povoleny pravidla s různými délkami levých a pravých kontextů a kdy kontextová pravidla mají přednost před nekontextovými. Jelikož mnoho symbolů reprezentuje pouze geometrii a nejsou zajímavé z topologického hlediska, je definována skupina symbolů, které jsou při testování kontextů ignorovány.

Zmíněná rozšíření jsou však stále příliš omezující, neboť nejmenší stavební jednotkou je symbol a pro reprezentaci segmentů různých délek je zapotřebí buď každý segment vyjádřit jako násobek speciálního symbolu, nebo zavést symboly pro všechny možné délky. Z tohoto důvodu byly řetězce rozšířeny o parametry a vznikly parametrické L-systémy. Každý symbol v řetězci může obsahovat množinu parametrů. Na parametry se lze odkazovat jak na levé tak na pravé straně přepisovacích pravidlech. Takto lze vytvořit přepisovací pravidla, která se aplikují jen v případě, že parametry v levém kontextu splňují zadané podmínky. Na druhé straně přepisovacího pravidla lze pak nové hodnoty parametrů vyjádřit v závislosti na hodnotách parametrů z levé strany přepisovacího pravidla. Parametry mohou sloužit nejenom pro výběr a aplikaci přepisovacích pravidel, ale také mohou být reprezentovány při interpretaci řetězce želvičkou. Parametry mohou určovat délku, barvu či úhel.

Zde si uvedeme jednoduchý příklad parametrického systému, jehož interpretací vznikne rostlina na obrázku 1.3:



Obrázek 1.3: Grafická interpretace L-systému (1.1) a jeho vývoje (1.2).

$$\begin{aligned}
\omega :&\quad A(0.0, 0.2)\\
p_1 :&\quad A(s,f) \quad : s < f \rightarrow F[+@L][-@L]A(s+0.1, f)\\
p_2 :&\quad A(s,f) \quad : s \geq f \rightarrow F@O(0.5)
\end{aligned} \tag{1.1}$$

Počáteční řetězec je označen znakem ω a je tvořen symbolem A se dvěma parametry. Vývoj L-systému je pak určen pravidly p_1 a p_2. Pravidlo p_1 lze aplikovat pouze v případě, že hodnota parametru s je menší než hodnota parametru f. V opačném případě se aplikuje pravidlo p_2. Vývoj L-systému pak bude vypadat takto:

$$\begin{aligned}
\mu_0 :&\quad A(0.0, 0.2)\\
\mu_1 :&\quad F[+@L][-@L]A(0.1, 0.2)\\
\mu_2 :&\quad F[+@L][-@L]F[+@L][-@L]A(0.2, 0.2)\\
\mu_3 :&\quad F[+@L][-@L]F[+@L][-@L]F@O(0.5)
\end{aligned} \tag{1.2}$$

V každém kroku je pravidlem p_1 hodnota prvního parametru symbolu A zvětšena o 0.1, dokud nelze aplikovat pravidlo p_2. Jelikož neexistuje žádné pravidlo, které by mělo na levé straně symbol F, systém automaticky aplikuje pravidlo $F \rightarrow F$. Symbol $+$ označuje odklon o 45 stupňů od stonku, symbol $-$ pak odklon opačnou stranu. Symbol F je při interpretaci řetězce nahrazen geometrií odpovídající části stonku. Symbol @ slouží pro vkládání předpřipravené geometrie, $@L$ vloží geometrii listu a $@O$ geometrii květu. Nepovinný parametr symbolu @ určuje hodnotu zvětšení vkládané geometrie.

Pokud chceme simulovat vývoj rostliny a vytvářet animace, narazíme brzy na omezení L-systému, kdy jsou přepisovací pravidla aplikována v diskrétních krocích, což ztěžuje vytváření plynulých animací, podobně jako L-systémy bez

parametrů omezovaly vytváření segmentů různých délek. Pro simulaci plynulého vývoje rostlin představil Prusinkiewicz a kol. dL-systémy [12], které se snaží zachytit růstové funkce jednotlivých částí rostliny a podle toho zjemňovat diskrétní kroky, při kterých jsou aplikována přepisovací pravidla.

L-systémy popisují lokální změny v růstech rostlin pomocí růstových pravidel, což odpovídá tomu, že vznik L-systémů byl původně biologicky motivován. Tento přístup může být intuitivní pro biology, ale z hlediska modelování je zajímavější práce s globálními parametry popisujícími celkový vzhled modelované rostliny. Postupně vznikly programy, které umožňovaly specifikovat parametry simulace graficky, přesto zůstalo kontrolování globálních aspektů obtížné.

Je vidět, že většina uvedených rozšíření L-systémů nepřináší nic neočekávaného a vytvářejí rozumný základ, se kterým lze při modelování rostlin začít pracovat. S aplikací každého rozšíření se však syntaktická forma zápisu L-systémů stává složitější a pro uživatele značně nepřehledná. Vytváření reálně vypadajících modelů a jim odpovídajícím komplexním L-systémům se tak stalo oborem skupiny specialistů, kdy se psaní L-systémů podobá spíše programování než modelování. Uživatel tak většinou dostal hotový model, který mohl modifikovat pomocí vstupních parametrů poskytnutých autorem L-systému.

Zjednodušit práci s L-systémy se pokusil ve své práci Curry [8], kdy je uživateli umožněno modelovat rostliny aplikací genetických algoritmů. Skupina parametrů L-systému je interpretována jako genom (skupina genů). Postupnou kombinací genomů různých rostlin z náhodně vygenerované populace a eliminováních nevyhovujících rostlin je možno vizuálně usměrňovat finální podobu rostliny.

Dalším směrem, kterým se začaly L-systémy vyvíjet, byla interakce systému s okolním prostředím. Radomír Měch ve své práci [9] studuje interakce mezi rostlinami a okolním prostředím. Zakomponováním přírodních pochodů do formalizmu L-systému vzniklo rozšíření L-systémů, které umožňuje výměnu informací mezi modelem rostliny a prostředím, které ji obklopuje.

1.2.7 Interaktivní modely

S postupem času se začaly objevovat systémy, které se snažily dát koncovým uživatelům větší volnost při modelování rostlin či stromů. Objevila se idea modelovat rostliny jako skládanku z komponent. Uživatel zvolí komponenty, nastaví jejich atributy a spojí je do hierarchie, ze které vznikne konečný model rostliny.

Lintermann a Deusenn ve své práci [13] představili systém, který umožňuje jednoduše vytvářet modely rostlin a stromů. Základem je množina komponent. Každá komponenta představuje určitou část rostliny a umožňuje specifikovat její vlastnosti. Uživatel spojuje komponenty do tzv. p-grafu (graf prototypů), což je stromová struktura, kde uzly odpovídají komponentám a hrany představují spoje mezi komponentami.

Systém umožňuje tři druhy napojení komponent. Standardní hrana představuje spojení rodičovské a dceřinné komponenty, kdy potomek je umístěn relativně k rodičovské komponentě. Rekurzivní hrana způsobí duplikaci a na-

pojení podstromu až do zadané hloubky. U rekurzivní kombinace lze specifikovat komponenty, které se umístí po dokončení rekurze jako listy stromu. Poslední typ hrany umožňuje duplikovat zadaný podstrom a vytvořit z něj několik dceřinných stromů zadané komponenty.

Uživatel v grafickém rozhraní zadefinuje komponenty a jejich napojení a vytvoří tak vstupní p-graf. Systém poté pracuje tak, že zpracuje všechny hrany v grafu a provede instanciaci všech komponent, čímž vznikne tzv. i-graf (graf instancí), který již obsahuje pouze standardní hrany. Z i-grafu se pak průchodem vytváří konečná geometrie modelované rostliny.

Systém v základní podobě poskytuje jedenáct základních komponent.

- *Simple* komponenta obsahuje základní sadu atributů, které poskytují všechny ostatní komponenty, definuje transformace dceřinných komponent a umožňuje vkládání základních geometrických primitiv jako jsou koule, kvádry či válce.

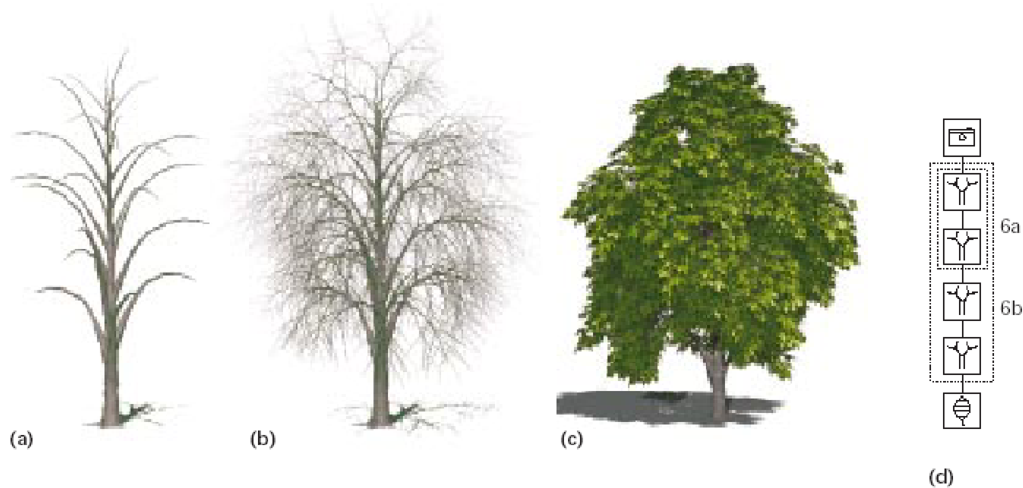
- *Horn* komponenta slouží pro vytváření stonků, větví a kmenů. Je tvořena posloupností geometrických primitiv vkládaných podél křivky nebo relativně k předchozím. Typicky je stonek tvořen posloupností kružnic, které jsou později spojeny polygony dohromady pro vytvoření uzavřeného tělesa.

- *Leaf* komponenta slouží pro definici nejrůznějších druhů listů. Komponenta generuje množinu bodů reprezentujících plochu listu, které jsou později spojeny polygony. Na body lze aplikovat deformace a konturu listu lze zadat křivkou.

- *Tree* komponenta kombinuje jak generování geometrie tak duplikace dceřinných komponent. Podobně jako u komponenty *Horn* je výstupní geometrií větev či kmen. *Tree* komponenta může navíc duplikovat komponenty, které ji následují v p-grafu, jako větve vyrůstající z geometrie kmene. To umožňuje uživateli definovat celý model stromu kaskádovitým nebo rekurzivním napojením těchto komponent. Narozdíl od L-systémů, které pracují s pravidly popisujícími lokální charakter rostliny, poskytuje *Tree* komponenta parametry ovlivňující celkový charakter modelu, jako je úhel odklonu větví, jejich hustota či rozložení podél kmene.

- *Hydra* komponenta umísťuje dceřinné komponenty do hvězdicového tvaru. *Wreath* komponenta umísťuje komponenty do spirály podobně jako rozložení svíček na Vánočním stromku. U obou komponent lze určit počet vytvořených potomků a poloměr kruhu, ve kterém jsou komponenty vytvářeny. Podobně komponenta *Phinball* vytváří dceřinné komponenty na úseku koule, což lze využít pro modelování květů.

- Komponenty *FFD* a *Hyperpatch* umožňují zadávat deformace výstupní geometrie. Pomocí komponenty *World* lze specifikovat parametry charakterizující okolní prostření.



Obrázek 1.4: Modelování stromu skládáním komponent.

Pro specifikaci parametrů komponent lze využít standardní matematické funkce a funkce pracující s náhodnými veličinami. Ve funkcích lze využívat jak hloubku komponenty v grafu tak pořadové číslo iterace, pokud komponenta vznikla jako potomek násobící komponenty.

Na obrázku 1.4 je zobrazeno modelování stromu pomocí sekvence *Tree* komponent. V první části jsou spojeny dvě komponenty pro vytvoření dvou úrovní stromu, kmene a větví z něj vyrůstajících. Kombinací parametrů se dosáhne požadovaného tvaru. V druhé části jsou přidány větve následujících dvou úrovní větví. Ve třetí části je výsledný strom, který vznikl přidáním komponent reprezentující listy stromu. V pravé části je zobrazen odpovídající p-graf, jehož části odpovídají jednotlivým fázím modelování.

Tento přístup se stal základem pro komerční systém *XFrog*. Přestože vytvoření jednoduchého modelu rostliny může být provedeno během okamžiku, vymodelování komplexního a realisticky vypadajícího stromu může podle autorů zkušenému uživateli trvat několik hodin. Přesto je tato doba velice krátká v porovnání s jinými metodami a navíc lze existující modely jednoduše upravovat a měnit.

1.2.8 Komerční systémy

V současné době existuje několik komerčních systémů pro modelování rostlin a stromů se zaměřením na interaktivní aplikace a zobrazování modelů v reálném čase na současných grafických kartách. Většina těchto systémů vychází z nějakého přístupu, který byl veřejně publikován ve vědeckých kruzích, a tento přistup zdokonalili a nasadili do praxe. Vylepšení a implementace těchto přístupů se samozřejmě stala softwarovým tajemstvím.

Jením z nejznámějších komerčních systémů je *SpeedTree* od společnosti Interactive Data Visualization. Systém poskytuje kompletní prostředí pro modelování stromů od začlenění do moderních modelovacích programů až po zob-

razování v reálném čase stovky a tisíců stromů v jedné scéně se zaměřením na interaktivní aplikace, zvláště počítačové hry.

Jiným známým komerčním systémem je *XFrog* od společnosti Greenworks. Systém vychází z práce, kterou publikoval Lintermann a Deusenn [13]. Systém je zaměřen spíše na modelování vysoce detailních realistických stromů a již méně na zobrazování v reálném čase. Strom je modelován uživatelem z komponent skládaných do grafů, ze kterého se generuje konečná geometrie.

Co nejvěrnějším modelováním a vytvářením animací vývoje stromů se zabývá systém *OnyxTREE* od společnosti Onyx Computing. Systém se nezaměřuje na zobrazování modelů v reálném čase, pouze na vytváření detailních modelů pro moderní 3D modelovací aplikace, přesto umožňuje uživatelům měnit růstové parametry graficky a s rychlou odezvou.

De Reffye a kol. [5] vytvořil biologicky motivovaný procedurální model založený na vzniku a zániku pupenů, který umožňoval řídit růst stromu pomocí mnoha parametrů. Na tomto základě vznikla komerční technologie *AMAP*.

1.2.9 Kombinace přístupů

Některé zmíněné přístupy se zaměřují na co nejrychlejší zobrazování modelů stromů, jiné spíše na vlastní modelování. To samozřejmě vede ke kombinaci jednotlivých přístupů. Remolar a kol. [10] představil algoritmus pro zobrazování modelů stromů generovaných systémem *AMAP* v reálném čase. Algoritmus je založen na spojování a nahrazování skupin listů jednodušší geometrií podle vzdálenosti od pozorovatele, díky čemuž lze dosáhnout výrazné snížení množství vykreslovaných polygonů.

1.3 Rose systém

Součástí této práce bylo vytvoření systému umožňujícího modelovat vývoj rostlin a generovat geometrii pro co nejrychlejší zobrazování na současných grafických kartách. Výsledkem je Rose systém a aplikace *Viewer*. Rose systém je obecné programové rozhraní pro modelování rostlin. Systém je založen na komponentách, které se dokáží vyvíjet v čase a spojením do jediného grafu vzniká model celé rostliny. Výsledný graf lze převést do formátu vhodného pro rozhraní poskytovaná současnými grafickými kartami. V Rose systému je implementován model stromu vycházející z práce, kterou publikovat Weber a Penn[6], který umožňuje vytvářet nejen modely různých stromů, ale i vytvářet animaci jejich vývoje. Aplikace *Viewer* dokáže výslednou geometrii zobrazovat v reálném čase pomocí knihovny OpenGL.

Kapitola 2

Popis Rose systému

2.1 Komponenty

2.1.1 Idea komponent

Rose systém je založen na komponentovém přístupu. Komponenty představují samostatné objekty, které se určitým způsobem vyvíjejí a komunikují s ostatními komponentami pomocí signálu a přenosu informací. Modelovaná rostlina je tvořena ze základních komponent, které jsou pospojovány do grafu. Během vývoje rostliny se komponenty vyvíjejí, reagují na signály, vytvářejí nové komponenty či nahrazují samy sebe jinými komponentami.

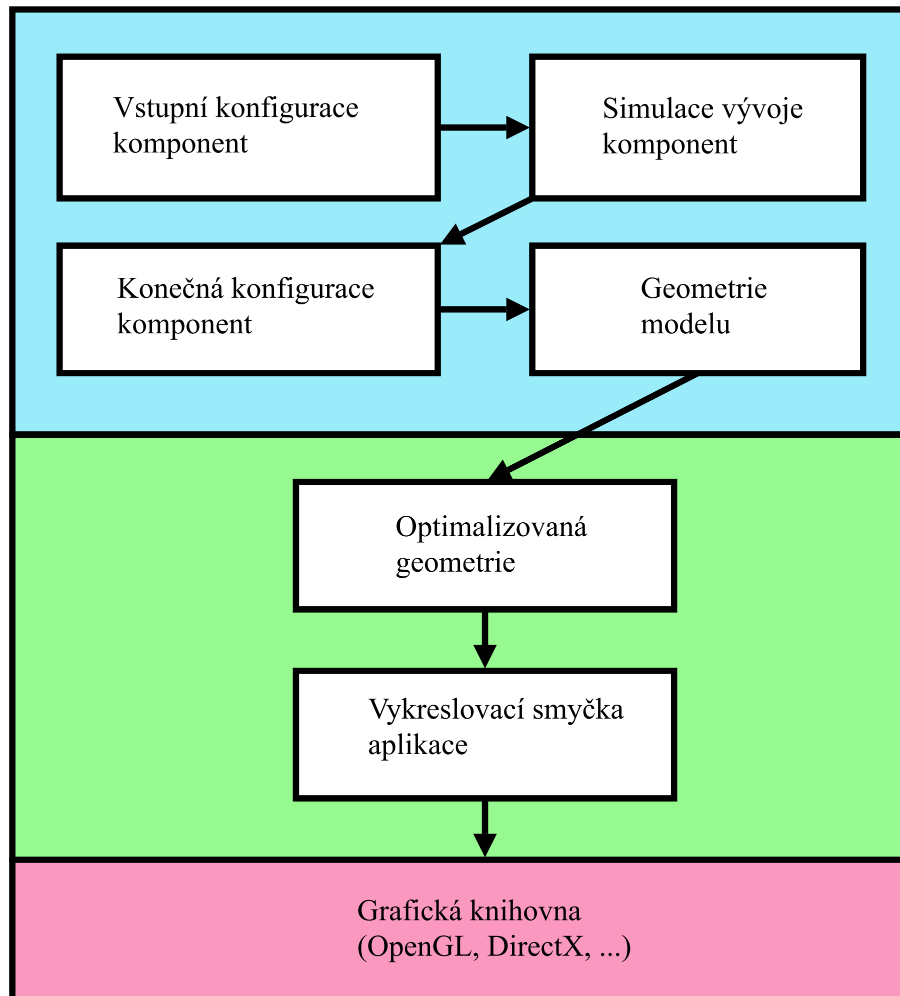
Každá komponenta má sadu atributů, které ovlivňují její vlastnosti a chování. Vstupní atributy jsou atributy, které komponenta očekává při svém vytvoření. Dále má komponenta vnitřní atributy, které udržují její aktuální stav. Při svém vývoji pak komponenta může některé atributy předávat jiným komponentám. Tyto atributy označujeme jako výstupní atributy. Jelikož výstupní atributy představují vstupní atributy jiných komponent, označují se v Rose systému vstupní a výstupní atributy souhrnně jako parametry. Obecný pojem atributy pak necháváme pro označení vnitřních atributů komponenty.

Skupina komponent pak reprezentuje modelovanou rostlinu. Každá komponenta zapouzdřuje data a chování určité části rostliny. Jakmile je vytvořena konečná podoba rostliny, je z komponent vytvořena vlastní geometrie. Ne nutně musí z každé komponenty vzniknout nějaká geometrie, některé komponenty mohou existovat jen jako spojovací články jiných komponent.

2.1.2 Modelování rostliny

Modelování rostliny v Rose systému probíhá v několika fází. Na začátku stojí vstupní graf komponent, který představuje výchozí podobu rostliny. Typicky tento graf obsahuje jedinou komponentu, která postupně během simulace bude vytvářet další a další komponenty a tak postupně vznikne konečná podoba rostliny.

Druhou fází je vývoj vstupního grafu v čase. Každá komponenta se vyvíjí, vznikají nové komponenty, jiné zanikají, mění se parametry a atributy



- Jádro systému nezávislé na platformě.
- Vykreslovací část aplikace závislá na platformě.
- Systémové knihovny.

Obrázek 2.1: Průběh zpravování a vykreslení modelu v Rose systému.

komponent. Jakmile simulace dosáhne cílového času, vývoj komponent se zastaví. Tímto vznikne konečný graf komponent, který představuje modelovanou rostlinu v daném čase.

Třetí fází je převod finálního grafu komponent na geometrii rostliny. Formát geometrie závisí na tom, k jakému účelu bude geometrie použita. Rose systém obsahuje podporu pro generování geometrie do formátu optimalizovaném pro OpenGL a do formátu pro program POV-Ray.

Poslední fází je samotné vykreslování vygenerované geometrie a závisí na použitém programu. Součástí této práce je i aplikace Viewer, která umí zobrazovat vygenerovanou geometrii za využití knihovny OpenGL.

2.1.3 Graf komponent

Komponenty se spojují do skupin. Skupina komponent pak představuje modelovanou rostlinu či její část v určitém časovém okamžiku.

Spoje mezi komponentami mají geometrický charakter, označovat je budeme jako geometrické spoje mezi komponentami. Geometrické spoje slouží k definici grafu komponent, ze kterého se bude generovat geometrie rostliny.

Pokud má komponent A spoj na komponentu B, říkáme, že A je rodič B, resp. B je potomek (nebo dceřinná komponenta) A. Vztah rodič—potomek pak říká, že geometrie dceřinné komponenty je umístěna relativně ke geometrii rodičovské komponentě. Při generování geometrie systém prochází graf komponent definovaný vztahem rodič—potomek a geometrii dceřinné komponenty umísťuje relativně k rodičovské komponentě.

Jelikož v grafu komponent není povolen cyklus, tvoří graf strom v matematickém smyslu slova. Obsahuje tedy kořenovou komponentu, z které vede právě jedna cesta ke každé ze zbylých komponent v grafu.

Komponenty mohou mít samozřejmě i další odkazy (reference) na jiné komponenty. Tyto spoje však Rose systém nijak neinterpretuje a je čistě na logice komponent, jak s nimi budou zacházet.

2.1.4 Prototypy

Vezměme kořenovou komponentu grafu, tedy takovou komponentu, která nemá rodičovskou komponentu. Taková komponenta a její potomci pak vytvářejí spojitý graf komponent. Tento graf lze chápat jako "prototyp" určité části nebo celé rostliny. Pokud bychom vygenerovali geometrii z prototypu (tedy z komponent daného grafu), dostaneme část rostliny, kterou daný prototyp reprezentuje.

V Rose systému může v jeden okamžik existovat libovolné množství komponent a prototypů, limitované pouze dostupnou pamětí. Simulace vývoje probíhá na jednom vybraném prototypu. Během vývoje se tento prototyp postupně mění a vzniká výsledný graf, ze kterého se později bude generovat geometrie rostliny. Využití ostatních prototypů záleží čistě na logice komponent či aplikace. Aplikace může mít v paměti různé prototypy pro několik rostlin

a simulovat jejich vývoj současně. Nebo mohou být tyto prototypy připojeny k rostlině během vývoje a stát se tak součástí simulace.

Komponenta během vývoje může využívat prototypy pro definování nových potomků. Komponenta může daný prototyp napojit jako svého potomka tím, že na něj vytvoří geometrický spoj, nebo může prototyp pouze zduplikovat a vytvořit geometrický spoj pouze na jeho kopii, čímž původní prototyp může být opět využit později.

Využití je pak nasnadě. Například můžeme mít komponentu reprezentující květ, která si drží referenci na prototyp reprezentující okvětní lístek. Komponenta květu pak při simulaci vývoje může vytvářet okvětní lístky duplikací daného prototypu.

Lintermannova interaktivní metoda [13] nepodporuje vývoj rostliny v čase. Uživatel pouze vytvořil graf komponent a systém pak průchodem tohoto grafu a podle typu spoje vytvořil graf instancí komponent, ze kterého se ve výsledku generovala geometrie.

Rose systém naproti tomu nechává vytváření komponent z prototypů na komponentách, které mu pouze oznamují, kde vznikl nový geometrický spoj (nebo kde zanikl již existující). Rose systém podporuje vývoj rostliny v čase, tudíž komponenta může vytvářet libovolné geometrické spoje a nové komponenty v průběhu simulace.

2.1.5 Souřadný systém

Rose systém používá levotočivý souřadný systém. Pokud použijeme analogii lidského pohledu, osa x směřuje doprava, osa y nahoru a osa z dopředu.

Každá komponenta pracuje ve vlastním souřadném systému. Rose systém předpokládá, že hlavní osa, ve které rostlina či komponenta roste, je osa y. Souřadný systém celého modelu rostliny vypadá tedy tak, že osa y směřuje směrem k obloze.

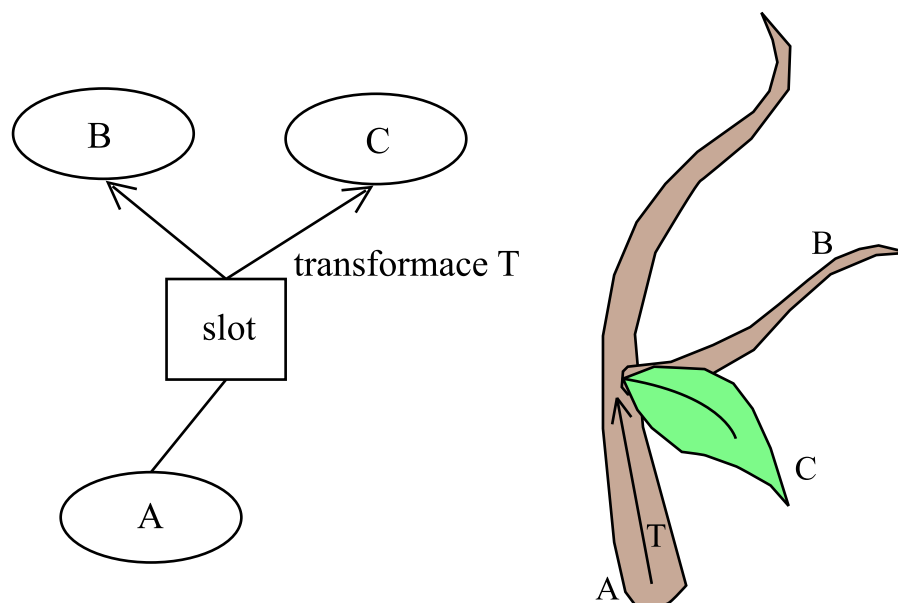
Komponenta specifikuje pozice a transformace vždy relativně ke svému souřadnému systému. Počátek souřadného systému představuje bod, ve kterém geometrie komponenty navazuje na geometrii rodičovské komponenty.

Stejně tak při generování geometrie komponenta specifikuje geometrii relativně ke svému souřadnému systému. O transformaci geometrie do souřadného systému rostliny se stará Rose systém automaticky.

2.1.6 Geometrické spoje a sloty

Geometrický spoj definuje spojení dvou komponent vztahem rodič—potomek. Součástí každého spoje je transformace. Transformace geometrického spoje určuje relativní transformaci souřadného systému dceřinné komponenty vzhledem k rodičovské komponentě, tzn. určuje její pozici a orientaci.

Geometrické spoje jsou implementovány pomocí tzv. *slotů*. Slot je místo v rodičovské komponentě, do kterého se připojuje potomek. Na jeden slot může být napojeno libovolné množství komponent. To je z toho důvodu, že často několik geometrických spojů obsahuje stejnou transformaci, a tedy by



Obrázek 2.2: Napojení komponent pomocí slotů. Na slot rodičovské komponenty A jsou napojeny dvě dceřinné komponenty B a C. V pravé části je geometrická reprezentace grafu. Geometrie komponent B a C je transformována relativně k souřadné soustavě rodičovské komponenty.

bylo plýtváním jak paměti tak procesorovým časem mít pro každou dceřinnou komponentu samostatnou transformaci. Slot tedy obsahuje jednu transformaci, společnou pro všechny komponenty napojené na tento slot.

Každá komponenta může mít libovolný počet slotů. Rose systém rezervuje dva sloty se speciálním významem, které jsou přítomny v každé komponentě. Jedná se o *primární* a *bázový* slot.

Primární slot je slot, který odpovídá intuitivnímu následovníků dané komponenty, tedy místo, kam fyzicky pokračuje geometrie dané komponenty. Například u komponenty reprezentující část kmene stromu je primární slot místo, kde pokračuje zbytek kmene.

Bázový slot je slot s prázdnou (resp. identickou) transformací, tzn. komponenty napojené na tento slot mají systém souřadnic identický jako rodičovská komponenta.

2.1.7 Úrovně komponent

Pokud se podíváme na nějaký přírodní strom, můžeme jeho strukturu větví intuitivně rozdělit na úrovně: nultá úroveň odpovídá kmenu stromu, první úroveň odpovídá větvím vyrůstajícím z tohoto kmene, druhá úroveň odpovídá větvím vyrůstajícím z větví první úrovně, a tak můžeme postupovat dále. U běžných stromů v přírodě bývá úroveň větví v rozmezí 4 až 6. Při modelování stromů na počítačích často vystačuje pracovat s úrovněmi 3 a 4. Při interaktivním zobrazování stromů se často zobrazuje pouze kmen či větve úrovně 1, zbylé úrovně se modelují jiným způsobem (např. texturami a metodou billboardů).

Každá komponenta v Rose systému má přiřazeno číslo odpovídající její úrovni. Typicky budou mít komponenty tvořící jednu větev stejnou úroveň a jejich dceřinné komponenty úroveň o jedna vyšší. Tomu odpovídá to, že komponenty napojené na primární slot mají úroveň stejnou jako komponenta, která vlastní primární slot, komponenty napojené na ostatní sloty pak úroveň o jedna vyšší. Rose systém však na dodržování číslování úrovní netrvá, pro něj je to jen číslo, pomocí kterého je možno komponenty konfigurovat a jehož interpretace závisí na vlastní logice každé komponenty.

2.1.8 Transformace

Transformace v geometrických spojích určuje umístění souřadného systému dceřinné komponenty relativně k souřadnému systému rodičovské komponenty. Rose systém používá pro reprezentaci transformací matice velikosti 4×4, jak je obvyklé v počítačové grafice. Transformace tedy může reprezentovat standardní transformace jako je posun, rotace, zvětšení, zmenšení, a násobením matic lze libovolné transformace skládat.

Transformace v geometrickém spoji transformuje celý podstrom dceřinné komponenty. Tuto transformaci smí měnit pouze rodičovská komponenta, tedy komponenta, která vlastní slot, ve kterém je transformace uložena.

V každém slotu je uložena nejenom příslušná transformační matice, ale také matice odpovídající inverzní transformaci, což umožňuje jednoduše provádět převody mezi souřadnými systémy jednotlivých komponent.

Reprezentace transformací pomocí matic umožňuje, aby systém při generování geometrie pro nějakou komponentu vytvořil jedinou matici vynásobením všech matic od kořene stromu až k dané komponentě, a jednotlivá geometrická primitiva generovaná komponentou transformoval touto jedinou maticí.

2.2 Parametrický subsystém

Rose systém poskytuje komponentám rozhraní pro práci s parametry. Parametry mohou sloužit jak pro konfiguraci samotných komponent, tak pro přenos informací od rodičovských k dceřinným komponentám.

2.2.1 Parametry a parametrické prostory

Každý parametr má typ a jemu odpovídající hodnotu. Parametr je identifikován jménem. Jména parametrů nejsou unikátní v rámci celého systému, ale v rámci tzv. parametrických prostorů (*ParameterSpace*). Parametrický prostor je úložný prostor pro skupinu parametrů, který poskytuje základní rozhraní pro práci s těmito parametry, jako je vyhledávání parametrů podle jména a registrace nových parametrů.

Každý parametr je jednoznačně identifikován svým jménem, typem a parametrickým prostorem, ve kterém se nachází. Při práci s parametrem je jeho typ a jméno určeno jednoznačně z logiky jeho použití a konvencí definování

jmen tak, aby na správný parametr přistupoval jak ten objekt, který jej modifikuje, tak ten objekt, který data z parametru čte. Poslední položkou, kterou je potřeba určit při hledání parametru, je nalezení parametrického prostoru, ve kterém se bude parametr hledat.

Rose systém definuje několik sdílených parametrických prostorů, které mohou být využívány všemi komponentami, a další parametrické prostoru jsou definované v různých částech grafu komponent.

- Globální parametrický prostor je prostor parametrů společný pro celou simulaci a sdílený všemi komponentami. Globální parametrický prostor může obsahovat parametry jako jsou celková velikost rostliny pro škálování, dobu simulace vývoje, maximální úroveň komponent v grafu či parametry pro výchozí nastavení materiálů.

- Pro každou úroveň komponent je definován samostatný parametrický prostor. Při hledání parametrů pak Rose systém používá číslo úrovně dané komponenty pro vybrání konkrétního parametrického prostoru z tohoto seznamu. Parametrické prostory pro jednotlivé úrovně mohou sloužit pro zadávání parametrů jako jsou tloušťka větví v n-té úrovně, poměr délky větví k délce rodičovské větve, zakřivení těchto větví a pod.

- Samostatný parametrický prostor je také vytvářen pro každý slot. Komponenta při hledání parametru může říci, že chce zadaný parametr z nejbližšího slotu, který je na cestě ke kořenu grafu. Tyto parametrické prostory slouží typicky pro předávání informací od rodičovských k dceřinným komponentám, jako jsou například délka rodičovské větve, pořadové číslo dceřinné komponenty či poloměr rodičovské komponenty v místě vzniku dceřinné komponenty.

Při simulaci vývoje mohou komponenty samozřejmě hodnoty parametrů aktualizovat, čímž se nová hodnota automaticky dostane ke komponentám, které z tohoto parametru budou opět číst. Například pokud se během vývoje větev prodlužuje, může aktualizovat parametr, který obsahuje aktuální délku této větve a tak tuto informaci jednoduše distribuovat pro dceřinné komponenty. Zde je vidět, že rodičovská komponenta neklade žádné nároky na rozhraní dceřinných komponent. Komponenta jednoduše poskytuje skupinu určitých parametrů a jiné komponenty tyto parametry mohou, ale nemusí, využívat. Jiné komponenty naproti tomu mohou některé parametry vyžadovat a je na tom, kdo komponenty spravuje, aby zaručil správnou kompatibilitu napojení.

2.2.2 Typy parametrů

Rose systém implementuje tři druhy parametrů:

- Celočíselné parametry.

  Základní typ číselného parametru, který obsahuje 32-bitovou celočíselnou hodnotu. Typicky využíván pro konfiguraci množství jako je počet větví, listů či počet snímků animace při simulaci vývoje.

- Číselné parametry s desetinnou čárkou.

  Druhý základní typ číselného parametru pro uložení desetinných čísel. Využíván od specifikace délky či poloměru větví, přes parametry zakřivení větví až po celkovou dobu života stromu. Jelikož při modelování rostlin se často pracuje s náhodnými hodnotami ze zadaných rozsahů, obsahují celočíselné parametry kromě vlastní hodnoty i rozsah v jakém se hodnota může náhodně měnit. Více viz. kapitola 2.2.3.

- Textové parametry.

  Poslední typ parametru umožňuje ukládat libovolný textový řetězec. Může sloužit pro zadávání materiálů a textur či specifikaci výčtových typů.

2.2.3 Číselné parametry s desetinnou čárkou

Při modelování rostlin se často pracuje s náhodnými hodnotami, aby bylo možno z jediného popisu modelu vygenerovat různě vypadající instance rostliny. Proto jsou číselné parametry s desetinnou čárkou rozšířeny o dvě funkce.

Každý číselný parametr obsahuje kromě vlastní hodnoty i odchylku, která určuje v jakém rozmezí se hodnota může pohybovat. Odchylka je typicky kladná hodnota určující maximální velikost vychýlení hodnoty parametru od iniciální hodnoty. Záporné hodnoty odchylky jsou přípustné a mohou být využity pro speciální mody, kdy jsou hodnoty parametru a odchylky interpretovány jiným způsobem.

Druhou funkcí, kterou každý číselný parametr obsahuje, je tzv. pevná náhodná hodnota. Pevná náhodná hodnota je náhodná hodnota, která je vygenerovaná z iniciální hodnoty parametru a jeho odchylky při registraci parametru v parametrickém prostoru. Poté se její hodnota nikdy nemění. Všechny dotazy na pevnou náhodnou hodnotu parametru z konkrétního parametrického prostoru tedy vracejí stejné číslo. To umožňuje, aby se různé instance modelované rostliny lišily při opětovném odsimulování vývoje, a přitom aby všechny komponenty pracovaly se stejnou hodnotou v rámci jedné simulace.

2.2.4 Správa parametrů

O celkovou správu parametrů a parametrických prostorů se stará objekt *ParameterManager*. Udržuje globální parametrický prostor, parametrické prostory pro jednotlivé úrovně komponent a spravuje mapování jmen parametrů.

ParameterManager umožňuje načíst konfiguraci globálních a úrovňových parametrů z konfiguračního souboru. Konfigurační soubor je textový soubor, který v blocích obsahuje definice parametrů, jejich jmen, hodnot a u číselných parametrů i odchylek. Takto lze jednoduše konfigurovat celý systém.

2.3 Simulace růstu

2.3.1 Simulace a dL-systémy

U standardní verze L-systémů probíhá simulace vývoje po pevných diskrétních krocích, bez explicitního vztahu k době života modelované rostliny. To je důsledek principu, na jakém L-systémy fungují, tedy řetězec znaků, který se mění aplikací přepisovacích pravidel. To sice umožňuje dívat se na L-systémy jako na gramatiku s množinou přepisovacích pravidel, kterou se vývoj rostliny redukuje na jednoduché manipulace se znaky, ale zároveň to limituje možnosti animovat plynule vývoj modelovaných rostlin.

Jako rozšíření základní verze L-systémů o plynulý vývoj rostlin vznikla varianta dL-systémy, kterou publikoval Prusinkiewicz a kol. [12]. dL-systémy se snaží o spojitost při vývoji tím způsobem, že přepisovací pravidla mohou obsahovat změny parametrů vyjádřených diferenciálními rovnicemi. Aplikace pravidla může být vázána na okamžik, kdy zadaný parametr překročí svůj obor přípustných hodnot.

Simulace u dL-systémů probíhá způsobem, kdy je zvolen jistý diskrétní krok. V zadaném intervalu se vyhodnotí změny parametrů zadanými diferenciálními rovnicemi a zjišťuje se, zda v daném intervalu překročil některý z parametrů svůj obor přípustných hodnot. Pokud ano, analýzou a dělením intervalu se nalezne okamžik, kdy k události došlo a v daném časovém okamžiku se aplikuje odpovídající přepisovací pravidlo.

dL-systémy stojí a padají na diferenciálních rovnicích, které je nutno neustále numericky řešit, a na způsobu, jakým se provádí simulace a testování podmínek aplikací pravidel. Ve zmíněné publikaci [12], která se zabývá dL-systémy, jsou použity pouze jednoduché diferenciální rovnice a je zmíněno, že pro složitější rovnice by bylo nutno použít obecné numerické metody pro jejich řešení.

Když jsem rozmýšlel fungování simulaci vývoje v Rose systému, bylo mým cílem začlenit čas a plynulý vývoj intuitivním způsobem přímo do systému. Přístup dL-systému se nezdál příliš vhodný jak pro svou výpočetní náročnost, neintuitivnost popisu vývoje komponent v čase, tak nevelkým přínosem pro uživatele systému.

Mnoho pochodů v rostlinách, od prodlužování délky stonku, přes zvětšování velikosti listu až po rozevírání okvětních lístků, si lze jednoduše představit explicitním vyjádřením v čase. Takovéto vyjádření můžeme zadat nejen matematicky, ale i grafem poskytnutým uživatelem systému.

2.3.2 Řízení událostmi

Vývoj komponent v Rose systému je řízen událostmi. Událost je nějaká akce naplánována na určitý časový okamžik vývoje rostliny. Události mohou být plánovány na libovolný čas libovolné komponentě. Když simulace dosáhne zadaného okamžiku, je událost komponentě doručena a komponenta může libovolným způsobem reagovat.

Při simulaci systém prochází naplánované události a oznamuje je odpovídajícím komponentám. Jakmile jsou zpracovány všechny události pro jeden okamžik, čas simulace poskočí na čas nejbližší naplánované události. Simulace tedy probíhá v diskrétních krocích, ale mezi dvěma kroku může uplynout libovolné množství času.

Události představují klíčové okamžiky ve vývoji rostliny a můžeme je přirovnat k aplikaci nějakého přepisovacího pravidla u L-systémů. Pokud je potřeba získat hodnotu nějakého parametru mezi dvěma simulačními kroky, použije se interpolace. Takto lze vygenerovat geometrii rostliny v libovolném okamžiku simulace.

Jako příklad uveďme článek stonku, který se prodlužuje v čase (kde délka stonku se řídí dle kubické funkce) a při určité délce se stonek rozdvojí. Takováto komponenta naplánuje událost "rozdvojit se" na čas, kdy dosáhne požadované velikosti. Pokud by komponenta potřebuje testovat nějakou podmínku v diskrétních krocích podobně jako u dL-systému, může to provést opět jako opakované plánovaní jediné události.

Cílem je počítat správné hodnoty atributů komponenty až když je to potřeba. Komponenty mohou počítat své atributy nebo pozice potomků podle nějakých spojitých funkcí, které je zbytečné (nebo příliš náročné) vyhodnocovat v každém simulačním kroku. To vyžaduje způsob, aby komponenty dokázaly pro zadaný čas aktualizovat svůj stav.

Aktualizaci komponenty můžeme rozdělit do dvou částí. První je aktualizace atributů dané komponenty, které jsou potřeba pro správné generování geometrie nebo pro zjištění aktuálního stavu komponenty, např. délka komponenty, barva a pod. Druhou části aktualizace je aktualizace slotů komponenty, tedy správné nastavené transformací pro dceřinné komponenty. Přestože pod pojmem aktualizace máme na mysli obě části, při implementaci je často potřeba pouze jedna část aktualizace, díky čemuž lze ušetřit výpočetní čas.

Pro zjištění stavu modelované rostliny pro konkrétní čas systém odsimuluje události až do zadaného okamžiku a poté provede aktualizaci všech komponent. Výsledný graf komponent reprezentuje aktuální stav rostliny pro zadaný čas.

Systém rozlišuje dva druhy času, se kterým mohou komponenty pracovat. *Globální čas* udržuje dobu vývoje celé rostliny od počátku simulace. Vedle globálního času je pro každou komponentu udržován *lokální čas*, který reprezentuje dobu života této komponenty a počítá se od jejího vzniku.

Kapitola 3

Generování geometrie

3.1 Základní přehled

Základní princip generování geometrie je totožný jako princip interpretace řetězce L-systému želví grafikou, pouze v souladu s komponentovým přístupem, kdy komponenta zapouzdřuje data a chování části rostliny, je generování geometrie v režii každé komponenty.

Simulací vývoje vstupního prototypu až do zadaného časového okamžiku a úplnou aktualizací všech komponent vznikne finální graf. Průchodem tohoto grafu pak vzniká výsledná geometrie rostliny.

3.1.1 Kontext

Základem při zpracování geometrie je pojem kontext. Kontext odpovídá želvičce v L-systémech a jedná se o objekt, který udržuje aktuální stav souřadného systému aktuálně zpracovávané komponenty a další atributy generované geometrie (jako je např. barva či textura), včetně geometrie samotné. Průchodem grafu od kořene k listům (podobně jako průchod řetězce L-systému želvičkou) se kontext modifikuje podle transformací jednotlivých komponent a každá komponenta je požádána o specifikaci své geometrie pro aktuální stav.

Pokud se při průchodu grafem narazí na komponentu se dvěma a více potomky, je nutno před zpracováním každého potomka uložit aktuální kontext na globální zásobník kontextů, aby při návratu z potomka mohl být obnoven původní kontext i pro ostatní potomky dané komponenty.

Jelikož komponenty jsou navzájem spojovány přes sloty, z důvodů optimalizace se aktuální transformace nedrží v každém kontextu, ale vedle zásobníků kontextů existuje speciální zásobník pro transformace. To umožňuje eliminovat nadbytečné ukládání transformací pro potomky jedné komponenty.

3.1.2 Materiály

Při generování geometrie je potřeba mít možnost specifikovat způsob, jakým se bude geometrie vykreslovat. K tomuto účelu slouží v Rose systému materiály a databáze materiálů.

Materiál je struktura, která popisuje geometrické vlastnosti skupiny polygonů. Atributy materiálu byly zvoleny tak, aby odpovídaly základním vlastnostem, které lze specifikovat při vykreslování polygonů na současných grafických kartách pro rozhraní OpenGL a DirectX. Atributy materiálu tvoří:

- Barva.

  Barva určuje základní barvu pro všechny polygony v dané skupině. Komponenty mohou později explicitně předefinovat barvu pro každý vrchol kteréhokoli polygonu.

- Textura.

  Textura identifikuje obrázek, který se použije pro texturování polygonů v dané skupině.

- Příznak billboard.

  Materiál s tímto příznakem způsobí, že polygony v dané skupině jsou vykreslovány tak, že jejich přední strana je otočena vždy ke kameře. Tato metoda označována jako *billboarding* je využívána pro kreslení listů na stromech.

- Příznak pro testování alfa kanálu.

  Pokud je tento příznak nastaven, při vykreslování se interpretuje alfa kanál textury tak, že všechny pixely, které mají hodnotu alfy menší než zadaný práh, nejsou vykreslovány. Tato metoda se používá pro vykreslování listů, kdy jediná textura je použita pro vykreslení několika listů dohromady. Alfa kanál pak určuje tvar a ohraničení jednotlivých listů v textuře.

- Příznaky pro oboustranné vykreslování.

  Tyto příznaky určují, zda se mají polygony vykreslovat i když nejsou orientovány přední stranou ke kameře. Tento způsob je potřeba, pokud jsou listy modelovány jako jednoduché polygony bez objemu s průhlednou texturou.

3.1.3 Databáze materiálů

Rose systém udržuje informace o materiálech v databázi materiálů. Každý existující materiál je zaregistrován v databázi pod jednoznačným jménem a identifikátorem. Komponenty pracují s materiály pomocí jmen. Materiály, které daná komponenta využívá, lze takto konfigurovat pomoci textových parametrů. Díky tomu lze jednoduše změnit vizuální vlastnosti geometrie, jako je změna textury listů či kůry, bez nutnosti jakkoli modifikovat komponentu nebo vygenerovanou geometrii.

Databáze materiálů je načítána z textového souboru jednoduchého formátu, který umožňuje pohodlně měnit všechny atributy materiálů či vytvářet materiály nové.

3.1.4 Fáze generování geometrie

Generování geometrie z finálního grafu komponent probíhá ve dvou fázích:

- Generování geometrie do obecného formátu z komponent.

  V této fázi systém prochází graf komponent a každá komponenta je požádána o vygenerování geometrie. Každá komponenta dostane referenci na objekt *Turtle* (jméno želvička bylo zvoleno jako analogie pro v literatuře zavedený pojem v L-systémech, kdy želvička prochází při generování geometrie výsledný řetězec modelovaného L-systému).

  Turtle objekt poskytuje obecné rozhraní pro zadávání geometrie, udržuje aktuální kontext podle průchodu grafem a aktuální materiál. Může existovat libovolné množství implementací *Turtle* objektů podle toho, pro jakou platformu je výsledná geometrie určena.

- Konverze obecného formátu do formátu pro cílovou platformu.

  Jakmile jsou průchodem grafu zpracovány všechny komponenty, jsou získána data převedena do formátu pro cílovou platformu. Rose systém implementuje dva typy objektu *Turtle*, jeden pro generování geometrie optimalizovaný pro zobrazování v reálném čase pomocí knihovny OpenGL, druhý pro generování textového popisu geometrie pro program POV-Ray.

Vložení obecného formátu mezi vstupní a výstupní data je častá programová technika (využívaná např. v kompilátorech), která má mnoho výhod.

Obecný formát umožňuje implementovat různé výstupné formáty pro různé knihovny nebo programy bez nutnosti jakýchkoli změn v komponentovém systému a opačně, a zakrývá technické detaily v implementacích formátů.

Obecný formát může navíc poskytovat komponentám intuitivní rozhraní pro speciální konstrukce, které by bylo složité generovat přímo jako polygonovou geometrii.

3.1.5 Stem objekt

Pro jednoduché modelování stonků a větví obsahuje Rose systém speciální *Stem* objekt. *Stem* objekt modeluje stonek pomocí posloupnosti N bodů. Každý bod má pozici, orientaci a poloměr. *Stem* při generování polygonové geometrie vloží do každého bodu kružnici o daném poloměru a natočí ji dle zadané orientace. Kružnice je pak spojena polygony s kružnicí předchozího bodu. *Stem* objekt se automaticky stará o generování texturových souřadnic, o množství polygonů mezi dvěma kružnicemi a správné napojení jednotlivých polygonů.

Komponenty tedy pouze přidávají nové body *Stem* objektu bez toho, aby se musely starat o to, jak bude polygonová geometrie ve výsledku vypadat. Komponenty zadávají pozice a orientace bodů *Stem* objektu relativně vůči svému lokálnímu souřadnému systému. Pokud jsou body jednoho *Stem* objektu definovány více komponentami, *Stem* objekt automaticky zařídí správnou transformaci jednotlivých bodů.

3.1.6 Průchod grafu komponent

Nyní popíšeme postup, jakým Rose systém prochází graf komponent při generování geometrie. Postup začíná na kořenové komponentě grafu, ze kterého se geometrie generuje. Bodově můžeme tento postup zapsat rekurzivním vyjádřením takto:

1. Vygeneruj geometrii komponenty.
2. Pokud jsou všechny sloty komponenty prázdné, tzn. komponenta nemá žádné dceřinné komponenty, tak proces končí (návrat o jednu rekurzivní úroveň zpět). V opačném případě pro každý slot pokračuj následujícími body.
3. Aplikuj transformaci souřadného systému uloženou ve slotu na aktuální kontext.
4. Vyzvedni nezpracovanou komponentu napojenou na daný slot.
5. Ulož aktuální kontext na zásobník kontextů.
6. Rekurzivně vygeneruj geometrii zvolené komponenty pokračováním na bod 1.
7. Obnov aktuální kontext ze zásobníku kontextů.
8. Pokračuj na bod 3. dokud nejsou zpracovány všechny komponenty napojené na tento slot.

Na proces lze ihned aplikovat dvě jednoduché optimalizace:

- Uložení a obnovení kontextu není nutno provádět, pokud se zpracovává poslední komponenta posledního slotu, neboť tento kontext již nebude žádnou komponentou využit.
- Místo neustálého ukládání kontextu pro komponenty jednoho slotu stačí uložit kontext pouze jednou a poznamenat si, kolik uložení reprezentuje. Při obnovování není kontext z vrcholu zásobníku vymazán, dokud počet obnovení neodpovídá poznamenanému počtu. Tímto lze šetřit počet kopírování aktuálního kontextu na zásobník, neboť kontext může obsahovat velké množství dat.

3.2 Jak fungují moderní karty

Při implementaci Rose systému jsem se zaměřil na generování geometrie do formátu, který by ji umožňoval zobrazovat v reálném čase na současných grafických kartách. Aby bylo možno zobrazovat geometricky složité modely v reálném čase co nejrychleji, je zapotřebí navrhnout celý zobrazovací systém na míru možnostem a požadavkům současných grafických karet a rozhraním, která poskytují. Proto se nyní podívejme, jakým způsobem je potřeba připravit geometrická data, aby byla zpracována grafickými kartami co nejefektivněji.

3.2.1 Grafická rozhraní OpenGL a DirectX

Pro zobrazování geometrie v reálném čase se v současné době staly standardem dvě knihovny, které poskytují programům rozhraní pro přístup a využití hardwarově akcelerované grafiky. Těmito knihovnami jsou OpenGL a DirectX (resp. Direct3D).

OpenGL jakožto multiplatformní knihovna se stala standardem v mnoha oblastech počítačové grafiky a umožňuje pracovat pomocí jednotného rozhraní s grafickými kartami na různých platformách (Windows, Linux, MacOS). Oproti tomu DirectX je knihovna pouze pro platformu Windows a rozhraní pro 3D grafiku tvoří její podčást.

Obě knihovny, OpenGL a DirectX, vycházejí z různých konceptů a ideí a liší se v mnoha oblastech (což také vede k nekonečným debatám, v čem je která knihovna lepší než druhá). Nicméně pokud člověk pronikne pod slupku syntaktických rozdílů obou rozhraní, tak zjistí, že se knihovny v základních principech přístupu a komunikace s grafickými kartami velice podobají. Není se ani čemu divit, knihovny běží nad stejnými grafickými kartami a ty v důsledku poskytují stejnou funkcionalitu.

Z tohoto důvodu není zajímavé se zabývat rozlišnostmi mezi těmito knihovnami, ale principy, na jakých funguje komunikace se současnými grafickými kartami. Zobrazovač modelů, který je součástí této práce, je napsán pro knihovnu OpenGL, nicméně základní jádro systému Rose je na grafické knihovně nezávislé.

3.2.2 Fungování současných grafických karet

Podívejme se nyní, jak v současné době funguje komunikace mezi procesorem (programem) a grafickým hardwarem na architektuře osobních počítačů PC.

Grafická karta vyžaduje pro vykreslení scény data dvojího druhu: vlastní data popisující geometrii vykreslovaného modelu (geometrická data), a data specifikující jakým způsobem se má zadaná geometrie vykreslit (konfigurační data), typicky nastavení světel, materiálů, textury a pod.

Procesor a grafická karta pracují každý ve vlastním paměťovém prostoru. Operace a přesuny dat jsou extrémně rychlé uvnitř každého paměťového prostoru, úzké hrdlo systémů však tvoří až tok dat mezi těmito paměťovými prostory, tedy přesun dat mezi pamětí počítače (data generovaná programem) a pamětí grafické karty. Přesun dat mezi pamětmi je několikanásobně pomalejší než přesuny dat uvnitř paměti. Z tohoto důvodu rychlost či pomalost celého vykreslování je určena těmito faktory:

1. Velikost dat, které se musí přesunout z operační paměti procesoru do pracovní paměti grafické karty.

2. Frekvence, s jakou jsou data do grafické karty přesouvána.

3. Množství geometrických primitiv, které musí grafická karta zpracovat a vykreslit.

Nyní se můžeme podívat na tyto body jednotlivě a jejich analýzou určit požadavky na fungování celého vykreslování, abychom dosáhli maximální efektivnosti.

3.2.3 Velikost přenášených dat

Množství dat, které je potřeba přesunout do grafické karty, závisí na počtu geometrických primitiv, ze kterých se kreslený model skládá. Základními geometrickými primitivy jsou trojúhelníky (dále polygony). Každý polygon je popsán třemi vrcholy (vertexy). Každý vrchol obsahuje pozici, normálový vektor (pokud je model vykreslován s využitím světel), barvu a texturové souřadnice v textuře.

Prvotním způsobem jak minimalizovat množství dat je redukce počtu polygonů, ze kterých se model skládá. Zde je potřeba zpřístupnit uživateli systému možnost, jak konfigurovat složitost generovaných modelů. Výsledný model lze také zjednodušit až po vytvoření pomocí algoritmů na redukci počtu polygonů. Zde je ale nutno poznamenat, že většina algoritmů na redukci složitosti modelů byla vytvořena pro "spojité" modely, tedy takové, které mají spojitý povrch, jednotlivé polygony na sebe navazují a lze dobře definovat pojem "uvnitř" a "vně" modelu. U modelů stromů, které jsou z velké části tvořeny malými skupinkami polygonů představující listy, lze tyto algoritmy jen ztěží aplikovat.

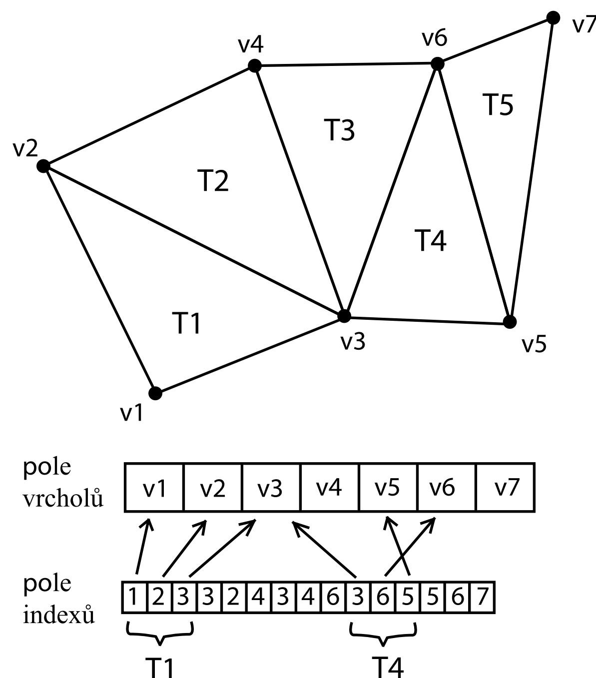
Další možností pro zmenšení objemu dat je sdílení dat mezi polygony. Pokud polygon sousedí s jiným polygonem, obsahují oba polygony dva společné vrcholy, jejichž data lze tedy sdílet a reprezentovat je v paměti pouze jednou. Čím více polygonů spolu sousedí (a tvoří tedy souvislou plochu), tím větší úsporu sdílení dat přinese. Data vrcholů jsou ukládána do pole vrcholů, která jsou v literatuře často označovány jako *vertex buffery*.

Nevýhodou sdílení dat vrcholů mezi polygony je skutečnost, že nyní grafická karta potřebuje další data o tom, které trojúhelníky jsou tvořeny kterými vrcholy. Data jsou označována jako indexová pole (*index buffers*). Každý index určuje pořadové číslo vrcholu. Pro popis vrcholu jednoho polygonu jsou pak potřeba tři indexy.

Pokud by jen málo polygonů sdílelo vrcholy, přinesly by indexová pole v důsledku zvětšení dat. Vzhledem k množství dat potřebných pro popis vrcholu se však indexová pole vyplatí skoro vždy. U modelu stromů jsou například listy tvořeny minimálně dvěma polygony, takže i zde se indexová pole a sdílení vrcholů vyplatí.

Sdílení vrcholů však nepřináší jen zmenšení přenášených dat, ale také urychlení samotného vykreslování. Pokud je stejný vrchol posílán do grafické karty dvakrát, musí karta provést jeho transformace a výpočty osvětlení také dvakrát. Pokud jsou však data vrcholu přeneseny jednou a samotný vrchol je pouze specifikován dvakrát pomocí indexů, grafické karty umí provést transformaci a osvětlení tohoto vrcholu pouze jednou a výsledky si uložit do vyrovnávací paměti a při opětovném zpracování daného vrcholu pak využít již napočítaná data.

Základní modul Rose systému pro generování polygonové geometrie z kom-



Obrázek 3.1: Reprezentace polygonů pomocí pole vrcholů a pole indexů. Vrcholy sdílené více polygony jsou uloženy pouze jednou. Pole vrcholů je přesunuto do paměti grafické karty pro co nejefektivnější zpracování.

ponent obsahuje podporu pro sdílení vrcholů a indexových polí a jeho výstupní formát je navržen pro co nejjednodušší napojení na rozhraní grafických knihoven OpenGL a Direct3D.

3.2.4 Frekvence přesunu dat

Frekvence posílání dat do grafické karty je základním faktorem, který určuje celkovou rychlost vykreslování. Pokud například chceme model o 30 tisících polygonů, což může představovat zhruba 3 MB dat jen na popis geometrie, vykreslovat frekvencí 60 Hz (šedesát snímků za sekundu), dostáváme najednou $3*60 = 180$ MB dat, které je potřeba přenést do grafické karty každou sekundu.

Pokud se geometrická data mění v každém snímku, přenos dat nelze nijak vylepšit. Pokud se však data často neměnní, je zbytečné je neustále posílat do grafické karty pro každý snímek. Současné grafické karty umožňují statická geometrická data přenést pouze jednou a uložit je v paměti karty (*vertex buffers* — pole vrcholů).

Tento postup lze v případě modelů stromů a rostlin velmi dobře využít. Aplikace Viewer dokáže zobrazovat geometrii v obou režimech, tedy jak přenášet data neustále v každém snímku, tak přenést data do paměti karty jen jednou a v dalších snímcích na ně používat referenci. Výkonnostní nárůst při použití statických vertexových polí je několikanásobný.

Situace se trochu komplikuje, pokud chceme stromy animovat. Abychom minimalizovali přenos dat, je nejlepší nechat provádět animaci grafickou kartu.

To lze provést dvěma způsoby.

Prvním způsobem je využit matice, kterými se geometrická data transformují. Všechny polygony se před vykreslením transformují množinou transformací, typicky je každý model posunut na nějakou pozici ve vykreslované scéně a určitým způsobem natočen. Transformace lze samozřejmě skládat a provádět je jen na skupinách vertexů. Proto lze model stromu rozsekat na disjunktní skupiny polygonů a model animovat změnami transformací pro jednotlivé skupiny. Tento způsob sice neumožňuje příliš komplexní animace, ale lze s ním provádět jednoduchý pohyb větví a simulovat tak vítr, což je pro interaktivní aplikace často dostačující. Navíc pokud skupin polygonů nejsou stovky, nepředstavují tyto transformace skoro žádné zpomalení vykreslování.

Rose systém podporuje přímo v modulu pro generování geometrie možnosti rozsekat geometrie na bloky polygonů, které lze při vykreslování tímto způsobem animovat. Toho využívá i aplikace Viewer, která implementuje simulaci větru právě tímto způsobem.

Druhým způsobem je využít vertexové shadery. Vertexové shadery jsou programy, které provádějí transformace vrcholů polygonů transformačními maticemi. Moderní grafické karty umožňují programátorům vytvářet vertexové shadery, díky čemuž lze provádět libovolné manipulace s polygony. Pro každý blok polygonů, který se vykresluje, je tedy možno specifikovat jiný vertexový program (nebo jiné parametry) a díky tomu provádět náročnější animaci či deformace modelu. Jedinou nevýhodou vertex shaderů je, že jejich používání není úplně standardizované a starší karty vertex shadery nepodporují vůbec.

3.2.5 Množství geometrických primitiv

Jakmile má grafické karta všechna geometrická data zpřístupněna, záleží již čistě na jejím výkonu, jak rychle dokáže data zpracovat a vykreslit, a programátor zde již nemá možnost výsledek jakkoli ovlivnit. Grafická karta může umožňovat uživateli konfiguraci různých aspektů svého chování (například přesnost výpočtů) a tak ještě o něco urychlit vykreslování, ale to je zcela mimo kontrolu programu (a navíc málokdy způsobuje pozorovatelnou změnu).

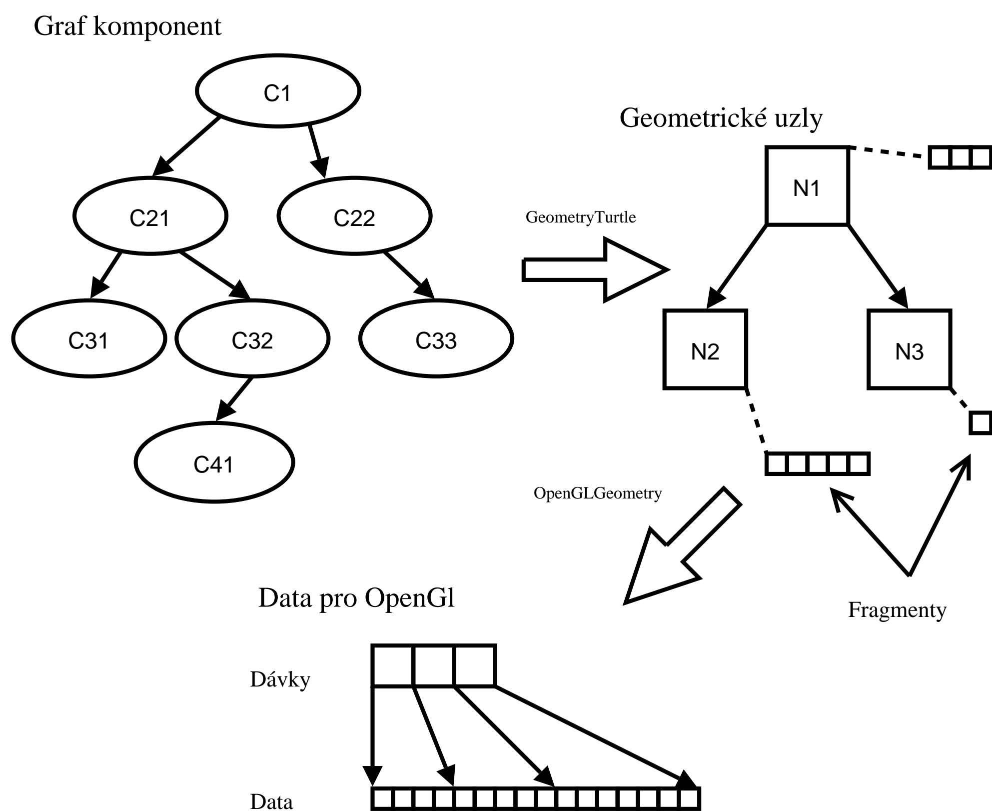
3.3 Generování geometrie pro OpenGL

Implementace generování geometrie do formátu vhodném pro OpenGL je v Rose rozdělena do dvou částí. Postup je znázorněn na obrázku 3.2.

První část tvoří objekt *GeometryTurtle*, který z grafu komponent vytváří struktury dat, která částečně kopírují strukturu grafu komponent, ale již obsahují skupiny polygonů rozděleny podle použitých materiálů.

Druhou část tvoří objekt *OpenGLGeometry*, který na vstupu přijímá data vygenerovaná z *GeometryTurtle* a vytvoří z nich bloky dat přímo optimalizovaná pro OpenGL rozhraní.

Toto rozdělení bylo zvoleno tak, aby šlo v budoucnu jednoduše implementovat výstup pro Direct3D nahrazením *OpenGLGeometry* jiným objektem, protože rozhraní OpenGL a Direct3D si jsou velmi podobná.



Obrázek 3.2: Průběh zpracování grafu komponent a vytvoření dat pro OpenGL.

3.3.1 Fragment

Geometrický fragment je základní stavební jednotka, která popisuje skupinu geometrických primitiv (v našem případě polygonů). Fragment definuje skupinu polygonů vztaženou ke společné souřadné soustavě.

Fragment je tvořen polem vrcholů a polem indexů. Pole vrcholů obsahuje popis všech vrcholů všech polygonů z daného fragmentu. Jeden vrchol může být sdílen více polygony. Každý vrchol definuje pozici, barvu, texturové souřadnice a normálový vektor.

Polygony obsažené ve fragmentu jsou pak definovány v poli indexů. Každý prvek indexového pole obsahuje index do pole vrcholů, každé tři indexy tedy definují jeden polygon.

Fragment sám o sobě není nijak spojen s materiálem a tedy se způsobem, jakým se budou polygony vykreslovat. Popis vrcholů pouze obsahuje definice textových souřadnic, ty je však možno použít pro libovolnou texturu. Stejně tak fragment neví nic o pozici, vzhledem ke které se budou polygony fragmentu vykreslovat. Všechny polygony fragmentu jsou definovány vzhledem k pevné souřadné soustavě a je na tom, kdo bude polygony vykreslovat, aby tuto souřadnou soustavu umístil na správné místo a orientoval správným směrem.

3.3.2 Stem

Stem objekt slouží pro jednoduché zadávání geometrie stonků a větví. Stonek je definován posloupností N kontrolních bodů. Každý kontrolní bod má pozici, orientaci a poloměr, a představuje kružnici o daném poloměru na této pozici natočenou dle zadané orientace. Každá kružnice je pak spojena polygony s předchozí a následující kružnicí, čímž vzniká tvar "trubky" použitý pro reprezentaci stonku.

Stem objekt je při vytvoření napojen na fragment, do kterého postupně vytváří geometrická data. Podobně jako fragment i *Stem* objekt sám o sobě neví nic o použitém materiálu, pouze generuje polygonová data do příslušného fragmentu a automaticky generuje texturové souřadnice tak, aby textura povrchu plynule navazovala.

Komponenty zadávají pouze kontrolní body a *Stem* objekt automaticky vytváří polygony stonku či větve. Kontrolní body jsou zadávány relativně vůči souřadnému systému odpovídající komponenty. Kontrolní body nemusí zadávat pouze jediná komponenta, ale i více komponent po sobě. V tom případě jsou zaručeny správné transformace kontrolních bodů podle souřadných systému daných komponent do souřadného systému, ve kterém *Stem* objekt geometrii generuje.

3.3.3 Uzel

Geometrický uzel reprezentuje část geometrie v modelované rostlině. Typicky geometrický uzel obsahuje definici geometrie z jedné či více komponent.

Každý geometrický uzel je spjat se souřadnou soustavou, která určuje, kde je umístěna geometrie z tohoto uzlu. Veškerá geometrie uložená v tomto uzlu

je pak automaticky transformována do tohoto souřadného systému. Souřadná soustava uzlu odpovídá místu, kde uzel vznikl při procházení grafu komponent, ze kterého se geometrie generovala.

Geometrické uzly jsou spojovány vztahem rodič/potomek podobně jako komponenty. Tento vztah říká, že souřadný systém dceřinného uzlu závisí na souřadném systému rodičovského uzlu. Při změně souřadného systému rodičovského uzlu se odpovídajícím způsobem změní umístění všech (i nepřímých) dceřinných uzlů.

Každý geometrický uzel obsahuje dvě transformace. První transformace určuje umístění souřadné soustavy uzlu vzhledem k celé rostlině. Druhá transformace určuje umístění souřadné soustavu uzlu relativně k rodičovskému uzlu.

Dále uzel obsahuje seznam fragmentů, které obsahují geometrii daného uzlu. Fragmenty jsou rozděleny podle materiálů, každý použitý materiál má přiřazen jeden fragment. Veškerá geometrie (i z více komponent) se stejným materiálem je ukládána do jediného fragmentu. Při vykreslování modelu v reálném čase je potřeba pro každý materiál nakonfigurovat grafickou kartu, proto je výhodné minimalizovat počet použitých materiálů. Z tohoto důvodu je brán zřetel na to, aby co nejvíce geometrie se stejným materiálem bylo uloženo do jediného fragmentu.

Například jedna větev může obsahovat desítky menších větví a větviček, což může znamenat stovky komponent. Pokud by se každá větvička vykreslovala zvlášť, už jen samotný průchod komponentami a konfigurace grafické karty by zabralo veliké množství času. Pokud ale všechny větve používají stejný materiál a použije se jeden geometrický uzel pro celou větev, lze celou geometrii uložit do jediného fragmentu. Při vykreslování pak stačí nakonfigurovat materiál jednou a do grafické karty poslat celý fragment.

Dále uzel obsahuje seznam použitých *Stem* objektů pro generování geometrie větví a stonků. Každý použitý *Stem* objekt je napojen na fragment odpovídající jeho materiálu. Opět pro jeden materiál existuje jeden fragment, pokud by existovalo několik *Stem* objektů se stejným materiálem, budou všechny napojeny na stejný fragment. Rozhraní pro generování polygonů a implementace *Stem* objektů byla vytvořena tak, aby mohlo více objektů bez problémů přidávat a měnit geometrii v jediném fragmentu a nedocházelo ke konfliktům.

3.3.4 GeometryTurtle

GeometryTurtle objekt implementuje rozhraní *Turtle* objektu tak, že geometrii generovanou komponentami převádí na graf geometrických uzlů. Objekt převádí geometrická data ze souřadných systémů komponent do souřadného systému aktuálního uzlu a ukládá je do odpovídajících fragmentů. Výsledkem celého procesu je graf uzlů reprezentující geometrii celé modelované rostliny. Stejně jako graf komponent i graf geometrických uzlů tvoří strom v matematickém smyslu slova.

3.3.5 OpenGLGeometry

Posledním krokem pro vytvoření dat, která lze přímo použít v rozhraní OpenGL pro vykreslování geometrie modelované rostliny, je konverze geometrie reprezentované grafem geometrických uzlů, který tvoří výstup objektu *GeometryTurtle*. K tomuto účelu slouží objekt *OpenGLGeometry*.

OpenGLGeometry objekt dostává na vstupu graf uzlů, tento graf zpracuje a vytvoří data optimalizovaná pro rozhraní OpenGL tak, aby šla geometrie co nejefektivněji zobrazovat na současných grafických kartách, jak bylo popsáno v kapitole 3.2.

Výstupní data obsahují tři struktury popisující geometrii:

- Pole vrcholů.

  Pole vrcholů obsahuje popis vrcholů všech polygonů geometrie rostliny. Pro každý vrchol je uložena jeho pozice, normálový vektor, barva a texturové souřadnice. Pro zmenšení velikosti dat je vrchol sdílený více polygony uložen v poli pouze jednou. Které vrcholy tvoří polygony je definováno v poli indexů.

- Pole indexů.

  Pole indexů obsahuje definice polygonů. Každý prvek indexového pole obsahuje index do pole vrcholů a identifikuje tak jeden vrchol v geometrii. Tři sousedící prvky indexového pole pak definují právě jeden polygon.

- Seznam dávek.

  Dávka je struktura, která popisuje, jakým způsobem se má vykreslit skupina polygonů. Seznam dávek pak popisuje jak vykreslit všechny polygony a tak celý model rostliny. Seznam dávek je vytvořen tak, že uchovává informaci z jakého geometrického uzlu skupina polygonů pochází, jakou úroveň měla odpovídající komponenta a transformace souřadných soustav dávek jsou popsány relativně k rodičovským dávkám odpovídajícím rodičovským geometrickým uzlům, což lze využít pro animace napodobující vlnění ve větru, jak je popsáno v kapitole 5. Přestože seznam dávek popisuje graf geometrických uzlů, k jeho vykreslení stačí jednoduchý lineární průchod seznamem.

3.3.6 Dávka

Dávka je struktura popisující skupinu polygonů z vygenerované geometrie modelu rostliny. Dávka obsahuje:

- Identifikace polygonů patřící do dávky.

  Které polygony patří do dávky je určeno jako rozsah prvků v poli indexů. Polygony tvoří vždy souvislý blok v poli indexů, proto k jejich určení stačí pouze index prvního vrcholu a počet následujících indexů. Jelikož každé tři indexy definují jeden polygon, je počet indexů v dávce vždy násobek tří.

- Transformaci souřadné soustavy.

  Dávka obsahuje dvě matice velikosti 4×4, které popisují, jakým způsobem se musí transformovat pozice vrcholů polygonů z této dávky do souřadného systému modelu rostliny. První matice specifikuje transformaci relativně vůči modelu rostliny a lze ji použít, pokud nebudou transformace skládány při vykreslování při průchodu seznamem dávek. Druhá matice specifikuje transformaci relativně vůči rodičovské dávce. Pokud se budou na některé dávky při vykreslování aplikovat přídavné transformace (například pro animaci vlnění ve větru), je potřeba použít matice popisující relativní transformace a tyto matice skládat s rodičovskými maticemi při průchodu seznamem dávek. Obě matice jsou uloženy v OpenGL formátu a lze je použít přímo jako argumenty funkcí tohoto rozhraní.

- Materiál.

  Identifikace materiálu, který se má použít pro vykreslení polygonů z dávky.

- Úroveň.

  Popisuje úroveň, v jaké se dávka nachází. Odpovídá úrovni komponenty, ze které polygony uložené v dávce pocházejí. Pokud dávka obsahuje geometrii z více komponent z různých úrovní, je úroveň dávky rovna nejnižší z těchto úrovní.

- Identifikace geometrického uzlu.

  Při zpracování grafu uzlů jsou všechny uzly jednoznačně očíslovány. Dávky pocházející z jednoho uzlu mají v sobě uloženu identifikace tohoto uzlu. Tuto identifikace lze použít pro optimalizace práce s transformacemi dávek, kdy stačí pro dávky z jednoho geometrického uzlu aplikovat transformační matice pouze jednou a tak eliminovat redundantní operace.

3.3.7 Vykreslování dávek v OpenGL

Prvním krokem pro vykreslování modelu v OpenGL je specifikace dat popisujících vrcholy polygonů. Data jsou uložena v poli vrcholů a je tedy potřeba oznámit OpenGL, kde se nachází. Jak bylo popsáno v kapitole 3.2, aby grafická karta mohla s daty pracovat, je potřeba data přenést do paměťového prostoru karty. Rozhraní OpenGL verze 1.2, pro kterou byl program této práce primárně napsán, bohužel nepodporuje možnost přenést data do grafické paměti pouze jednou. Tato možnost přišla až v novějších verzích tohoto rozhraní. Rozhraní OpenGL verze 1.2 poskytuje pouze možnost označit blok dat jako pole vrcholů a specifikovat polygony pro kreslení přes indexy do tohoto pole. Tento postup se ale musí aplikovat pro každý vykreslovaný snímek a data pole vrcholů jsou tak přenášena z paměti procesoru do paměti grafické karty neustále, což má velký dopad na celkový výkon vykreslování.

OpenGL verze 1.2 sice neposkytuje standardní možnost přesunu pole vrcholů do paměti grafické karty pouze jednou, což je limitace verze rozhraní, ale ne grafických karet, jejichž ovladače tuto možnost umí. OpenGL rozhraní ale

při svém vzniku počítalo s tím, že poskytované rozhraní bude zaostávat za možnostmi poskytovaných grafickými kartami. Proto OpenGL obsahuje možnost, jak využívat novějších možností grafických karet pomocí tzv. extenzí. Extenze jsou rozšíření standardního rozhraní OpenGL, které umožňují využívat funkce grafických ovladačů, které ještě nebyly zahrnuty do tohoto rozhraní.

Pro přenos dat do grafické karty a jejich opětovné využítí bez nutnosti je znova přenášet existuje několik extenzí podle různých výrobců grafických karet. Tyto extenze byly později sjednoceny do extenze *ARB vertex buffer object*, která se stala součástí standardního rozhraní OpenGL verze 1.5.

Program *Viewer*, který je součástí této práce, používá tuto extenzi pro ukládání geometrických dat v paměti karty. Jelikož jsem pracoval také se starší kartou od firmy ATI, která tuto extenzi nepodporuje, používá program na těchto grafických kartách podobnou extenzi *ATI vertex array object*. Pokud ani jedna extenze není podporována, používá program standardní rozhraní OpenGL, kdy jsou data vrcholů posílána do grafické karty pro každý snímek.

Jakmile jsou data vrcholů připravena, je možno přikročit k vlastnímu vykreslování geometrie. Celý popis jak geometrii vykreslit je uložen v seznamu dávek. Před započetím zpracování dávek je ještě potřeba nakonfigurovat základní stav OpenGL a nastavit pozici a orientaci celého modelu ve vykreslované scéně, ale to je čistě technická záležitost a není potřeba ji zde zmiňovat. Seznam dávek se poté prochází lineárně po jednom prvku a každá dávka se zpracuje následujícím způsobem:

- Podle materiálu uloženého v dávce se nakonfiguruje OpenGL. Vybere se aktuální textura (pokud není v paměti je potřeba ji načíst) a nastaví se módy zpracování alfa kanálu a způsob vykreslování polygonů.

- Provede se nastavení OpenGL matic, kterými se budou transformovat zadávané vrcholy. Pokud má dávka úroveň stejnou jako předchozí dávka a pochází ze stejného geometrického uzlu, není potřeba nijak transformační matici měnit. V opačném případě je potřeba zaručit správné nastavení matice. Pokud má dávka menší úroveň než předchozí, obnoví se transformační matice buď ze zásobníku matic, který poskytuje OpenGL, nebo lze použít matici uloženou v dávce, která určuje transformaci dávky relativně vůči celé rostlině. Pokud má dávka úroveň větší než předchozí, složí se relativní transformace dávky s aktuální transformační maticí prostým maticovým vynásobením.

- Nakonec se pomocí pole indexů a informací uložených v dávce provede specifikovat polygonů, které se mají vykreslit.

Celý postup se opakuje pro každý vykreslovaný snímek.

Kapitola 4

Model stromu

4.1 Přehled

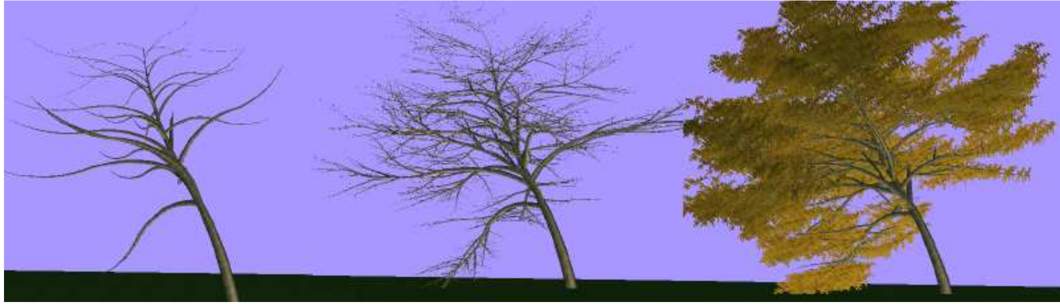
V Rose systému jsem implementoval model stromu, který vychází z práce, kterou publikoval Weber a Penn [6]. Přestože má implementace se od modelu popsaného v této práci liší (už jen z toho důvodu, že v práci jsou některé části modelu popsány jen zevrubně a nejsou zmíněny implementační detaily) a rozšiřuje tento model o vývoj stromu v čase, zůstal jsem u označení této implementace jako Weber model.

Přístup tohoto modelu je založen na tom, aby vygenerované stromy odpovídaly vizuálně co nejvíce reálným stromům a aby poskytoval mnoho parametrů pro vytváření nejrůznějších variací. Model si neklade za cíl snažit se simulovat biologické pochody odehrávající se při vývoji stromu, ale místo toho se snaží pozorováním reálných stromů a větví popsat jejich konečnou strukturu. Parametry mají globální charakter a ovlivňují celkové vzezření stromu. Z těchto důvodů rozšíření o plynulý vývoj při bližším pohledu příliš neodpovídá vývoji reálnému stromu v přírodě, ale slouží hlavně jako ukázka možností tohoto systému.

4.2 Model stromu

Struktura stromu je modelována kmenem a několika úrovněmi větví. Na poslední úrovni větví pak vyrůstají listy. Kmen a větve jsou tvořeny komponentou *Stem*, která představuje různorodě zakřivenou strukturu podobnou kuželu. Listy jsou tvořeny instancemi komponenty *SimpleLeaf*, která modeluje listy jako texturu namapovanou na dva polygony, pro co nejmenší paměťové nároky generované geometrie vzhledem k velikému počtu listů a co nejrychlejší zobrazování.

Z hlavního kmene a jeho větví vyrůstají dceřinné větve vyšších úrovní. Tyto větve mají typicky výrazně odlišné atributy než jejich rodičovské větve. Mnoho atributů jako je délka či poloměr je definováno relativně vůči odpovídajícím atributům rodičovských větví. Například délka dceřinných větví je definována jako zlomek délky rodiče. Dceřinné větve pak vytvářejí další dceřinné větve vyšší úrovně až do zadaného limitu.



Obrázek 4.1: Modelování stromu přidáváním úrovní.

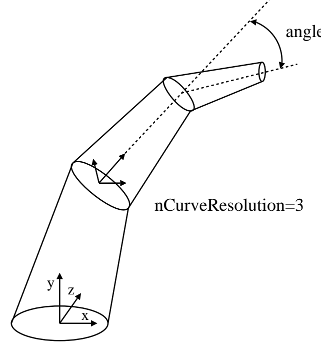
Parametry odlišené pro jednotlivé úrovně větví umožňují celý model jednoduše konfigurovat. Na obrázku 4.1 je ukázka, jak rozdělení na úrovně pomáhá při návrhu stromu, kdy se muže začít se zobrazováním pouze nulté úrovně větví (tedy kmene). Jakmile je vzhled kmenu vyhovující, povolí se zobrazování první úrovně větví (větve vyrůstající z kmene). Takto se pokračuje pro druhou a případně třetí úroveň. Tento přístup umožňuje rychle definovat celkový tvar již v počátku návrhu, bez nutnosti generovat a zobrazovat geometrii všech úrovní.

Parametry budeme v textu označovat kurzívou, jako např. parametr *Shape*. Mnoho parametrů se opakuje pro každou úroveň. Tyto parametry budeme prefixovat číslem před jménem, které reprezentuje úroveň daného parametru. Pokud se budeme chtít odkázat na parametry ze všech úrovní, použijeme jméno parametru s prefixem *n*, např. *nLength*. Pro číselné parametry s desetinnou čárkou lze zadávat i odchylky, v jakém rozmezí se může hodnota parametru pohybovat. Odchylka je typicky kladná hodnota určující maximální velikost vychýlení hodnoty parametru od základní hodnoty. Odchylky budeme označovat příponou *V*, např. *nLength* a *nLengthV*. Všechny parametry specifikující úhly jsou zadávány ve stupních.

Vzorce a konstanty uváděné dále pocházejí ze zmíněné práce, kterou publikoval Weber a Penn [6]. Jak již bylo uvedeno, model se zajímá pouze o konečný vzhled stromu, a tak lze předpokládat, že uváděné vzorce a konstanty byly zvoleny experimentálně, než aby pocházely z exaktních měření.

4.2.1 Větev

Pojem "větev" budeme používat pro obecné označení jak kmene, tak větví na všech úrovních. Větve jsou reprezentovány komponentou *Stem*. Větev je zakřivená struktura odpovídající kuželu, která se zúžuje ke svému konci a má kruhový průřez. Každá větev má vlastní souřadný systém, kdy osa y odpovídá ose kuželu. Každá větev tedy "roste" ve směru lokální osy y. Osa z směřuje směrem k obloze a osa x je orientována dle pravidla levé ruky tak, aby byla rovnoběžná s rovinou země. Pro kmen jsou obě osy x a z rovnoběžné s touto rovinou. Větev na úrovni n je rozdělena na několik válcovitých segmentů, jejichž počet je definován parametrem *nCurveResolution*. Každý segment si udržuje informaci o pozicích kruhových řezů, které jsou později spojeny polygony v je-



Obrázek 4.2: Reprezentace větve pomocí objektu *Stem*.

dinou geometrii.

Zakřivení větve je určeno parametry *nCurve* a *nCurveBack*. Pokud je *nCurveBack* nulový, osa y každého segmentu je otočena relativně k předchozímu segmentu o úhel

$$angle = (nCurve/nCurveResolution)$$

kolem osy x. Pokud je *nCurveBack* nenulový, je každý segment z první polovina větve otočen o úhel

$$angle = (nCurve/(nCurveResolution/2))$$

a každý segment ze zbylých otočen o úhel

$$angle = (nCurveBack/(nCurveResolution/2))$$

Rozdělení segmentů do dvou částí umožňuje vytvářet zakřivení větve ve tvaru písmene S. V obou případech je na každý segment aplikováno náhodné otočení o úhel $(nCurveV/nCurveResolution)$.

4.2.2 Dceřinné větve

Parametry popisující větve se zaměřují na jejich celkový charakter a ne pouze na relativní vztah mezi sousedními segmenty, jak tomu je u některých jiných přístupů, což umožňuje uživateli jednodušeji pochopit a představit si vliv změny parametrů.

Parametr *nMaximumBranchCount* definuje maximální počet dceřinných větví, které mohou vzniknout na všech segmentech rodičovské větve. Skutečný počet však může být menší než toto maximum, aby se kompenzovala proměnlivá délka rodičovské větve a dceřinné větve nevytvářely shluky.

Pro kmen na nulté úrovni je počet větví roven maximu. Pro větve vyrůstající z kmene na první úrovni je počet dceřinných větví roven

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

kde $stems_{max}$ je maximální počet dceřinných větví, $length_{child}$ označuje v metrech délku větve na první úrovni, $length_{parent}$ označuje délku rodičovské větve (v tomto případě přímo kmene), a $length_{child,max}$ označuje maximální relativní poměr mezi délkou větve na první úrovni a délkou kmene stromu.

Maximální relativní poměr je definován jako

$$length_{child,max} = nRelativeLength \pm nRelativeLengthV$$

kde parametr *nRelativeLength* určuje délku větví dané úrovně. Parametr *nRelativeLength* je zadáván jako zlomek relativně k délce rodičovské větve. Pokud je hodnota $length_{child,max}$ například rovna $0,3$ a rodičovská větev má délku 10 metrů, může dceřinná větev dosáhnout maximální délky 3 metry.

Pro větve na druhé a vyšších úrovních je počet dceřinných větví spočten jako

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

kde $offset_{child}$ je vzdálenost podél rodičovské větve od jejího počátku až po místo, kde se nachází dceřinná větev. Čím dále je větev od základny rodičovské větve, tím méně dceřinných větví o jednu úroveň výše na ní vznikne.

Skutečná délka kmene stromu je spočtena jako

$$length_{trunk} = scale*length_{child,max} = scale*0RelativeLength \pm 0RelativeLengthV$$

kde *scale* je celková velikost stromu definována globálním parametrem *Scale* jako

$$scale = Scale \pm ScaleV + ExtraTrunkScale \pm ExtraTrunkScaleV$$

Zatímco hodnota parametru *Scale* určuje celkovou velikost stromu, parametr *ExtraTrunkScale* slouží pro škálování pouze velikosti kmene stromu. Při určování délek dceřinných větví je parametr *ExtraTrunkScale* ignorován, jakoby na délku kmene neměl vliv.

Délka větve první úrovně je určena výrazem

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

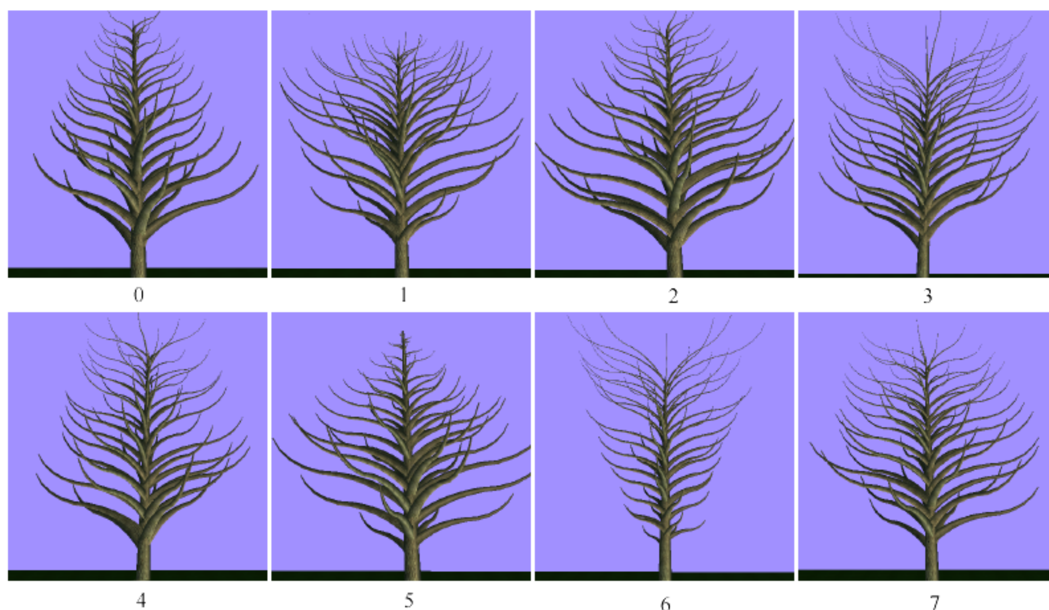
kde *shape* je faktor, který ovlivňuje délky větví tak, že celý strom získá požadovaný tvar, a je definován jako

$$shape = ShapeRatio((length_{trunk} - offset_{child})/(length_{trunk} - length_{base}))$$

Hodnota $length_{base}$je délka v metrech určená globálním parametrem *BaseSize*, která určuje v jaké vzdálenosti od země mohou vyrůstat první větve na kmeni stromu. Pro větve na jiné než první úrovni je hodnota $length_{base}$ rovna nule. *ShapeRatio* je funkce, která podle globálního parametru *Shape* a zadané hodnoty v rozmezí od nuly do jedné vrátí hodnotu *scale*. Jak se postupuje od spodních větví ke větvím ke konci kmene, tak roste hodnota parametru zadávanému funkci *ShapeRatio*.

Shape	Hodnota funkce
0	$0,2 + 0,8 * ratio$
1	$0,2 + 0,8 * \sin(\pi * ratio)$
2	$0,2 + 0,8 * \sin(0,5 * \pi * ratio)$
3	$1,0$
4	$0,5 + 0,5 * ratio$
5	$\begin{cases} ratio/0,7 & ratio \leq 0,7 \\ (1,0 - ratio)/0,3 & ratio > 0,7 \end{cases}$
6	$1,0 - 0,8 * ratio$
7	$\begin{cases} 0,5 + 0,5 * ratio/0,7 & ratio \leq 0,7 \\ 0,5 + 0,5 * (1,0 - ratio)/0,3 & ratio > 0,7 \end{cases}$

Tabulka 4.2: Hodnoty funkce *ShapeRatio* v závislosti na parametru *Shape*, jak uvádějí Weber a Penn [6].



Obrázek 4.3: Vliv hodnoty parametru *Shape* na tvar stromu.

Funkce *ShapeRatio* vrací hodnoty z připravených funkcí, které odpovídají nejznámějším tvarům stromů. Hodnoty v závislosti na parametru *Shape* zobrazuje tabulka 4.2, kde *ratio* označuje vstupní hodnotu funkce. Vliv hodnoty parametru *Shape* na tvar stromu lze vidět na obrázku 4.3.

Délka větví druhé a vyšších úrovní je pak definována jako

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

Pro odklon dceřinných větví od rodičovské větve slouží parametr *nBranchDownAngle*. Pokud je hodnota *nBranchDownAngleV* kladná, je dceřinná větev otočena kolem osy x od rodičovské větve o úhel

$$downangle_{child} = nBranchDownAngle \pm nBranchDownAngleV$$

Pokud hodnota *nBranchDownAngleV* není kladná, je odchylka parametru distribuována podél rodičovské větve, kdy je každá dceřinná větev otočena o úhel

$$downangle_{child} = nBranchDownAngle \pm (nBranchDownAngleV * factor)$$

kde

$$factor = 1 - 2 * ShapeRatio(0, ratio)$$

$$ratio = (length_{parent} - offset_{child}) \, / \, (length_{parent} - length_{base})$$

Čím se dceřinná větev nachází blíže ke konci rodičovské větve, tím se hodnota *ratio* blíží k nule a úhel odklonu se zmenšuje. To lze využít ke změně odklonu v závislosti na pozici dceřinné větve, kdy větve v horní části stromu směřují k obloze a větve v dolní části stromu směřují k zemi.

Pro rotaci dceřinných větví kolem rodičovských slouží parametr *nBranchRotation*. Pokud je hodnota *nBranchRotation* kladná, je každá dceřinná větev otočena kolem osy y rodičovské větve relativně k předchozí dceřinné větvi o úhel

$$rotation = nBranchRotation \pm nBranchRotationV$$

Pokud je hodnota *nBranchRotation* záporná, je každá lichá větev otočena kolem osy y rodičovské větve o úhel

$$rotation = (270 + nBranchRotation \pm nBranchRotationV)$$

a každá sudá o úhel

$$rotation = (90 + nBranchRotation \pm nBranchRotationV)$$

Takto jsou dceřinné větve vytvářeny střídavě na obou stranách rodičovské větve, čímž lze jednoduše docílit efektu, kdy větve mají tendenci růst rovnoběžně se zemí.

Parametr *nBranchesLocationRatioPower* určuje distribuci dceřinných větví podél rodičovské větve. Dceřinná větev je vytvořena podél rodičovské větve ve vzdálenosti

$$pos = ratio^{nBranchesLocationRatioPower} * length_{parent}$$

kde

$$ratio = (length_{parent} - offset_{child}) \,/\, (length_{parent} - length_{base})$$

Čím menší je hodnota parametru, tím více jsou dceřinné větve koncentrovány blíže ke konci rodičovské větve. Pokud je hodnota parametru rovna 1, jsou dceřinné větve distribuovány rovnoměrně podél rodičovské větve.

Materiál, který bude použit při vykreslování větví, lze nastavit parametrem *StemMaterial*.

4.2.3 Poloměr větví

Pro všechny úrovně kromě kmene je poloměr větve definován v závislosti na poloměru rodičovské větve v místě vzniku dceřinné větve. Poloměr kmene je definován vzhledem k velikosti celého stromu

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

kde *Ratio* je hodnota parametru určující poměr mezi délkou a poloměrem kmene stromu. Poloměr větví první a dalších úrovní je roven

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

kde $radius_{parent}$ určuje poloměr rodičovské větve a hodnota parametru *RatioPower* určuje rychlost poklesu poloměru dceřinných větví. Maximální poloměr dceřinné větve je automaticky limitován hodnotou poloměru rodičovské větve. Poloměr větve se podél její délky plynule zmenšuje.

Pro přenos informací mezi větvemi nastavuje každá rodičovská větev několik parametrů do svých slotů pro dceřinné větve. Parametr *ParentLength* označuje skutečnou délku rodičovské větve v metrech. Poloměr rodičovské větve v jejím počátku určuje parametr *ParentRadius*. Poloměr rodičovské větve v místě napojení dceřinné větve je předáván v parametru *CurrentBaseRadius*. Parametr *StemOffset* určuje vzdálenost v metrech od počátku rodičovské větve až po místo, kde vzniká dceřinná větev. Všechny tyto parametry jsou automaticky aktualizovány při vývoji stromu v čase.

4.2.4 Listy

Celkový počet úrovní větví je limitován globálním parametrem *LevelCount*. Pokud je hodnota parametru rovna jedné, je vytvořen pouze kmen stromu. Obvyklá hodnota je 3 nebo 4. Větve na poslední úrovni vytvářejí místo dceřinných větví listy na úrovni rovné hodnotě *LevelCount*.

Počet listů je omezen hodnotou globálního parametru *MaximumLeafCount*. Hodnota tohoto parametru je interpretována podobně jako parametr *nMaximumBranchCount* pro počty dceřinných větví, kdy reálný počet listů závisí i na dalších faktorech. Skutečný počet listů na jedné větvi je definován jako

$$leaf_{count} = MaximumLeafCount * ShapeRatio(4, ratio)$$

kde

$$ratio = (offset_{child}/length_{parent})$$

Pro orientaci listů jsou obdobně využity parametry *nBranchRotation*, *nBranchDownAngle* a *nBranchesLocationRatioPower*.

Listy jsou tvořeny instancemi komponenty *SimpleLeaf*, která modeluje list jako texturu namapovanou na dva polygony tvořící obdélník, jehož velikost lze řídit parametry komponenty. Textura listu nemusí představovat pouze jediný list. Naopak je výhodné, aby textura reprezentovala shluk několika listů. Velikost listu pak může být mnohem větší, čímž se drasticky zmenší množství generované geometrie, neboť listů bývá typicky stovky až tisíce na jeden strom a i při jednoduché reprezentaci dvou polygonů na list by množství výsledné geometrie bylo příliš velké. Pokud by bylo cílem vymodelovat co nejdetailnější model stromu, lze jednoduše nahradit komponentu *SimpleLeaf* jinou komponentou, která může modelovat každý list zvlášť libovolným počtem polygonů.

Velikost listů lze kontrolovat parametry *nLeafWidth* a *nLeafHeight*. Hodnoty *nLeafWidthV* a *nLeafHeightV* opět slouží pro začlenění náhodnosti do zadaných hodnot. Parametr *nLeafExtraOffset* lze využít pro větší odsazení listů od větve v případě, že textura listu neobsahuje stonek a list je tak vizuálně příliš blízko větvi. Materiál použitý listy lze určit parametrem *LeafMaterial*.

Pokud je hodnota globálního parametru *AlignLeaves* nenulová, jsou listy orientovány tak, aby osa x jejich souřadného systému byla rovnoběžná s rovinou země. V praxi však změna tohoto parametru nepřináší žádné lepší výsledky.

4.2.5 Vývoj v čase

Celý model se dokáže vyvíjet v čase. Celková doba vývoje stromu je určena globálním parametrem *LifeTime*. Větve se vyvíjí tak, aby v zadaném čase dosáhly svého konečného tvaru.

Celá simulace je řízena událostmi. Události představují klíčové okamžiky ve vývoji stromu. V čase mezi událostmi se k získání hodnot používá interpolace. Komponenta *Stem* používá událost *EVENT_SEGMENT* pro vytváření segmentů, ze kterých je větev složena, a událost *EVENT_BRANCH* pro vytváření dceřinných větví. Při vzniku větve jsou naplánovány události pro vznik všech segmentů. Události pro vznik dceřinných větví jsou plánovány až v průběhu vývoje větve podle stavu segmentů, na kterých dceřinné větve vyrůstají.

Obrázek 4.4: Vývoj modelu stromu v čase.

Kapitola 5

Aplikace Viewer

Součástí této práce je také aplikace *Viewer*. Jedná se o program, který dokáže zobrazovat v reálném čase modely stromů vygenerované v Rose systému. Pro zobrazování geometrie používá program rozhraní OpenGL. Při zobrazování program umožňuje uživateli pohybovat se plynule ve scéně a prohlížet si model z libovolného směru.

Program dokáže pracovat ve dvou módech. V prvním módu vygeneruje zadaný model stromu a ten pak zobrazuje a umožňuje si strom prohlížet. V druhém módu program umožňuje zobrazovat animace vývoje celého modelu stromu v čase. Program zobrazuje geometrii vygenerovanou komponentami popsanými v kapitole 4.

5.1 Zobrazení modelu stromu

Pokud není zadáno jinak, program pracuje v módu, kdy načte konfigurační soubor a databázi materiálů, spustí simulaci vývoje a z výsledku vygeneruje geometrii, kterou poté zobrazuje.

Vstupní konfigurační soubor obsahuje definici parametrů popisující model stromu. Obsahem tohoto souboru je nainicializována databáze parametrů a parametrických prostorů pro jednotlivé úrovně komponent. Poté je načten soubor obsahující databázi materiálů. Databáze může obsahovat libovolné množství materiálů, protože program načítá pouze textury použitých materiálů až při vykreslování stromu.

5.2 Animace vývoje modelu stromu

Program umí kromě zobrazování celého modelu stromu zobrazovat i animaci jeho vývoje. Na příkazové řádce lze zadat počet snímků animace vývoje. Program poté simuluje vývoj zadaného stromu po úsecích tak, aby vznikl požadovaný počet snímků animace. Pro každou fázi vývoje se vygeneruje geometrie odpovídající geometrii stromu pro daný snímek.

Program poté zobrazuje model stromu tak, že vykresluje animaci rychlostí 30 snímků za sekundu. Poslední snímek animaci reprezentující finální geometrii

stromu vykresluje 5 sekund a poté pustí animaci vývoje pozpátku. Celý proces se opakuje.

5.3 Parametry programu

Volání programu má tvar

viewer *konfigurační_soubor* [*volby*]

Parametr *konfigurační_soubor* je povinný, ostatní volby jsou nepovinné. Kromě konfiguračního souboru program k běhu potřebuje databázi materiálu. Pokud není databáze explicitně zadána ve volbách, použije se implicitně soubor *materials.config*, který však musí existovat. Pokud je některá volba zadána vícekrát, použije se poslední zadaná hodnota.

Následující tabulka obsahuje popis všech voleb programu:

Volba	Popis
-f	Spuštění programu v celoobrazovkovém režimu. Pokud není tato volba zadána, program se spouští jako běžná aplikace v okně.
-w	Spuštění programu ve standardním okně.
-m [*soubor_materiálů* **]**	Jméno souboru obsahující definici materiálů. Pokud soubor není explicitně zadán, použije se soubor *materials.config*. Soubor musí existovat.
-i [*počet_iterací* **]**	Je-li volba zadána, program pracuje v animačním módu, kdy se vytvoří animace vývoje stromu. Animace vývoje bude mít tolik snímků, kolik je počet zadaných iterací. Pokud je počet iterací roven 1, chová se program jako v normálním módu, kdy zobrazuje pouze kompletní model stromu.

5.4 Využití OpenGL

Pro vykreslování geometrie využívá program extenze rozhraní OpenGL, které umožňují uložit pole vrcholů přímo v paměti grafické karty. Pokud nejsou extenze nalezeny, program používá standardní rozhraní pro zadávání pole vr-

cholů, které kopíruje geometrická data do paměti karty v každém snímku. Při použití extenzí je vykreslování zhruba 3 až 6 krát rychlejší, než standardní metoda.

Program dokáže využít extenze *ARB vertex buffer object* a *ATI vertex array object*. První extenze se stala součástí novějšího rozhraní OpenGL verze 1.5 a měla by být podporována většinou současných grafických karet. Druhá extenze je specifická extenze pro grafické karty od společnosti ATI a měla by být podporována i staršími kartami, které nemají podporu pro *ARB* extenzi. Obě extenze si jsou velmi podobné a liší se převážně v typech a formátech parametrů předávaným funkcím.

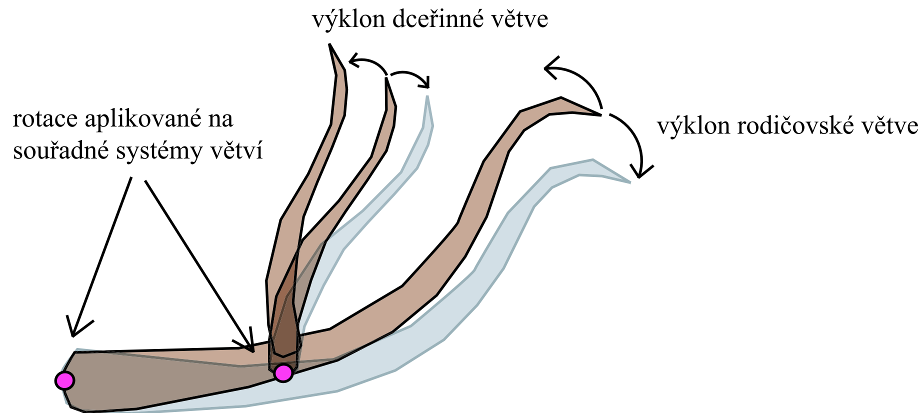
Po spuštění programu, vygenerování geometrie a nainicializování OpenGL program přenese pomocí extenzí vygenerované pole vrcholů do paměti karty. Získaná reference na toto pole vrcholů je pak použita pro specifikaci indexů polygonů při vykreslování dávek, jak bylo popsáno v kapitole 3.3.7.

5.5 Animace vlnění větví ve větru

Aplikace *Viewer* používá jednoduchou metodu pro simulaci vlnění stromu a větví ve větru. Cílem nebylo, aby se simuloval reálný vliv větru na jednotlivé větve, protože takový přístup by byl příliš náročný a vyžadoval by zpracovávat geometrická data procesorem pro každý vykreslovaný snímek, což by znamenalo značné snížení výkonu. Snahou bylo co nejjednodušším způsobem animovat větve tak, aby celkový dojem působil, jako by se celý strom vlnil ve větru. Jakmile jsou geometrická data přesunuta do paměťového prostoru grafické karty, je obtížné a pomalé je jakkoli modifikovat, proto bylo snahou provádět animaci tak, aby veškerou práci odváděla sama grafická karta.

Program provádí animaci větví tak, že v základně každé větve aplikuje na souřadnou soustavu transformaci, která provede rotaci geometrie kolem osy x o úhel α. Úhel vychýlení větve se periodicky mění v čase. Frekvence a velikost změny úhlu závisí na úrovni geometrického uzlu, jehož dávka se vykresluje. Čím blíže jsou větve topologicky ke kmeni, tím menší a pomalejší je změna výchylky. To způsobuje efekt, kdy velké větve napojené na hlavní kmen stromu pomalu pohybují, zatímco malé větévky se rychle třepotají, podobně jako má náhodný vítr malý vliv na velké větve o velké hmotnosti a velký vliv na malé větévky o malé hmotnosti. Jelikož je kmen interpretován také jako větev nulté úrovně, je malá výchylka aplikovaná i na něj, což způsobuje pomalé naklánění celého stromu.

Při vykreslování pak všechna vlnění větví dohromady působí dostatečně realisticky a projevuje se efekt lidského vidění, kdy při velkém množství detailů člověk nedokáže vnímat všechny detaily jednotlivě, ale celý model je vnímán věrohodně jako jeden celek.



Obrázek 5.1: Animace vlnění větví. Rotace jsou prováděny grafickou kartou bez snížení výkonu.

5.6 Formát konfiguračního souboru

Program *Viewer* na svém vstupu očekává konfigurační soubor, který obsahuje definici parametrů použitých při simulaci vývoje stromu. Soubor je uložen v textovém formátu a rozdělen na sekce. Jedna sekce definuje parametry jednoho parametrického prostoru.

Definice sekce má tvar

```
jméno_sekce {
    parametry...
}
```

kde *jméno_sekce* je buď *global* pro definici globálních parametrů, nebo má formát *level N*, kde *N* je číslo úrovně parametrického prostoru. Úrovně jsou číslovány od nuly, nultá úroveň odpovídá parametrům kmene, první úroveň odpovídá parametrům větví vyrůstajícím z kmene stromu atd. Oddíl *paramery...* je část obsahující definici vlastních parametrů. Parametry se zapisují za sebou, pokud je parametr stejného jména zadán vícekrát, použije se poslední definice.

Definice parametru má tvar

```
jméno_parametru  typ hodnota
```

kde *jméno_parametru* je jméno, pod jakým bude parametr zaregistrován, *typ* definuje typ parametru a *hodnota* definuje vlastní hodnotu parametru v závislosti na jeho typu. Podporované typu a odpovídající formát hodnoty ukazuje následující tabulka:

typ	*hodnota*
uint	Celočíselná hodnota.
double	Číselná hodnota s desetinnou čárkou následována odchylkou.
string	Libovolný řetězec ohraničený uvozovkami.

Číselné hodnoty s desetinnou čárkou následuje hodnota odchylky ve tvaru

```
var odchylka
```

kde *odchylka* je opět definována jako desetinné číslo.

5.7 Formát databáze materiálů

Druhým souborem, který program *Viewer* načítá při spuštění, je soubor obsahující databázi materiálů. Podobně jako konfigurační soubor je databáze materiálů uložena v textovém formátu a rozdělena na sekce. Jedna sekce odpovídá definici jednoho materiálu.

Definice materiálu má tvar

```
jméno_materiálu {
    attributy...
}
```

kde *jméno_materiálu* definuje jméno, pod jakým bude materiál zaregistrován. Oddíl *attributy...* je část popisující atributy definovaného materiálu. Atributy se zapisují za sebou, pokud je stejný atribut zadán vícekrát, použije se poslední zadaná hodnota.

Specifikace atributu má tvar

```
jméno_atributu hodnota
```

kde *jméno_atributu* určuje atribut, jehož hodnota se bude měnit, a *hodnota* definuje vlastní hodnotu atributu. Některé atributy hodnotu nevyžadují. Seznam atributů zobrazuje následující tabulka:

atribut	hodnota
texture	Jméno souboru s obrázkem textury materiálu. Program využívá knihovny *SDLImage* pro načítání různých grafických formátů jako jsou *Jpeg*, *Truevision Targa, PNG* a další.
color	Základní barva materiálu. Zadávaná jako tři číselné hodnoty v rozmezí od nuly do jedné. Každý hodnota specifikuje intenzitu barevné složky v pořadí červená, zelená a modrá.
alpha_test	Příznak určující, že při vykreslování se má interpretovat alfa kanál textury. Všechny pixely, které mají hodnotu alfy menší než práh zadaný atributem *alpha_threshold*, nejsou vykreslovány.
alpha_threshold	Hodnota v rozmezí od nuly do jedné definující práh použitý při alfa testu.

two_size	Příznak pro oboustranné vykreslování. Pokud je atribut specifikován, polygony tohoto materiálu jsou interpretovány jako oboustranné a jsou tedy vykreslovány z obou stran.
two_side_lighting	Příznak pro oboustranný nasvětlovací model v případě oboustranného vykreslování. Pokud je atribut specifikován, každá strana polygonu je nasvětlována zvlášť. Pokud atribut není specifikován, je polygon nasvětlován pouze z jedné strany a při vykreslování opačné strany se použije spočítané nasvětlení z přední strany.
billboard	Příznak určující, že polygony tohoto materiálu jsou vykreslovány tak, aby jejich přední strana byla vždy natočena ke kameře.

5.8 Srovnání

Jedním z nejznámějších systému pro modelování stromů je komerční systém *SpeedTree*. Jedná se o kompletní řešení pro modelování a zobrazování stromů v reálném čase. Na rozdíl od Rose systému je *SpeedTree* zaměřen pouze na zobrazování stromů a implementuje vlastní chráněný model pro vytváření různých druhů stromů. Vlastní technologie umožňuje zobrazovat stovky a tisíce stromů v jediné scéně. Systém však neumožňuje vytvářet animace vývoje stromů v čase.

Lintermann a Deusenn společně se svou prací [13] vytvořil i program implemetující představený systém založený na principu skládání komponent. Z programu se později vyvinul komerční systém *XFrog*. Uživatelské rozhraní umožňuje jednoduše spojovat komponenty do grafu. Na rozdíl od Rose systému však model nepodporuje vývoj modelu v čase a nijak se nezabývá optimalizací výstupní gemometrie pro současné grafické akcelerátory.

Jiný komerční systém *OnyxTREE* představuje sadu aplikací zaměřujících se na modelování konkrétních rostlin a stromů (jehličnany, palmy a pod.). Systém umožňuje růst modelů odpovídající biologickému růstu v přírodě a simulaci větru, ale je určen pouze pro vytváření komplexních modelů a nezabývá se interaktivním zobrazováním.

V nekomerční sféře je těžké nalézt program, který by umožňoval stromy či rostliny modelovat a zobrazovat v rozumné kvalitě. Jedním z mála je program *POV-Tree*, který je založen na makru pro program *POV-Ray*. Zatímco Rose systém je systém, nad kterým lze implementovat různé modely a přístupy, makro implementuje konkrétní rekurzivní algoritmus. Výsledkem jsou sice kvalitní, ale velice složité modely. Makro nepodporuje vývoj modelu v čase.

Kapitola 6

Dokumentace Rose systému

Tato kapitola poskytuje stručný přehled o programovém rozhraní Rose systému a použitých datových strukturách pro rychlou orientaci v systému a zdrojových textech. Podrobnější popisy jednotlivých funkcí a modulů lze nalézt v komentářích přímo ve zdrojových textech.

Rose systém je implementován v jazyce C/C++ a využívá objektových vlastností tohoto jazyka. Komponenty a další části systému jsou implementovány jako třídy. Pro implementaci vlastních komponent se využívá dědičnosti.

Základ systému obsahuje obecný kód pro práci s komponentami, simulací vývoje a generování geometrie, kde byl brán zřetel na co největší přenositelnost mezi platformami. Systém byl vyzkoušen na platformě Windows, ale jelikož nevyužívá žádné specifické vlastnosti této platformy, pouze prostředky jazyka C/C++, neměl by být problém s kompilací např. na platformě Linux. Závislá na platformě je až aplikace, která Rose systém využívá. K aplikaci se systém linkuje jako externí knihovna. Aplikace *Viewer* je určena pro platformu Windows s využitím knihovny OpenGL. I zde byl brán zřetel na přenositelnost, proto pro inicializaci OpenGL a uživatelský vstup byla využita knihovna SDL.

6.1 Moduly

Zdrojový kód je rozdělen pro přehlednost do logických bloků — modulů. Každý modul obsahuje zdrojové texty k určité části systému a je uložen v samostatném adresáři.

- Molib

  Modul Molib je knihovna obsahující obecné datové struktury pro obecné využití. Modul obsahuje podporu pro ladění programu, práci s řetězci, obecné načítání textových konfiguračních souborů a třídy pro práci s matematickými strukturami. Modul obsahuje implementaci tří-rozměrných vektorů pro reprezentaci pozic a normálových vektorů, podporu pro matice velikosti 4×4 a práci s nimi. Pro základní datové struktury jako jsou pole, seznamy či asociativní pole je využívána knihovna *Standard Template Library*, která je součástí překladačů C/C++.

- Core

  Modul Core obsahuje jádro systému. Zde je definována bázová třída pro reprezentaci a správu komponent. Dále se zde nachází implementace slotů pro napojování komponent, simulační události a jejich rozvrh při simulaci vývoje, základní rozhraní systému a implementace parametrů, parametrických prostorů a jejich načítání z konfiguračních souborů.

- Turtle

  Modul Turtle obsahuje implementaci *Turtle* objektu pro průchod a zpracování grafu komponent a pro generování výstupní geometrie.

- Geometry

  Modul Geometry obsahuje kód pro práci s transformacemi, implementaci *Stem* objektu pro generování geometrie větví a stonků, a třídy pro konverzi výstupní geometrie z obecného formátu do formátu optimalizovaném pro OpenGL.

- Material

  Modul Material obsahuje správu materiálů, textur a načítání konfiguračního souboru s definicí existujících materiálů.

- Models

  Modul Models je místo vyhrazené pro implementace komponent. Zde se nachází implementace komponent modelu stromu z kapitoly 4.

- Viewer

  Modul Viewer obsahuje zdrojové kódy aplikace *Viewer* pro zobrazování vygenerované geometrie v OpenGL.

6.2 Jádro systému

Přístup k systému je zpřístupněn aplikaci pomocí třídy *System*. Po inicializaci systému je vytvořena jedna instance této třídy. Různá funkcionalita poskytována systémem je spravována správci konkrétních subsystémů jako je správce materiálů, správce parametrů a pod. Po inicializaci objekt *System* poskytuje reference na jednotlivé správce.

6.2.1 Komponenty

Základem pro implementaci komponent je bázová třída *Component*. Každá komponenta při inicializaci vyžaduje odkaz na objekt *System*, jehož je součástí a který využívá pro práci se subsystémy, a svou hodnotu úrovně komponenty.

Při inicializaci komponenta vytvoří automaticky primární a bázový slot. Vytváření a správa ostatních slotů je ponechána na vlastní logice komponenty.

Třída *Component* obsahuje mnoho pomocných funkcí:

- Funkce pro přístup k systému, jako je přístup ke správě parametrů nebo materiálů.
- Funkce pro získání odkazu na rodičovskou komponentu včetně odkazu na slot, ve kterém je komponenta zaregistrována.
- Funkce pro práci se sloty komponenty, jako je přístup k primárnímu či bázovému slotu, funkce pro vytváření nových slotů a funkce pro iteraci skrz existující sloty.
- Funkce pro napojení dceřinných komponent.
- Funkce pro simulaci vývoje, jako je plánování událostí a zjišťování globálního či lokálního času simulace.
- Funkce pro transformace vektorů ze souřadného systému rostliny do souřadného systému komponenty a zpět.
- Funkce pro generování geometrie a zpracování událostí popsaných níže, které systém volá při odpovídajících požadavcích.

6.2.2 Sloty

Každý slot je zpřístupněn pomocí třídy *Slot*. Každý slot obsahuje tyto položky:

- Odkaz na rodičovskou komponentu, která slot vlastní. Pomocí tohoto odkazu lze procházet graf komponent směrem ke kořenu.
- Seznam dceřinných komponent, které jsou právě napojeny na tento slot. Seznam lze využít pro průchod grafu komponent od kořenové komponenty do hloubky.
- Transformaci tohoto slotu, kterou je transformována geometrie všech dceřinných komponent. Transformace udržuje současně i inverzní matici této transformace.
- Parametrický prostor, který udržuje parametry uložené v tomto slotu. Pro minimální paměťové požadavky je parametrický prostor vytvořen až při registraci prvního parametru, neboť mnoho slotů nemusí mít přiděleny žádné parametry.

6.2.3 Simulace vývoje

O simulaci vývoje se stará objekt *GrowthManager*. Tento objekt spravuje všechny naplánované události, aktuální simulační čas, graf komponent (prototyp), jehož simulace se provádí, a stará se o vlastní doručování událostí jednotlivým komponentám.

Před započetím simulace je potřeba vytvořit vstupní graf komponent, který se bude vyvíjet. O dodání tohoto grafu se stará aplikace, například může být graf zadán uživatelem. Vstupní graf se poté zaregistruje do simulace pomocí funkce

```
ComponentReference void link( ComponentReference component ) ;
```

Tato funkce očekává kořenovou komponentu vstupního grafu. Všechny komponenty z celého grafu jsou pak zaregistrovány do simulace. Pokud je v simulaci zaregistrován jiný graf, je vrácena jeho kořenová komponenta. Pro odpojení grafu ze simulace slouží funkce

```
ComponentReference unlink( void ) ;
```

která vrací kořenovou komponentu grafu. Jelikož během simulace mohou komponenty jak vznikat tak zanikat, nemusí komponenta vracená touto funkcí odpovídat komponentě, která byla zaregistrována metodou *link()*.

Vývoj komponent v čase se provede funkcí

```
void simulace( const Time duration ) ;
```

která provede vývoj grafu v čase po zadanou dobu. Jakmile jsou všechny události ze zadaného času odsimulovány, funkce se ukončí. Aplikace poté může buď pokračovat v simulaci opětovným zavoláním funkce, nebo může z grafu komponent vygenerovat geometrii.

Pokud chce aplikace vygenerovat geometrii z grafu, je potřeba nejdříve celý graf aktualizovat pomocí funkce

```
void update( void ) ;
```

Tato funkce průchodem grafu provede aktualizaci všech transformací a interních stavů komponent tak, aby odpovídaly konečnému času simulace.

Po aktualizaci grafu jej lze odpojit ze simulace metodou *unlink()* a poté z něj vygenerovat geometrii. V simulaci vývoje grafu lze pokračovat jeho opětovným zaregistrováním pomocí funkce *link()*.

6.2.4 Události

Aby bylo možno komponenty připojovat a odpojovat ze simulace a přitom nedošlo ke ztrátě naplánovaných událostí, jsou události ukládány vždy u cílové komponenty. Každá komponenta si udržuje rozvrh naplánovaných událostí. Při zaregistrování grafu do simulace, objekt *GrowthManager* zanalyzuje naplánované události všech komponent a vytvoří globální rozvrh, který určuje, kdy se musí zpravovat které rozvrhy událostí z jednotlivých komponent.

Při simulace vývoje je vyzvednuta komponenta s nejbližší událostí, provede se posun času na čas odpovídající této události, a událost se doručí cílové komponentě. Poté jsou aktualizovány všechny rozvrhy.

Každá událost obsahuje identifikátor určující o jakou událost se jedná, její typ, a nepovinné atributy *tag* a *data*. Atribut *tag* je libovolné číslo, které může komponenta použít pro upřesnění události. Pokud jedno číslo nestačí, lze použít atribut *data*, který umožňuje spojit událost s libovolnou strukturou.

Rose systém podporuje dva typy událostí — lokální a statické. Lokální události jsou události, které jsou doručeny přímo komponentě. Pro zpracování události slouží virtuální funkce

```
void execute_local_event( const Event & event ) ;
```

Statické události jsou události, které nejsou doručeny přímo komponentě, ale místo toho je zavolána zadaná statická funkce, která dostane referenci jak na událost, tak na cílovou komponentu.

Při plánování události komponenta zadává kromě struktury *Event* také čas doručení události. Čas je zadáván relativně k aktuálnímu času jako zpoždění, za jaké bude událost doručena.

6.2.5 Správa materiálů

Databáze materiálů je spravována objektem *MaterialManager*. Tento správce udržuje informace o zaregistrovaných materiálech a použitých texturách.

Uvnitř systému je každý materiál identifikován pomocí *MaterialId*, což je unikátní číslo, které je přiděleno materiálu při jeho registraci. *MaterialManager* pak udržuje asociativní pole, které určuje mapování mezi *MaterialId* a objektem *Material*, který udržuje informace o daném materiálu. Komponenty a aplikace mohou pracovat s materiály nejenom pomocí *MaterialId*, ale materiály lze identifikovat také dle řetězce odpovídajícího jménu daného materiálu. *MaterialManager* udržuje asociativní pole, které obsahuje mapování mezi jmény materiálů a jejich identifikátory.

Podobně jako materiály i textury mají přidělen identifikátor *TextureId*, který jednoznačně identifikuje danou texturu v systému. Informace o textuře jsou uloženy ve třídě *Texture*. Obdobně lze textury identifikovat dle jejich jména. Správce materiálů udržuje odpovídající asociativní pole. Oddělení textur od materiálů umožňuje sdílet jednu instanci textury více materiály.

Správce materiálů umožňuje načíst databázi materiálů pomocí funkce

```
void read_materials( const char * const input_data ) ;
```

která zaregistruje do systému všechny materiály a textury ze zadaného řetězce. Pokud je potřeba načíst databázi ze souboru nebo z jiného zdroje, musí se daný soubor načíst do paměti a funkci předat odkaz na tuto paměť. Díky tomu je jádro systému udržováno nezávislé na platformě a konkrétním uložení vstupních dat. Textový formát databáze je popsán v kapitole 5.7.

6.3 Parametrický subsystém

Správa parametrů je koncentrována v objektu *ParameterManager*. Tento objekt existuje jediný pro celý systém a udržuje tyto informace:

- Přidělování identifikátorů parametrům a mapování ze jména parametru na odpovídající identifikátor *ParameterId*.

  Správce parametrů udržuje tabulku všech jmen parametrů a přiděluje jim jednoznačné identifikátory. Všechny parametry pak interně pracují pouze s tímto identifikátorem.

- Globální parametrický prostor.

  Globální parametrický prostor obsahuje všechny globální parametry, které může využívat nejenom kterákoli komponenta, ale i samotná aplikace, která Rose systém využívá.

- Parametrické prostory pro jednotlivé úrovně komponent.

  Pro každou zaregistrovanou úroveň udržuje správce parametrů odpovídající parametrický prostor, který obsahuje všechny parametry pro danou úroveň. Komponenty z dané úrovně mohou tyto parametry jednoduše využívat.

Správce parametrů umožňuje načíst konfiguraci parametrů pomocí funkce

```
void read_parameters( const char * const input_data ) ;
```

která zaregistruje do systému všechny parametry ze zadaného řetězce. Podobně jako u správce materiálu, pokud je potřeba načíst databázi ze souboru nebo z jiného zdroje, musí se daný soubor načíst do paměti a funkci předat odkaz na tuto paměť. Textový formát konfiguračního souboru je popsán v kapitole 5.6.

6.3.1 Parametry

Parametry jsou implementovány bázovou třídou *Parameter* a z ní zděděných tříd *NumberParameter*, *UnsignedIntegerParameter* a *StringParameter*, které implementují odpovídající typy parametrů. Každý parametr je vlastněn nějakým parametrickým prostorem. Parametrické prostory implementuje třída *ParameterSpace*. Instance této třídy udržují asociativní pole parametrů, které jsou v daném parametrickém obsaženy, a obsahují funkce pro registraci parametrů a jejich rychlé vyhledávání.

Pro přístup k parametrům komponenty nevyužívají přímo objekty typu *Parameter*, ale specializované objekty s bázovou třídou *ParameterValue*. Zatímco objekty typu *Parameter* slouží pro uložení vlastních dat parametrů a jejich propojení se systémem, objekty typu *ParameterValue* slouží jako zástupné objekty pro přístup k odpovídajícím parametrům. Objekt typu *ParameterValue* představuje chytrou referenci na parametr, která skrývá implementační detaily, kde se odpovídající parametr nachází, a automaticky se stará o jeho vyhledávání. Pomocí operátorů přetypování v jazyce C/C++ mohou komponenty pracovat s parametry jako s běžnými typy.

Pokud komponenta potřebuje přístup k nějakému parametru, vytvoří odpovídající objekt *ParameterValue* a nainicializuje jej pomocí jména odpovídajícího parametru, lokace parametrického prostoru a hodnoty, která je použita, pokud není parametr nalezen. Lokace parametrického prostoru je symbolické jméno, které určuje, ve kterém parametrickém prostoru se má zadaný parametr hledat. Jak bylo zmíněno v kapitole 2.2, jsou poskytnuty tři možné lokace:

- Globální pro přístup ke globálnímu parametrickému prostoru.

- Přístup k parametrickému prostoru, který odpovídá úrovni komponenty.
- Parametrický prostor uložený ve slotu směrem ke kořenu grafu komponent.

```
class MyComponent : public Component {
    NumberParameterValue parent_length ;
}
void MyComponent::initialize_parameters( void )
{
    parant_length.initialize(
        this,
        ‘‘ParentLength’’,
        PARAMETER_LOCATION_BRANCH_SLOT,
        1, 0
    ) ;
}
double MyComponent::compute_length( void )
{
    // Spočítej poměr délky k rodičovské komponentě.
    const double ratio = ... ;
    // Parametr lze používat jako běžný číselný typ.
    return ( parent_length * ratio ) ;
}
```

Příklad práce s parametrem *ParentLength* v C/C++. Parametr je vyhledáván ve slotech směrem ke kořenu grafu.

6.4 Generování geometrie

Pro generování geometrie z grafu komponent slouží třída *Turtle*. Tato třída obsahuje základní prostředky pro průchod grafu komponent a definuje rozhraní, které komponenty mohou použít pro definici geometrie. Implementací tohoto rozhraní lze vytvořit různé instance třídy *Turtle* pro různé výstupní formáty.

Základní funkčností, kterou třída *Turtle* poskytuje, je průchod grafu komponent a práce s transformacemi. Objekt udržuje zásobník transformací, pomocí kterého implementuje průchod grafu do hloubky. Při průchodu grafu je potřeba také ukládat kromě transformací i aktuální stav objektu. K tomu slouží virtuální funkce

```
void push_state( const uint num_times ) ;
void pop_state( void ) ;
```

které jsou volány průběžně při průchodu grafem komponent. Funkce *push_state()*, která slouží pro uložení stavu objektu na zásobník, dostává argument *num_times*,

který označuje, kolikrát se má stav na zásobník uložit. Toho se využívá při průchodu slotu, kdy je potřeba stejný stav uložit pro každou komponentu, která je na slot napojena. Aby se neplýtvalo pamětí, lze na zásobník uložit stav pouze jednou a poznamenat si počet, kolik záznamů představuje.

V Rose systému je implementována třída *GeometryTurtle*, která slouží pro vytvoření polygonové geometrie z grafu komponent. Výstupem je graf geometrických uzlů, kde jsou bloky geometrie seskupeny do fragmentů podle materiálů. Pro převod geometrie do formátu pro OpenGL lze pak využít třídu *OpenGLGeometry*, která vstupní graf geometrických uzlů s fragmenty převede do pole vrcholů, indexů a dávek.

Kapitola 7

Závěr

V této práci jsem navrhl a vytvořil obecný systém pro modelování rostlin a stromů. Systém je založen na komponentovém přístupu, který umožňuje nejen dívat se na modelování stromu jako na skládanku z mnoha samostatných celků, ale díky zakomponování pojmu času a vývoje jako nedílné součástí systému vychází vstříc modelování biologických pochodů, které v rostlinách probíhají. Ani jeden úhel pohledu však neomezuje a oba přístupy lze využívat nezávisle. Systém tak kombinuje to nejlepší z interaktivních přístupů [13], ze kterých převzal ideu komponent a jejich spojování a přiblížil tak modelování rostlin více programátorskému úhlu pohledu, a z L-systémů [12], ve kterých se inspiroval pro začlenění času do simulace a přenosu parametrů mezi komponentami.

Systém se podařilo navrhnou modulárně, díky čemuž je dobře odděleno nezávislé jádro celého systému, implementace vlastních komponent a vlastní generování geometrie. Systém je tak připraven na implementaci nejrůznější škály komponent, které mohou vycházet z různých přístupů k problematice modelování rostlin a stromů.

V Rose systému jsem naprogramoval model stromu, který vychází z práce [6], je založen na geometrických pozorování a je rozšířen o možnost vývoje stromu v čase. Model lze konfigurovat mnoha parametry pro docílení nejrůznějších tvarů a po uživateli nejsou požadovány botanické znalosti. Vygenerované modely lze prohlížet v reálném čase pomocí aplikace *Viewer* za využití rozhraní OpenGL a i samotné vygenerování geometrie z modelu trvá velmi krátce, typicky v řádu sekund.

Konfigurací parametrů lze dosahovat různé detailnosti a tedy i náročnosti vygenerované geometrie. Přestože i složitější model stromu lze s optimálním využití prostředků poskytovaných současnými grafickými kartami zobrazovat v reálném čase rychlostí desítek snímků za sekundu, při velkém počtu stromů v jedné scéně přestává hrubá síla grafických karet stačit.

Obecnost systému však přináší i jeho nedostatky. Přestože vygenerovanou geometrii lze efektivně zobrazovat a konfigurací parametrů lze dosahovat různé kvality modelů, množství polygonů je stále příliš velké pokud bychom chtěli zobrazovat např. les obsahující desítky, stovky či tisíce stromů. Zajimavým směrem by bylo navržení nebo vyzkoušení algoritmů, které by umožňovaly

měnit složitost vykreslované geometrie dynamicky v reálném čase. Zvlástě speciální postupy pro redukci počtu listů by mohly přinést citelné vylepšení výkonu. Je možné, že by tyto algoritmy vyžadovaly změny ve správě komponent a přinesly omezení v napojování komponent a jejich komunikaci.

Jiným vylepšením by mohla být nádstavba v nějakém objektovém skriptovacím jazyce pro jednodušší definici nových komponent. Jazyk v C/C++ je ideální pro implementaci jádra systému, při tvorbě komponent však programátor naráží na technické detaily, kdy k vyjádření jednoduchých principů je potřeba napsat mnoho kódu.

Literatura

[1] Akenine-Möller T., Haines E. (2002): Billboarding. *Excerpt from Real-Time Rendering, 2nd edition.* http://www.flipcode.com/articles/article_rtr2billboards.shtml

[2] Décoret X., Durand F., Sillion F., Dorsey J. (2003): Billboard clouds for extreme model simplification. *In Proceedigs of the ACM SIGGRAPH.* ACM Press.

[3] Max N., Ohsaki K. (1996): Rendering Trees from Precomputed Z-buffer views. *Eurographics Worshop on Rendering,* 165-174.

[4] Jakulin A. (2000): Interactive Vegetation Rendering with Slicing and Blending. *Proceedings Eurographics 2000 (Short Presentations).* Eurographics.

[5] De Reffye, P. (1988): Plant Models Faithful to Botanical Structure and Development. *ACM SIGGRAPH Computer Graphics* 22, 151–158.

[6] Weber J., Penn J. (1995): Creation and Rendering of Realistic Trees. *ACM SIGGRAPH '95 Conference Proceedings*, 119–128.

[7] Lindenmayer A. (1968): Mathematical Models for Cellular Interactions in Development, Parts I and II'. *Journal of Theoretical Biology* 18, 280–315.

[8] Curry R. (1999): On the Evolution of Parametric L-systems. *Computer Science Technical Reports.* University of Calgary.

[9] Měch R. (1997): Modeling and simulation of the interaction of plants with the environment using the L-systems and their extensions. *PhD disertace, University of Calgary.*

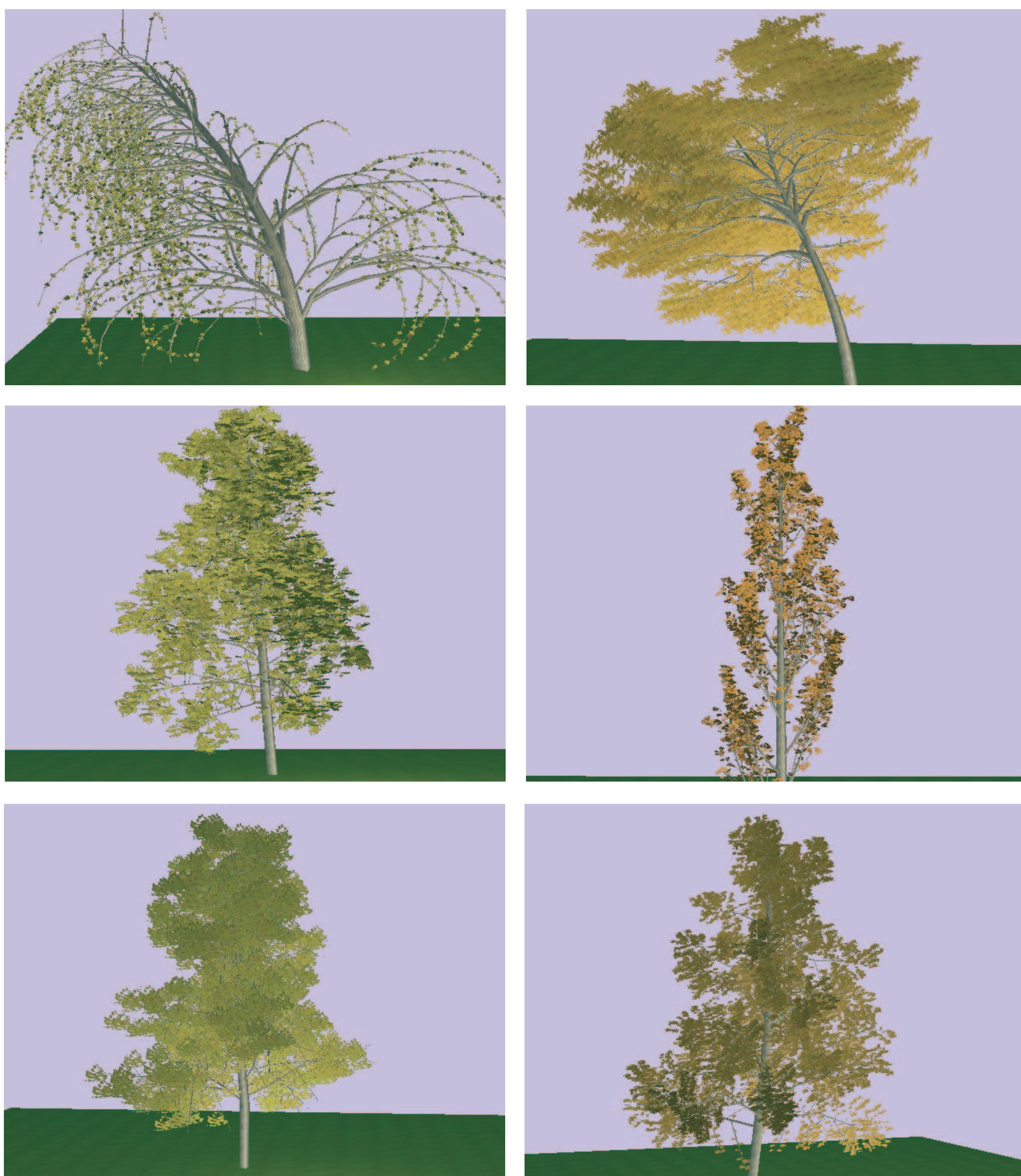
[10] Remolar I., Chover M., Belmonte O., Ribelles J., Rebollo C. (2002): Real-Time Tree Rendering. *Technical report, Departmento de Lenguajes y Sistemas Informaticos, Universitas Jaume I, Campus de Riu Sec, E-12080.*

[11] Prusinkiewicz P., Lindenmayer A. (1990): The Algorithmic Beauty of Plants. *Springer-Verlag, New York.*

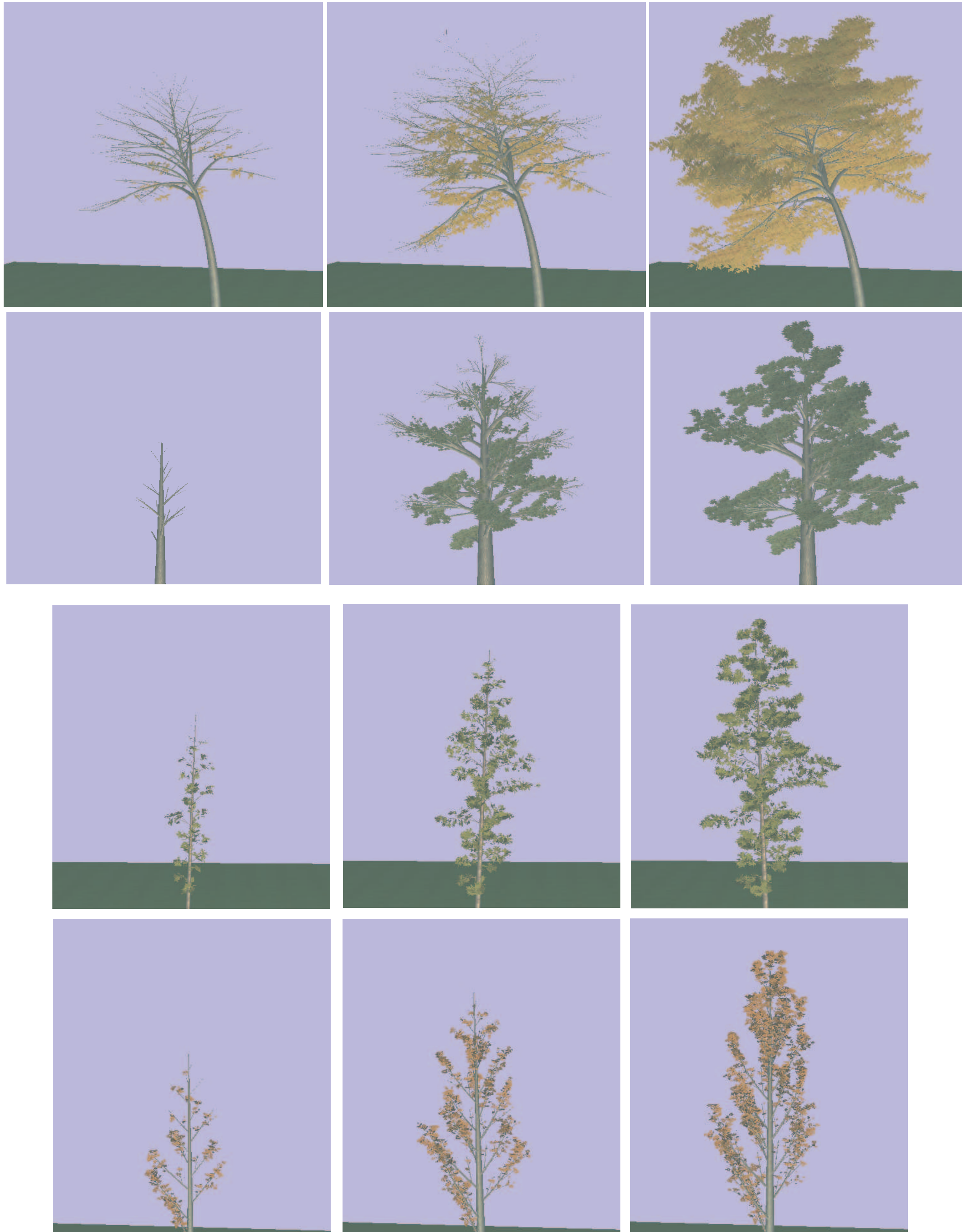
[12] Prusinkiewicz P., Hammel M., Mjolsness E. (1993): Animation of Plant Development. *ACM SIGGRAPH Computer Graphics* 22, 351–360.

[13] Lintermann B., Deussen O. (1999): Interactive Modeling of Plants. *IEEE Computer Graphics and Applications* 19(1), 56–65.

Obrázek 7.1: Modely stromů vytvořené programem *Viewer* a zobrazené pomocí rozhraní OpenGL.



Obrázek 7.2: Modely stromů vytvořené programem *Viewer* a zobrazené pomocí rozhraní OpenGL.



Obrázek 7.3: Animace vývoje vytvořené programem *Viewer*.