VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MINIMALISTICKÁ REPREZENTACE MODELU AREÁLU BOŽETĚCHOVA

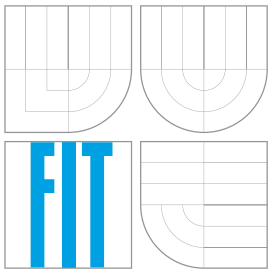
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTĚPÁN ROSA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MINIMALISTICKÁ REPREZENTACE MODELU AREÁLU BOŽETĚCHOVA

MINIMALISTIC REPREZENTATION OF THE BOŽETĚCHOVA COMPLEX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠTĚPÁN ROSA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL VYSKOČIL

BRNO 2007

Zadání bakalářské práce/5146/2006/xrosas00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Rosa Štěpán**
Obor: Informační technologie
Téma: **Minimalistická reprezentace modelu areálu Božetěchova**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s problematikou tvorby grafických aplikací s omezenou velikostí.
2. Prostudujte techniky komprese modelů vhodné pro reprezentaci těles v grafických aplikacích omezené velikosti.
3. Vytvořte model areálu Božetěchova v úrovni detailu vhodné pro minimalistické uložení.
4. Implementujte program zobrazující minimalistickou reprezentaci daného modelu.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; prezentujte projekt plakátkem.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- body 1.-3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vyskočil Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Štěpán Rosa**
Id studenta: 84166
Bytem: Novosady 881/113, 594 01 Velké Meziříčí
Narozen: 29. 05. 1985, Velké Meziříčí - Mostiště
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Minimalistická reprezentace modelu areálu Božetěchova
Vedoucí/školitel VŠKP: Vyskočil Michal, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato práce se zabývá vytvořením a zobrazením minimalistického modelu areálu Božetěchova pomocí programovacího jazyka C s využitím knihovny OpenGL a její nadstavby GLUT. Na začátku práce je teoretický základ, který vedl ke zvládnutí této problematiky. V následujících kapitolách je popsán vytvořený model a cesta k jeho vzniku.

Klíčová slova

počítačová grafika, OpenGL, model, omezená velikost, procedurální textury

Abstract

This work concerns about creation and display minimalistic model of the Božetěchova Complex implemented in the programming language C using graphical libraries OpenGL and GLUT. In the introduction of this thesis is theoretical base, that led to making this issue. The following chapters describe implemented model and the way of its creation.

Keywords

computer graphics, OpenGL, model, limited size, procedural textures

Citace

Štěpán Rosa: Minimalistická reprezentace modelu areálu Božetěchova, bakalářská práce, Brno, FIT VUT v Brně, 2007

Minimalistická reprezentace modelu areálu Božetěchova

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Vyskočila.

. . . . . .
Štěpán Rosa
14. května 2007

Poděkování

Děkuji tímto vedoucímu bakalářské práce Ing. Michalu Vyskočilovi za metodické vedení, ochotnou spolupráci a cenné rady při zpracování bakalářské práce.

Obsah

Kapitola 1

Úvod

Člověk má pět smyslů. Jedním z nich je zrak, díky kterému vnímáme prostor, tvary, předměty, písmo, lidi. Je pro nás velmi důležitý. V mozku si promítáme obraz svých nejbližších, vzpomínky, ale i sny a představy.

Už v pravěku měl člověk potřebu obrazy ve své mysli nějak reflektovat. Kreslil uhlíkem po stěnách jeskyně, modeloval sošky. Postupem času se člověk vyvíjel a spolu s ním také prezentace jeho myšlenek a představ. V roce 1960 se poprvé objevuje slovo počítačová grafika.

S touto oblastí informatiky se setkáváme denně ani si toho nemusíme být vědomi. Když si prohlížíme nebo upravujeme fotografie v počítači, když hrajeme hry nebo se díváme na skvělé efekty filmů posledních let či nejrůznější počítačové animace. Hojně se využívá v průmyslu a v lékařství.

Jak je uvedeno v [18], počítačová grafika je obor, který se zaměřuje na vytváření umělých snímků a také na úpravu zobrazitelných a prostorových informací, nasnímaných z reálného světa. Počítačová grafika byla již od svých počátků oblastí, ve které se uplatnili lidé s různými dovednostmi a zájmy, ať už to byly programátoři, grafici nebo inženýři.

V současné době výpočetního výkonu a kapacit pamětí se stále setkáváme s velkou skupinou lidí po celém světě, kteří tvoří grafické aplikace s omezenou velikostí. Mohlo by se zdát, že do takové aplikace, která zabírá okolo 4 KB až 10 MB na pevném disku, se příliš mnoho grafiky nevejde. Sám jsem byl překvapen, jakých efektů lze dosáhnout v tak malém programu. Podrobnosti naleznete v [4].

Ještě před pár lety bylo samozřejmostí programování těchto aplikací v assembleru. Dnes se assembler používá zřídka, namísto toho se tvoří pomocí vyšších programovacích jazyků např. jazyk C/C++.

Úspory místa lze docílit několika způsoby. Například tím, že aplikace nebude obsahovat předpočítaný obraz (ten se spočítá až po startu aplikace), nebo kompresí spustitelného souboru či kompresí textur. Touto problematikou se zabývá 3. kapitola, která rovněž čtenáře seznámí s grafickou knihovnou OpenGL a její nadstavbou knihovnou GLUT.

Kapitola 2 obsahuje formulaci cíle práce.

4. kapitola popisuje etapy vývoje, současný stav a architekturu vytvořeného modelu.

V 5. kapitole je vysvětlen návrh a konečná implementace modulů a funkcí, jež vedly k vytvoření modelu a programu zobrazujícího model.

V závěru je uvedeno shrnutí a možnosti pokračování v projektu. Tato práce navázala na předchozí semestrální projekt.

Kapitola 2

Formulace cíle práce

Cílem práce je vytvořit model areálu Božetěchova s takovou úrovní detailů, která je vhodná pro minimalistické uložení a následnou implementaci. K tomu je nutné seznámit se s problematikou tvorby grafických aplikací s omezenou velikostí a prostudovat techniky komprese modelů, vhodné pro reprezentaci těles v těchto aplikacích, dále nastudovat programování grafických aplikací pomocí knihovny OpenGL a její nadstavby GLUT.

Na základě získaných zkušeností provést návrh a implementaci. Poté zhodnotit dosažené výsledky a navrhnout možnosti pokračování projektu.

Kapitola 3

Teoretická a odborná východiska

3.1 Progressive Meshes

Při zobrazování grafiky ve vysokém rozlišení jsou modely reprezentovány velkými sítěmi trojúhelníků, což je náročné na výkon a paměť.

Progressive meshes je bezztrátová technika schopná zjednodušit libovolnou síť trojúhelníků, přičemž uchová nejen samotnou geometrii sítě, ale i identifikátor materiálu, normály, barvy i texturovací souřadnice.

Pro zrychlení vykreslování je běžné mít model v několika úrovních detailu (LOD). Při malých vzdálenostech kamery od modelu je použita detailnější síť. Jakmile se kamera vzdaluje od obrazovky, není potřeba tak detailní vykreslení, a zobrazí se hrubší model.

Princip je založen na tom, že hrana spojující dva vrcholy se nahradí jedním vrcholem a tím jsou také zrušeny dva polygony, které hrana spojovala. Důležité je, že tato operace je invertovatelná. Detaily v [1].

3.2 Procedurální texturování

Procedurální textury jsou procedury, jejichž zavoláním získáme hodnotu textury v daném bodě. Je vhodné je použít pro opakující se vzory, např. cihly, ale také pro efekty pohybu, např. oheň, mraky nebo přírodní materiály, např. dřevo, kamení, tráva. V kombinaci s šumem a turbulencí vytváří realistické textury.

Jejich velkou výhodou je, že procedury ve srovnání s bitmapovými obrázky zabírají mnohem menší místo. Mohou pokrývat neomezené prostory, nemají pevné rozlišení. Jsou parametrizovatelné, a proto lze změnou parametru generovat velké množství variací. Nevýhodou je, že jejich výpočet může být časově náročný a naprogramování není triviální záležitostí. Může dojít k alisu a jeho odstranění nemusí být jednoduché. Podrobnosti v [6].

3.3 UPX

UPX (Ultimate Packer for eXecutables) je výkonný kompresní program, který se používá pro kompresi spustitelných souborů. Při kompresi je přidán do souboru kód, který se postará o dekompresi a spuštění souboru. Detaily naleznete v [3].

3.4 OpenGL

Knihovna OpenGL (Open Graphics Library) [7], [8] je aplikační programové rozhraní grafického hardwaru. Byla vytvořena tak, aby byla nezávislá na hardware a na použitém operačním systému. Důsledkem toho je, že pomocí ní můžeme vykreslovat pouze geometrická primitiva. Mezi ně patří bod, úsečka, trojúhelník, čtyřúhelník, plošný konvexní polygon, bitmapa a pixmapa. Lze také využít funkcí pro proudové vykreslování některých primitiv (polyčáru, pruh trojúhelníků, pruh čtyřúhelníků nebo trs trojúhelníků). Jednotlivá primitiva jsou definována pomocí vrcholů. OpenGL je stavový automat. Je možné jej uvádět do různých stavů, které pak zůstávají v platnosti, dokud nejsou změněny.

3.4.1 Syntaxe funkcí

Jména funkcí se skládají:

- prefix gl, pokud se jedná o OpenGL, glu pro GLU a glut pro GLUT
- tělo jména funkce označující požadovanou vlastnost nebo vytvářený předmět
- číslo označující počet parametrů funkce (pokud žádné nemá, nepíše se)
- jeden nebo dva znaky označující typ parametrů
- znak v, pokud je parametrem pole

Typ parametrů může být b, s, i, f, d, ub, us, ui. Příkladem je *glColor3f()*, která nastaví barvu a očekává tři parametry typu float. Nebo *glNormal3fv()*, která nastaví normálu a očekává tři parametry typu float uložené v poli zadaném jako argument. Detaily v [11].

3.4.2 Transformační matice

Transformace jsou zapsány ve formě matic. Mezi operace, které můžeme provádět patří rotace, posun, změna měřítka, zrcadlení, ortografická projekce a projekce s perspektivou. Násobení matic není komutativní, proto musíme dát pozor na pořadí zadávaných transformací.

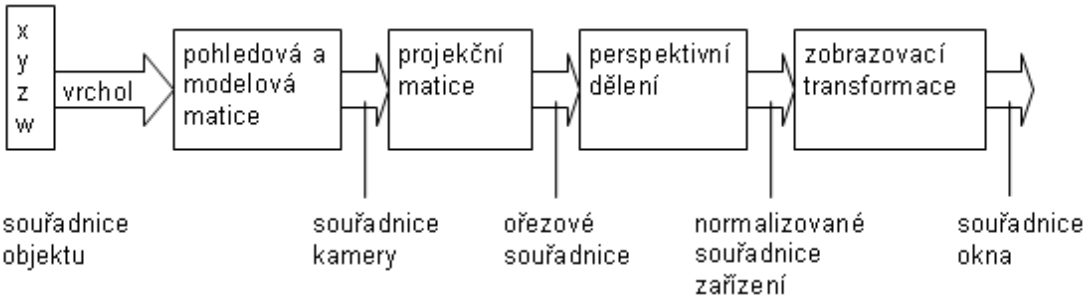
Transformace při vytváření scény můžeme přirovnat k fotografování. Postup:

1. Nastavíme stativ a zaměříme fotoaparát (pohledová transformace).
2. Připravíme scénu tak, aby odpovídala zamýšlené kompozici (modelovací transformace).
3. Vybereme objektiv nebo upravíme zoom (projekční matice).
4. Určíme, jak velká bude finální fotografie (zobrazovací transformace).

V OpenGL existují tři transformační matice, které se postupně (obr. 3.1) aplikují na body (vrcholy) popř. i na normály vrcholů. První z těchto matic se nazývá *ModelView matrix*. Tato matice je spojení modelové a pohledové matice, protože se používá jednak pro nastavení pozice kamery, jednak pro manipulaci s objekty (modely) ve scéně. Druhá transformační matice se jmenuje *Projection matrix* a používá se pro nastavení perspektivní projekce kamery. Konečně třetí transformační matice *Viewport matrix* je používána po provedení perspektivní projekce k mapování objektů z abstraktních souřadnic do souřadnic okna.

V OpenGL jsou transformace na vrcholy aplikovány v opačném pořadí než jsou zavolány jejich příslušné příkazy. Jestliže bude v programovém kódu sekvence posun, rotace, tak se ve skutečnosti

objekt nejdříve otočí a posléze posune. Na tuto skutečnost se lze dívat také tak, že se neprovádí transformace se samotným objektem, ale postupně s celým souřadným systémem. Nyní už ve správném pořadí.



Obrázek 3.1: Jednotlivé kroky transformace vrcholů

Transformační matice lze ukládat do zásobníku matic. Pro různé typy transformačních matic jsou vyhrazeny různé zásobníky s obecně odlišnou kapacitou. Největší kapacitu mívá většinou zásobník pro ModelView matici, protože ta se nejčastěji mění. Více informací také v [15].

Užitečné funkce

glMatrixMode(), glLoadIdentity(), glLoadIdentity(), glRotatef(), glPushMatrix(), glPopMatrix(), glViewPort(), gluPerspective()

3.4.3 Display-listy

Display-listy si můžeme představit jako makra, ve kterých je uloženo několik příkazů OpenGL. Tato makra pak můžeme spustit jedním příkazem. Jejich výhoda spočívá v tom, že se zvýší rychlost vykreslování a také se zjednoduší kód při vykreslování složitějších scén. Při tvorbě modelu areálu bude vytvořen display-list pro každou budovu a celá scéna potom bude vykreslena zavoláním těchto display-listů s různě nastavenou transformační maticí. Více informací v [16].

Užitečné funkce

glGenLists(), glNewList(), glEndList(), glCallList()

3.4.4 Pole vrcholů

Pole vrcholů [9] sníží počet volání funkcí a opakování již definovaných vrcholů[1]. Díky této skutečnosti klesne počet dat posílaných do grafického adaptéru po sběrnici a tím se zvýší výkon naší aplikace.

V jednotlivých polích mohou být souřadnice vrcholů, RGBA barvy, barevné indexy, normálové vektory, texturovací souřadnice a příznaky hrany. Před použitím je nutné dané pole zapnout funkcí *glEnableClientState()*.

Další krokem je specifikace dat pro pole. Odpovídající funkce jsou *glVertexPointer(), glColorPointer(), glTexCoordPointer()*... Funkce *glArrayElement()* získá data z jednoho vrcholu všech

[1] Např. u krychle, kde je 1 bod sdílen 3 stěnami.

zapnutých polí. Mnohem vhodnější je použití funkce *glDrawElements()*, kde jako jeden z parametrů zadáme pole, ve kterém budou uloženy indexy vrcholů v pořadí odpovídajícím jejich užití.

Užitečné funkce

gEnableClientState(), glVertexPointer(), glColorPointer(), glTexCoordPointer()

3.4.5 Osvětlení

V OpenGL je světlo rozloženo na červenou, zelenou a modrou složku. Barva zdroje světla je určena množstvím jednotlivých barevných složek, které vyzařuje. Materiál je vyjádřen částí barevných složek, které jsou odraženy do různých směrů.

Světlo dále můžeme rozdělit na ambientní, difúzní a odleskovou složku. Tyto složky jsou vypočítány oddělně a potom sečteny.

- ambientní - nelze určit jeho směr, je rozptýlené do všech stran
- difúzní - přichází z jednoho směru, avšak při dopadu na povrch se rovnoměrně rozptýlí do všech směrů
- odlesková - dopadá taktéž z jednoho směru, od povrchu se odráží jedním směrem
- vyzařovací - tato složka je pouze u materiálů, není uvažována jako světelný zdroj

Při vytváření světla je možné si vybrat mezi směrových světelným zdrojem, kdy se uvažuje, že zdroj je v nekonečnu a tedy že dopadající paprsky jsou rovnoběžné, a pozičním zdrojem, který má své pevné umístění ve scéně.

Světlo je možné nastavit tak, že jeho poloha bude neměnná a pořád bude vyzařovat z jednoho místa, ať je scéna jakkoliv natočená, nebo se bude pohybovat se scénou, či nezávisle na ní. Záleží na tom, kdy bude pozice světla nastavena.

Ve scéně je možné mít až 8 zdrojů světla, ale je nutné počítat s tím, že se sníží výkon aplikace. Nadefinovaná světla lze zapínat a vypínat podle potřeby, ale je třeba mít zapnuté osvětlení, jinak je použita aktuální barva a výpočet osvětlení se neprovádí.

Při výpočtu osvětlení je nutné mít definované normálové vektory. Normálový vektor je kolmý k povrchu. Pro nastavení se používá funkce *glNormal()*. Je důležité, aby velikost vektoru byla rovna jedné. Při plochém stínování se zadává normála pro polygon. Pokud je zadána pro každý vrchol, jedná se Gourandovo stínování. Změna měřítka a zkosení mění délku normály. Podrobněji v [10].

Užitečné funkce

glLightfv(), glLightModeli(), glShadeModel(),glMaterialfv(), glNormal()

3.4.6 Texturování

Motivace: Situace, kdy je potřeba ve své scéně vytvořit střechu. Střecha se skládá ze stovek až tisíců tašek. Kdybychom chtěli každou tašku „nakreslit", vzniklo by velké množství bodů a také programový kód by velmi narůstal.

Mnohem lepší řešení poskytuje použití texturování [12]. Stačí každou stěnu vykreslit jako jeden polygon a pouze tisíckrát zopakovat texturu tašky.

Použití texturování obecně přináší:

- úsporu paměti
- úsporu výpočetního výkonu
- zjednodušení kódu

Texturování je proces, při kterém se na vykreslovaný povrch tělesa při rasterizaci nanáší textury. Textury jsou obrazce nanášené na povrch těles. Nejčastěji jsou to dvojrozměrné obrázky (pixmapy), ale OpenGL podporuje i jednorozměrné a trojrozměrné.

Texturu je možno vytvořit použitím rastrových obrázků (fotografie, oskenovaný obrázek), ale také procedurou(to přináší jisté výhody, avšak i nevýhody, viz 3.2). Pixely v textuře se nazývají texely.

Postup při texturování:

1. vytvoření nebo načtení rastrové předlohy ze souboru
2. vytvoření texturovacího objektu
3. přiřazení textury
4. nastavení způsobu nanášení textury
5. zapnutí nanášení textur
6. zadání texturovacích souřadnic pro každý vrchol
7. vykreslení scény

Texturování je podporováno pouze u true-color barevných režimů. Textury musí mít rozměry, které jsou mocninami čísla 2. Detailněji v [13].

Užitečné funkce

glGenTextures(), glBindTexture(), glPixelStorei(), glTexParameteri()

3.4.7 Mipmapping

Motivace: V dynamické scéně při změně vzdálenosti texturovaného objektu je třeba také měnit velikost textury.

Aby se zabránilo vzniku nežádoucích efektů v určitých fázích přechodu při filtrování textury, používá se předfiltrovaných textur se zmenšujícím se rozlišením, tzv. mipmap. Tyto textury je možné vytvořit programově s použitím různých filtrů nebo využít funkce pro automatické generování mipmap, které obsahuje nadstavbová knihovna GLU. Informace naleznete v [14].

Užitečné funkce

gluBuild2DMipmaps()

3.4.8 Blending

Motivace: Jak vykreslit průhledné objekty, či udělat ze čtverce při nanesení textury kruh?

Blending [17] neboli míchání barev využívá čtvrtou barevnou složku nazývanou Alfa. Touto složkou lze specifikovat míru průhlednosti vybraných objektů. Blending kombinuje fragmenty[2] s hodnotami ve framebufferu podle míchací rovnice definované uživatelem. V průběhu míchání jsou barvy fragmentu, právě vzniklého rasterizací (zdroj) kombinovány s hodnotami barvy odpovídajícího pixelu uloženého ve framebufferu (cíl). Koeficienty míchání lze nastavit pro jednotlivé barevné složky zvlášť.

Při blendingu je třeba specifikovat míchací funkci a povolit míchání. Nejprve by se měly vykreslit všechny neprůhledné objekty a posléze průhledné objekty nejhlouběji ve scéně, které je nutné ručně seřadit.

Postup pro vytvoření kruhu:

1. vytvoření textury (kruh bude mít složku alfa rovnou jedné, zbytek nule)
2. zapnutí blendingu
3. nastavení zdrojového faktoru na GL_SRC_ALPHA,
   cílového na GL_ONE_MINUS_SRC_ALPHA
4. vykreslení čtverce s nanesenou texturou

Užitečné funkce

glBlendFunc(), glDepthMask()

3.5 GLUT

Knihovna GLUT (OpenGL Utility Toolkit) je stejně jako OpenGL nezávislá na operačním systému. Tvoří rozhraní pro tvorbu oken a jednoduchého grafického uživatelského rozhraní. Vytvoření okna a zaregistrování funkcí obsluhujících události je relativně jednoduché (viz 5.4.1).

GLUT umožňuje vykreslit základní trojrozměrná tělesa jako krychli, kouli, kužel, torus, čtyřstěn, osmistěn, dvanáctistěn, dvacetistěn a také čajník. Obsahuje jednoduché rozhraní pro vykreslování znaků.

Mezi nevýhody patří především to, že nejdou vytvořit složitější prvky grafického uživatelského rozhraní např. tlačítka, výběrové boxy, scrollbary atd.

3.6 Kreslení do rastru

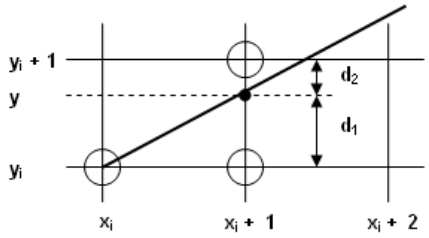
3.6.1 Bresenhamův algoritmus pro rasterizaci úsečky

Nejčastěji používaný algoritmus. Je velmi efektivní díky tomu, že používá pouze celočíselnou aritmetiku. Následující text předpokládá, že úsečka leží v prvním kvadrantu a nejrychleji roste ve směru osy x. Ostatní polohy je třeba převést na tento případ.

[2] Fragment je datová struktura složená z barvy pixelu, jeho průhlednosti, vzdálenosti od pozorovatele a případně dalších informací.

Princip:

Vykreslí se levý koncový bod úsečky. Inkrementuje se souřadnice x. O tom, zda souřadnice y zůstane nezměněna nebo se také inkrementuje, rozhodne znaménko prediktoru. Pokud je znaménko kladné, je skutečná hodnota blíže pixelu o souřadnici $y_i + 1$, jinak souřadnici y_i. Situace je znázorněna na obrázku 3.2. Takto se postupuje až k pravému koncovému bodu. Detailnější informace v [5], [2].



Obrázek 3.2: Část úsečky v rastru

Algoritmus:

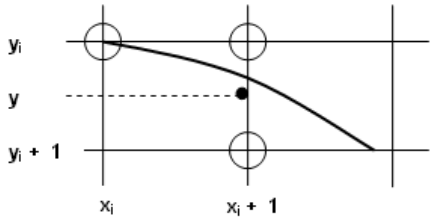
```
DrawLine(int x1, int y1, int x2, int y2)
{
   // pomocné proměnné
   int dx = x2 - x1, dy = y2 - y1;
   int k1 = 2 * dy,  k2 = 2 * (dy - dx);
   int P = 2 * dy - dx;
   int x = x1, y = y1;

   // dokud není pravý koncový bod
   while(x <= x2)
   {
      putPixel(x, y); // vykresli pixel
      x++;            // inkrementace x
      if(P <= 0)      // prediktor záporný
        P += k1;
      else            // prediktor kladný
      {
        y++;          // inkrementace y
        P += k2;
      }
   }
}
```

3.6.2 Bresenhamův algoritmus pro rasterizaci kružnice

Používá se také celočíselná aritmetika. Algoritmus využívá toho, že kružnice je symetrická. Stačí tedy spočítat hodnoty souřadnic bodů v první polovině prvního oktantu, tj. v úseku od $x = 0$ do $x = y$. Ostatní body se získají záměnou souřadnic. Následující text předpokládá, že střed kružnice leží v počátku souřadného systému.

Princip:

Vykreslí se bod o souřadnicích $[0, r]$. Inkrementuje se souřadnice x. O tom, zda souřadnice y zůstane nezměněna nebo se dekrementuje, rozhodne znaménko prediktoru. Pokud je znaménko kladné leží midpoint[3] mimo kružnici, blíže je tedy pixel $y_i + 1$, a proto se y dekrementuje, jinak zůstává nezměněno. Situace je znázorněna na obrázku 3.3. Takto se postupuje dokud není $x = y$. Detailnější informace v [5], [2].



Obrázek 3.3: Část kružnice v rastru

Algoritmus:

```
DrawCircle(int sx, int sy, int r)
{
   // pomocné proměnné
   int x = 0, y = R;
   int P = 1 - R, k1 = 3, k2 = 2 * r - 2;

   while(x < y)
   {
      putPixel(x, y); // vykresli pixel
      if(P >= 0)      // prediktor kladný
      {
        P += - k2;
        k2 -= 2;
        y --;
      }
      P += k1;
      k1 += 2;
      x++;
   }
}
```

[3] Midpoint je bod ležící v polovině mezi y_i a $y_i + 1$.

Kapitola 4

Vývoj a současný stav programu

Program se vyvíjel podle tzv. spirálového modelu. Po nastudování potřebné teorie a formulace cíle práce se objevil první návrh řešení a poté vlastní implementace. Již od počátku tedy existovala použitelná verze, která se postupně vylepšovala, co se funkčnosti a použitelnosti týče.

4.1 Vývoj programu

Prvotní verze

Na počátku byl model velmi jednoduchý. Byl součástí programu pro zobrazení modelu. Pole vrcholů pro jednotlivé budovy obsahovaly souřadnice bodů. Rozměry budov byly nepřesné, poněvadž v té době ještě nebyly k dispozici plány areálu. Vykreslení budovy bylo provedeno v display-listu, kde pro každou stěnu byla nastavena normála a stěna se vykreslila funkcí *glDrawElements()*.

Model bylo možno natáčet, přibližovat a oddalovat pomocí myši.

Druhá verze

Došlo k oddělení modelu a programu pro zobrazení. Protože při vykreslování v display-listech docházelo k opakování kódu, byly vytvořeny funkce *drawCuboid()* pro vykreslení kvádru a funkce *drawPrism()* pro vykreslení střechy, které tento nedostatek odstranily.

Ruční počítání souřadnic a jejich ukládání do pole bylo zdlouhavé a snadno mohlo dojít k chybě, proto byla implementována funkce *calculateCuboid()*, která podle zadaných rozměrů a počátečního bodu spočetla souřadnice a normálové vektory a uložila je do pole. Podobná funkce byla vhodná i pro vypočítání střechy. Vznikla funkce *calculatePrism()*.

Model v počáteční a druhé fázi je zobrazen na obrázku 4.1.

Třetí verze

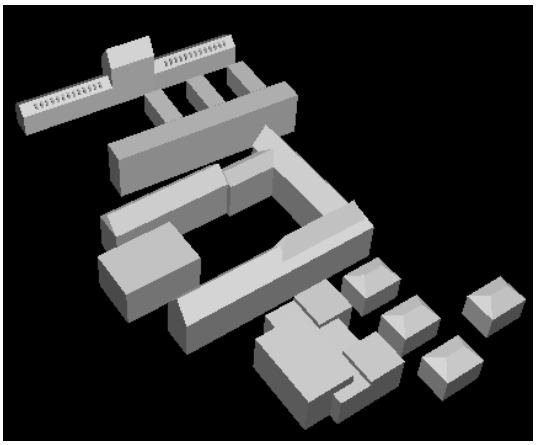
V této fázi byly dostupné plány areálu. Došlo ke korekci rozměrů a rozmístění budov v modelu. Dále byla provedena oprava výpočtu normál u funkce *calculatePrism()*. Vznikla funkce *calculateOctagon()*, která spočítá a uloží potřebně informace pro vykreslení osmibokého hranolu a funkce *calculatePyramid()* určená pro osmistěnný jehlan.

Hrubý model areálu byl hotov, byly započaty práce se zvyšováním detailů[1]. Vznikly variace funkce *calculateWinodow()* pro rovnoběžnost oken ve směru x, y a z.

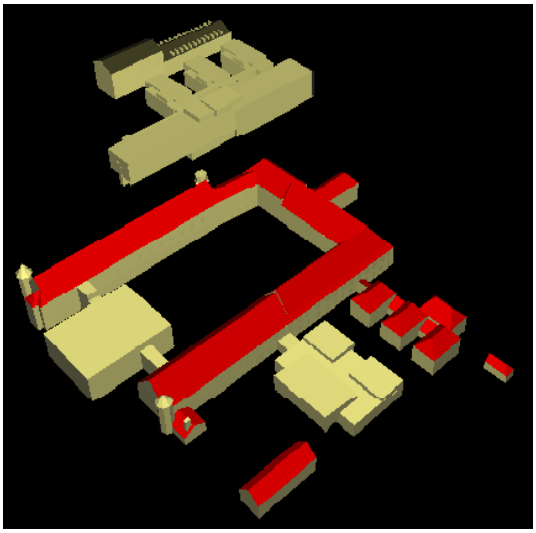
[1] Těmito detaily byla okna.

Při vykreslování oken docházelo k jejich častému opakování v řadě za sebou. Měnila se jen jedna či dvě souřadnice a bylo zbytečné každé okno počítat znovu. Byla naprogramována funkce *myTranslate()* a *myRotate()*. Poté se souřadnice vypočítaly jen jedenkrát a posléze byly v cyklu posunuty nebo otočeny.

Model je vidět na obrázku 4.2.



Obrázek 4.1: První verze modelu



Obrázek 4.2: Pokročilá verze modelu

Předposlední verze

Byly definovány materiály a funkce pro vytvoření textur a s nimi související funkce pro vykreslení úsečky, vyplněného obdélníku a kruhu do pole tvořícího texturu. Dále bylo rozšířeno ovládání programu o posouvání modelu pomocí šipek.

Finální verze

Byly provedeny konečné úpravy a byly naneseny textury. Místy došlo k předělání modelu. Byla vytvořena nápověda.

4.2 Charakteristika současného stavu

Popis

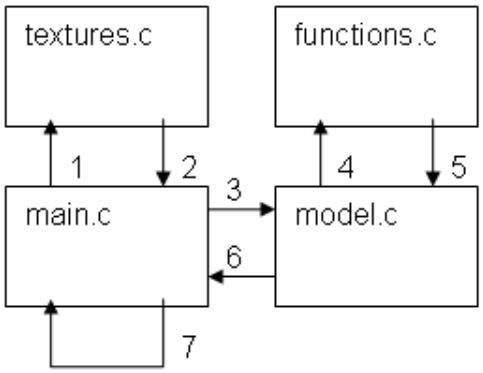
Současný stav modelu je na obrázku 4.4 a 4.5. Model je osvětlen jedním světelným zdrojem se stacionární polohou. Je zvoleno ploché stínování. Je definováno 5 materiálů a 31 mipmapových textur, které jsou vytvořené procedurálně po spuštění programu a jsou v barevném režimu RGBA. Velikost mipmap nultého levelu se pohybuje od 4 pixelů na šířku, 1 pixelu na výšku do 32 pixelů na šířku a 64 pixelů na výšku. Používá se trilineární filtrování a modulace textur. Program byl otestován v prostředí Microsoft Windows i Linux.

Chování

Modelem je možné rotovat podle osy x a y, posouvat s ním doleva, doprava, nahoru a dolů, je možné také přibližovat a oddalovat pozici kamery. Model lze zobrazit s vyplněnými a otexturovanými polygony nebo jako drátový či jednotlivé body.

Moduly

Program sestává ze čtyř modulů a dvou hlavičkových souborů. Jejich vzájemné propojení je vidět na obrázku 4.3.



Obrázek 4.3: Propojení modulů

Modul *main.c* je hlavním programem. Pomocí GLUTu vytvoří okno aplikace a zaregistruje funkce na obsluhu událostí jako jsou požadavek na překreslení, změna velikosti okna, stisknutí klávesy, tlačítek myši a pohybu myši při stisknutém tlačítku. Dále provede inicializaci osvětlení, zapnutí hloubkového bufferu, nastavení plochého stínování. Zavolání funkce z modulu *textures.c*, která vytvoří textury a funkce z modulu *model.c*, která vytvoří display-listy pro materiály a jednotlivé budovy. Budovy jsou potom vykreslovány s vhodně nastavenými transformačními maticemi. Je v něm definována funkce pro vykreslení znaků.

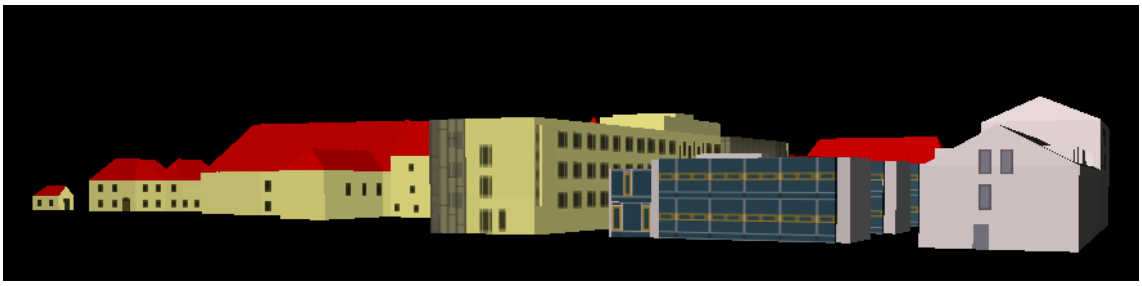
Modul *textures.c* obsahuje funkce pro vytvoření všech textur. Potom také funkci pro Bresenhamův algoritmus vykreslení přímky a kružnice upravený k vykreslení kruhu, funkci pro vyplněný obdélník a lineární interpolaci.

Modul *functions.c* slouží pro výpočet souřadnic, normál a texturovacích souřadnic obdélníku, kvádru, střechy, osmibokého hranolu a osmistěnného jehlanu. Dále obsahuje funkce pro vykreslení zmiňovaných těles, případně pro změnu souřadnic posunutím či rotací[2].

Poslední modul *model.c* definuje display-listy pro materiály a jednotlivé budovy areálu. Využívá funkcí implementovaných v modulu *functions.c*.

Hlavičkový soubor *head.h* deklaruje proměnné pro identifikátory textur a display-listů budov a materiálů. Dále pak funkce pro inicializaci textur a display-listů.

V hlavičkovém souboru *functions.h* jsou uloženy deklarace funkcí obsažených v modulu *functions.c*.



Obrázek 4.4: Současný stav modelu

Obrázek 4.5: Současný stav modelu

[2]Při rotaci se též rotují normály.

Kapitola 5

Návrh a implementace

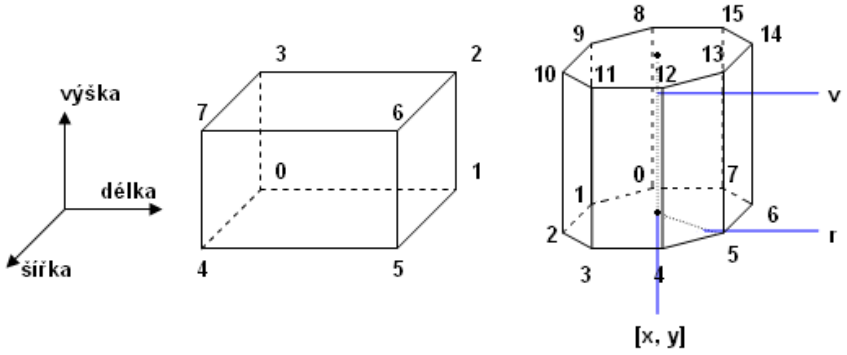
Tato kapitola popisuje konečnou implementaci jednotlivých části programu.

5.1 Geometrie

Část geometrie se zabývá modulem pro jednoduchou tvorbu stěn, střech a oken modelu. Využívá se polí vrcholů vysvětlených v 3.4.4.

5.1.1 Stěny

Stěny budov jsou reprezentovány kvádrem, stěny věží osmibokým hranolem. Funkce, které zajistí naplnění souřadnic a normál, jsou *calculateCuboid()* pro kvádr a *calculateOctagon()* pro hranol. Označení vrcholů je znázorněno na obrázku 5.1(a) a 5.1(b).

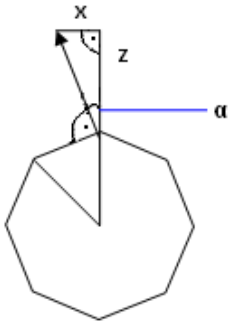


(a) Kvádr (b) Osmiboký hranol

Obrázek 5.1: Označení vrcholů

CalculateCuboid() se volá s argumenty souřadnic počátku, rozměry, polem souřadnic a polem normál. Výpočet souřadnic probíhá ze zadaných rozměrů, pro výpočet normál je definováno pole, jehož hodnoty se zkopírují.

CalculateOctagon() má argumenty stejné. Výpočet normál je vidět na obrázku 5.2. $|\alpha| = 22{,}5°$, $x = sin(\alpha)$, $z = cos(\alpha)$. Díky souměrnosti se ostatní normály spočítají pouze prohozením či invertováním těchto hodnot.



Obrázek 5.2: Výpočet normály

O vykreslení kvádru se postará funkce *drawCuboid()*, která má jeden parametr a to pole normál, či funkce *drawCuboidTex()*, která navíc obsahuje parametry s polem texturovacích souřadnic a počtem stěn, které mají být otexturované. Část kódu z těchto funkcí je uvedena níže a vykreslí pravou stěnu. Vykreslení zbylých stěn je analogické.

```
// nastav normálu pro pravou stěnu
glNormal3fv(normal[1]);
// vykresli pravou stěnu, v right jsou uvedeny indexy 1, 2, 6, 5
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, right);
```

Pro vykreslení osmistěnu je definována obdobná funkce *drawOctagon()*, u níž se taktéž vykreslí jednotlivé stěny pláště jako rovinné čtyřúhelníky, podstavy pak jako mnohoúhelníky, u kterých je normála zadaná nikoliv polem, ale přímo hodnotami.

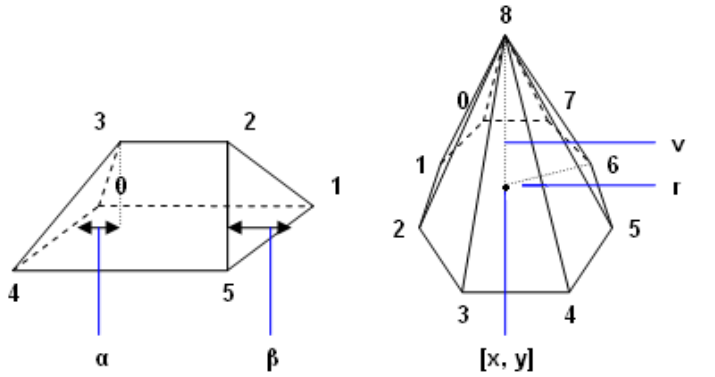
5.1.2 Střechy

Střechu budovy je možné si představit jako ležící trojboký hranol na obrázku 5.3(a) nebo u věží osmistěnný jehlan (obrázek 5.3(b)). Funkce *calculatePrism()* pro budovu a *calculatePyramid()* pro věž mají podobnou formu jako funkce pro stěny. Navíc obsahují parametr pro pole, do kterého uloží texturovací souřadnice. Deklarace funkce *calculatePrism()* je

```
void calculatePrism(GLfloat pocX, GLfloat pocY, GLfloat pocZ,
                    GLfloat length, GLfloat height, GLfloat width,
                    GLfloat alfa, GLfloat beta,
                    GLfloat *vertex, GLfloat normal[][3], GLfloat *texCoord);
```

- *pocX, pocY, pocZ* – souřadnice bodu 0
- *length, height, width* – délka, výška, šířka
- *alfa, beta* – posun vrcholů 2, 3
- *vertex* – ukazatel na pole souřadnic
- *normal* – ukazatel na dvojrozměrné pole normál
- *texCoord* – ukazatel na pole texturovacích souřadnic

Pole texturovacích souřadnic neobsahuje dvanáct prvků, jak by se dalo předpokládat, ale čtrnáct. Funkce uloží texturovací souřadnice pro přední a zadní stěnu a do zbývajících pak výšku a šířku střechy, kterých se využije, pokud funkcí *drawPrismTex()* budou otexturovány i boční stěny. V tom případě se uvnitř této funkce vytvoří lokální pole texturovacích souřadnic, které se inicializuje jako původní pole. Dojde k přepsání hodnot podle rozměrů na indexech 2, 4, 5, 6, 7, 8, 10. Před návratem z funkce se ukazatel na pole texturovacích souřadnic obnoví na původní pole, které se díky této technice nezmění.



(a) Střecha (b) Osmistěnný jehlan

Obrázek 5.3: Označení vrcholů

Pro vykreslení se používají funkce *drawPrism()*, *drawPrismTex()* a *drawPyramid()*. Funkce *drawPrism()* vykreslí přední a zadní stěnu otexturovanou. U funkce *drawPrismTex()* se navíc předá odkaz na texturovací souřadnice a počet stěn[1], které se mají otexturovat.*DrawPyramid()* se postará o vykreslení otexturované střechy věže.

5.1.3 Okna, dveře

Okno i dveře tvoří obdélník. Jsou implementovány tři funkce[2] pro výpočet souřadnic obdélníku. Výpočet normál ani texturovacích souřadnic neprobíhá. Ty je nutné definovat ručně. Vykreslení okna se pak provádí standardní funkcí OpenGL a to *glDrawElements()*. Pole indexů do pole vrcholů je odlišné podle orientace okna. Vrcholy jsou buď ve směru nebo proti směru chodu hodinových ručiček.

Protože oken je většinou více za sebou v jedné řadě, používá se v cyklu funkce *myTranslate()*, která provede posun podle zadaných souřadnic. Není tak nutné stále znovu počítat všechny souřadnice, ale pouze ty, které se mění. Obdobně pracuje funkce *myRotate()*, která však provede rotaci. Tyto dvě funkce je možné použít pro všechny dříve uvedené funkce pro výpočty stěn i střech.

5.1.4 Faktor minimalizace

Ve srovnání s naivní implementací, kdy by nebylo použito pole vrcholů, byly dosaženy tyto výsledky. Na místo 24 volání funkce *glVertex()* u krychle, 18 u střechy, 48 u osmibokého hranolu, 32 u osmistěnného jehlanu a 4 u obdélníku tvořícího okna, dveře, podchody, bylo zapotřebí pouze

[1] tři nebo čtyři

[2] jedna pro každou z os x, y, z

6 volání funkce *glDrawElements()* u kvádru, 5 u střechy, 10 u hranolu, 9 u jehlanu a 1 u oken v případě, že se nepočítá s texturováním. Pokud jej vezmeme v úvahu, pak počet funkcí posílaných přes sběrnici při naivním řešení ještě vzroste o zadávání texturovacích souřadnic, kdežto ve výsledném řešení nikoliv.

5.2 Textury

Tato část popisuje vytváření textur, které proběhne po spuštění programu. Jsou zde také uvedeny algoritmy pro vykreslení úsečky, vyplněného obdélníku a kruhu do pole tvořícího texturu. Dále následuje ukázka kódu k vytvoření textury a odpovídající obrázek.

5.2.1 Úsečka v textuře

Bresenhamův algoritmus 3.6.1 bylo nutné rozšířit o parametry: odkaz na pole tvořící texturu, barvu, kterou chceme úsečku vykreslit a šířku textury. Výsledná deklarace je potom

```
void drawLine(int x1, int y1, int x2, int y2, GLubyte *textureArray,
              int *color, int textureWidth);
```

Bylo třeba ošetřit následující stavy:

- funkce rychleji roste ve směru osy y, tedy $\Delta y > \Delta x$
  řešení: `SWAP(x1, y1); SWAP(x2, y2); SWAP(dx, dy);`[3]
- $x_1 > x_2$
  řešení: `SWAP(x1, x2); SWAP(y1, y2);`
- $y_1 > y_2$
  řešení: `stepY = -1;`[4]

5.2.2 Obdélník v textuře

Vykreslení obdélníku probíhá následujícím způsobem: Předpokládá se, že $x_1 < x_2$ a $y_1 < y_2$. Začne se od levého vrchního pixelu, pak se pokračuje dokud není $x_1 = x_2$. Dále se přejde na další řádek a inkrementuje se y. Takto se pokračuje, dokud není $y = y_2$.

Funkce obsahuje stejné parametry jako *drawLine()*, nazývá se *drawRectangle()*.

Bylo třeba ošetřit tyto stavy:

- $x_1 > x_2$
  řešení: `SWAP(x1, x2);`
- $y_1 > y_2$
  řešení: `SWAP(y1, y2);`

[3] Makro SWAP(int a, int b) je deklarováno v *head.h* a prohodí hodnoty dvou proměnných.
[4] StepY určuje krok v ose y. Na počátku je inicializována stepY = 1.

5.2.3 Kruh v textuře

Při tvorbě funkce se vycházelo z Bresenhamovu algoritmu pro rasterizaci kružnice 3.6.2. Přibyly další parametry. Deklarace funkce je

```
void DrawFilledCircle(int cx, int cy, int radius, GLubyte *textureArray,
                      int * color, int textureWidth, int textureHeight)
```

- *cx, cy* – souřadnice středu kruhu v textuře
- *radius* – poloměr
- *textureArray* – ukazatel na pole tvořící texturu
- *color* – ukazatel na pole obsahující barvy RGBA v rozmezí od 0 do 255 pro každou složku, které určuje barvu kruhu
- *textureWidth, textureHeight* – rozměry textury

Rozšíření oproti Bresenhamovu algoritmu pro kružnici:

- libovolná poloha středu kružnice
- záměnou souřadnic se získají souměrné body
- v jednotlivých řádcích se cyklem vyplní prostor mezi získanými body, aby vzniknul kruh
- test, zda bod leží uvnitř textury
- volitelná barva kruhu

5.2.4 Tvorba textury

Tvorba spočívá ve výběru velikosti textury a vykreslení požadovaných grafických primitiv se zvolenou barvou do příslušného pole.

Následující kód demonstruje vytvoření textury střechy a nastavení vlastností textury. Vzniklá textura je na obrázku 5.4.

```
// inicializace ukazatele na pole tvořící texturu
GLubyte *p = texture32_64;
// inicializace pole obsahujícího složky RGBA
int black[] = {0, 0, 0, 255};
// naplnění pole červenou barvou
   ...

DrawLine(0, 10, 15, 0, texture32_64, black, 32);
DrawLine(15, 0, 31, 10, texture32_64, black, 32);
// vykreslení dalších úseček
   ...

// vygeneruje, přidělí jméno textury
glGenTextures(1, &texid);
// texid bude aktivní textura
```

```
glBindTexture(GL_TEXTURE_2D, texid);
// způsob uložení
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
// opakovaní textury ve směru s
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
// opakovaní textury ve směru t
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// lineární filtr při zvětšení
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// trilineární filtr při zmenšení
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
// vytvoření mipmapy
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, 32, 64,
        GL_RGBA, GL_UNSIGNED_BYTE, texture32_64);
```

Obrázek 5.4: Textura tašky

5.2.5 Faktor minimalizace

V případě naivního řešení s texturami o rozměrech 32 bodů na šířku a 64 bodů na výšku uloženými jako bitmapové obrázky by se při stavu 37 textur spotřebovalo 222 KB paměti[5].

Ve výsledné implementaci jsou veškeré textury vytvořené procedurou, a proto zabírají podstatně méně místa. Navíc je zvolena vhodná velikost textur pohybující se od 4 do 32 pixelů na šířku a od 1 do 64 pixelů na výšku. Počet potřebných textur je snížen díky použití části již vytvořených textur vhodným nastavením texturovacích souřadnic.

5.3 Model

Vytváření modelu bylo implementováno s využitím dříve uvedených funkcí. Jednotlivé části budovy tvoří bloky, které jsou vždy ve stejném pořadí. Toto pořadí není nutné, slouží pro lepší vyhledávání a případnou editaci konkrétní části budovy.

Postup tvorby budovy:

1. stěny – Při vytváření stěn se ze všeho nejdříve nastaví materiál. Pokud stěny budou otexturovány, je třeba nastavit aktuální texturu. Následuje volání funkcí pro nastavení ukazatelů pole souřadnic (*glVertexPointer()*) a texturovacích souřadnic (*glTexCoordPointer()*) pro kvádr. Nyní už se jen zadají požadované rozměry, počáteční bod a příslušná pole pro funkci *calculateCuboid()*. Vykreslení je provedeno funkcí *drawCuboid()* nebo *drawCuboidTex()*.

[5] Velikost výsledného spustitelného souboru s použitím UPX je 224 KB

2. střechy – Na začátku je nutné změnit ukazatele do polí vrcholů tak, aby ukazovaly na pole pro střechu, dále pak změnit aktuální texturu. Poté stačí zavolat funkce *calculatePrism()* a *drawPrism()* či *drawPrismTex()*.

3. věže – Věže tvoří samostatný celek. Nejdříve jsou vykresleny stěny, poté střecha. Začne se opět změnou ukazatelů na pole. Následuje *calculateOctagon(), drawOctagon()*. Postup pro střechu je analogický (*calculatePyramid(), drawPyramid()*).

4. okna – 0 vykreslení oken se stará standardní funkce *glDrawElements()*. S tím souvisí to, že je nutné zapínat a vypínat texturování, nastavovat normály a texturovací souřadnice. Postupuje se tímto způsobem:

   - změna ukazatele na pole vrcholů (ruční nastavení texturovacích souřadnic[6])
   - nastavení aktuální textury
   - povolení texturování
   - nastavení normály
   - výpočet souřadnic[7]
   - vykreslení
   - vypnutí texturování

   Stejná okna většinou tvoří řadu. V tom případě se souřadnice spočítají jednou a v cyklu jsou měněny funkcí *myTranslate()*.

5. dveře – Postup je stený jako u oken.

6. podchody – Oproti oknům rozdíl spočívá v použití blendingu, který umožní dosažení oblého tvaru. Před vykreslením je zakázán zápis do hloubkového bufferu, povolen blending a nastavena míchací funkce. Po vykreslení je zápis opět povolen a blending zakázán. Řada podchodů vedle sebe je tvořena jedním obdélníkem se zopakovanou texturou ve směru s a protaženou ve směru t.

7. výklenky – Nejprve nastavíme pole souřadnic pro kvádr a pole texturovacích souřadnic[8]. Provede se výpočet funkcí *calculateCuboid()*. Pro vykreslení je buď použita funkce *drawCuboidTex()* nebo standardní funkce OpenGL *glDrawElements()*. Ve druhém případě je podle potřeby nastavena aktivní textura a zapnutí či vypnutí texturování a blendingu, případně změna texturovacích souřadnic.

Demonstrace vytvoření display-listu jednoduché budovy je uvedena v příloze C. Předpokládá se, že je definován display-list s materiálem brightMat, textura s identifikátorem texid a texid8 a všechna pole uvedená jako argument.

[6] V programu je dostupná konstantní proměnná typu pole, která obsahuje texturovací souřadnice $\{0, 0, 1, 0, 1, 1, 0, 1\}$, nebo je možné využít funkci *changeTexCoord()*, které se předají argumenty texturovací souřadnice a pole, do kterého se souřadnice přiřadí.

[7] fukncí *calculateWindowX(), calculateWindowY() nebo calculateWindowZ()*

[8] buď pro kvádr, nebo pro okno

5.4 Zobrazení

Tento oddíl blíže vysvětluje modul pro zobrazení modelu.

5.4.1 Použití GLUTu

Jak bylo uvedeno v části 3.5, je vytvoření okna a registrace funkcí reagujících na události relativně jednoduché. Provede se inicializace knihovny GLUT, nastaví se grafický mód, velikost a pozice okna. Následuje vytvoření okna a registrace funkcí pro obsluhu. Po těchto úkonech je možné zavolání našich uživatelských funkcích. Toho je také v programu využito. Nejprve se inicializuje osvětlení, poté textury a nakonec se vytvoří display-listy obsahující modely budov. Potom je spuštěna nekonečná smyčka reagující na události.

```
// inicializace knihovny GLUT
glutInit(&argc, argv);
// grafický mód okna
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
// počáteční velikost okna
glutInitWindowSize(600,600);
// počáteční pozice okna
glutInitWindowPosition(50,10);
// vytvoření okna
glutCreateWindow(‘‘Minimalistická reprezentace modelu areálu... ’’);

// registrace funkce volané při stisku klávesy
glutKeyboardFunc(onKeyboard);
// registrace dalších funkcí
...

// provést naši inicializaci textur
InitTexture();
...

// nekonečná smyčka
glutMainLoop();
```

5.4.2 Zobrazení modelu

Aby bylo možné s modelem manipulovat, byly vytvořeny proměnné jež obsahují informaci o kolik se má scéna posunout od kamery či otočit. Podle těchto proměnných se provede translace a rotace[9] modelview matice.

Hodnoty pro rotace se získají při stisku levého tlačítka myši a následným pohybem. Hodnoty pro translace jsou vypočteny na základě stisku šipek nebo pohybu myši se stisknutým pravým tlačítkem.

Poté je scéna nastavená pro vykreslování budov. Každá budova má svůj lokální souřadný sytém, takže je před jejím vykreslením nutné opět zvolit vhodnou transformaci. Aby se nepřepsala původní transformační matice, je možné ji vložit na zásobník a po vykreslení budovy ji buď obnovit, nebo počítat s tím, že počátek souřadného systému začíná v předchozí budově.

[9] v tomto pořadí

5.4.3 Výpis nápovědy

Pro výpis nápovědy byla využita funkce *glutBitmapCharacter()*, která vykreslí jeden znak v nastaveném formátu. Je obsažena v knihovně GLUT. Této funkci se předá v cyklu celý text, jenž chceme vykreslit[10].

5.5 Velikost binárního souboru

Použitím různých překladačů s různými přepínači lze dosáhnout rozdílných velikostí výsledného binárního souboru. Pro následnou minimalizaci velikosti souboru je možné použít kompresního programu UPX. Údaje získané experimentováním s překladačem MinGW jsou uvedeny v tabulce 5.1.

Překladač	UPX	Velikost souboru (KB)
g++		907
g++	-1	305
g++	-9	224
g++	- -brute	168
g++ -Os		881

Tabulka 5.1: Velikost binárního souboru

[10] Předtím je třeba nastavit pozici prvního znaku.

Kapitola 6

Závěr

Cílem této práce bylo seznámit se s tvorbou grafických aplikací s omezenou velikostí a vhodnými technikami komprese modelů a na základě získaných znalostí pak vytvořit model areálu Božetěchova v úrovni detailu vhodné pro minimalistické uložení a program, který tento model bude zobrazovat. Tyto požadavky byly splněny.

1. Problematikou tvorby grafických aplikací s omezenou velikostí se zabývá třetí kapitola.
2. Techniky komprese modelů zmiňuje rovněž třetí kapitola.
3. Vytvoření modelu popisuje čtvrtá a pátá kapitola.
4. Implementace programu zobrazujícího model je uvedena ve čtvrté a páté kapitole.
5. Shrnutím dosažených výsledků a možnostmi pokračování se zabývá zbytek této kapitoly.

Při řešení této práce jsem se především seznámil s knihovnou OpenGL, což byl jeden z hlavních důvodů mého výběru této práce. Naučil jsem se dekomponovat složitější problémy na menší části a vhodně využít modulárního programování.

Samozřejmě program není dokonalý a nabízí další možnosti k vylepšení. Zhodnocení ve srovnání s naivní implementací je uvedeno v podkapitole 5.1.4, 5.2.5. Za nedostatek považuji tvorbu jednotlivých oken. Lepší by bylo podobně jako podchody vykreslit obdélník a v něm zopakovat texturu okna s částí zdi. Pro menší velikost výsledného spustitelného souboru by také bylo vhodné souřadnice načítat z textového souboru, aby se ušetřil kód, který provádí jejich výpočet.

Další možností rozšíření je vypracovat způsob, jenž by získal potřebné údaje z CAD modelu, aby nebylo nutné údaje zadávat ručně. Dále by bylo možné domodelovat celý areál, až budou provedeny všechny rekonstrukce a dostavby, které se v současné době konají, vytvořit animaci, jež by představovala průlet areálem. V další řadě by mohlo následovat vytvoření povrchu, tj. silnice, chodníku, trávy a také interiéru budov. Zajímavé by rovněž bylo využití techniky progressive meshes.

Seznam použitých zdrojů

[1] HOPPE, H.: Progressive Meshes. 2007, [Online; navštíveno 8. 5. 2007].
URL <http://research.microsoft.com/~hoppe/pm.pdf>

[2] KRŠEK, P.; ŠPANĚL, M.: Rasterizace objektů ve 2D. 2007, [Online; navštíveno 8. 5. 2007].
URL <https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_rasterizace_print.pdf>

[3] KRČMÁŘ, P.: Unixová komprese v praxi: UPX. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/unixova-komprese-v-praxi-upx/>

[4] POKORNÝ, L.: Co je to demoscéna. 2000, [Online; navštíveno 8. 5. 2007].
URL <http://www.scene.cz/showarticle.php?num=01>

[5] ŽÁRA, J.; aj.: *Moderní počítačová grafika*. Brno: Computer Press, první vydání, 2004, ISBN 80-251-0454-0.

[6] RHOADES, J.; et al.: Real-Time Procedural Textures. 1992, [Online; navštíveno 8. 5. 2007].
URL <http://www.cs.umd.edu/projects/gvil/papers/i3d92.pdf>

[7] SHREINER, D.; et al.: *OpenGL Průvodce programátora*. Brno: Computer Press, první vydání, 2006, ISBN 80-251-1275-6.

[8] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (1). 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/graficka-knihovna-opengl-1/>

[9] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (16): pole vrcholů (vertex arrays). 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-16-pole-vrcholu-vertex-arrays/>

[10] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (18): vykreslování osvětlených těles. 2003, [Online; navštíveno 8. 5. 2007].
URL
<http://www.root.cz/clanky/opengl-18-vykreslovani-osvetlenych-teles/>

[11] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (2): syntaxe funkcí. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-2-syntaxe-funkci/>

[12] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (22): texturování. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-22-texturovani/>

[13] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (23): korektní otexturování. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-23-korektni-otexturovani/>

[14] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (25): mipmapping. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-25-mipmapping/>

[15] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (6): využití transformačních matic. 2003, [Online; navštíveno 8. 5. 2007].
URL
<http://www.root.cz/clanky/opengl-6-vyuziti-transformacnich-matic/>

[16] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (8): display-listy. 2003, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-8-display-listy/>

[17] TIŠNOVSKÝ, P.: Grafická knihovna OpenGL (28): blending. 2004, [Online; navštíveno 8. 5. 2007].
URL <http://www.root.cz/clanky/opengl-28-blending/>

[18] Wikipedie: Počítačová grafika — Wikipedie: Otevřená encyklopedie. 2007, [Online; navštíveno 8. 5. 2007].
URL
<http://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%A1_grafika>

Seznam příloh

Příloha A

Obsah přiloženého CD

Součástí této práce se přiložené CD, které obsahuje zdrojové soubory aplikace, úplnou programovou dokumentaci, textovou část bakalářské práce a zdrojové kódy potřebné k vytvoření požadovaných dokumentů. Adresářová struktura CD:

/src – adresář obsahující zdrojové soubory aplikace

/doc – adresář obsahující textovou část bakalářské práce a úplnou programovou dokumentaci

/tex – adresář obsahující zdrojové kódy k vytvoření textové části bakalářské práce

Příloha B

Ovládání programu

Klávesa	Význam
H	Zobrazení/skrytí nápovědy
F	Přepnutí zobrazení na celou obrazovku
W	Přepnutí zobrazení do okna
1	Zobrazení vyplněných polygonů
2	Zobrazení drátového modelu
3	Zobrazení bodů
$\leftarrow\rightarrow\uparrow\downarrow$	Posun modelu
ESC, Q, X	Ukončení programu

Tabulka B.1: Ovládání programu pomocí klávesnice

Akce myší	Význam
L + posun	Rotace modelu
P + posun	Zoom

Tabulka B.2: Ovládání programu pomocí myši

Příloha C

Ukázka kódu pro vytvoření budovy

Kód uvedený níže způsobí vykreslení obrázku C.1.

```
// vytvoření indexu display-listu
budova = glGenLists(1);
// začátek kompilovaného display-listu
glNewList(budova, GL_COMPILE);
  // nastaveni dříve definovaného materiálu
  glCallList(mat);
  // nastavení ukazatele pole souřadnic
  glVertexPointer(3, GL_FLOAT, 0, cuboidVertex);
  // nastavení ukazatele pole texturovacích souřadnic,
  // pokud by se neprovedlo došlo by k chybě, proto
  // dopředu nastavíme, i když se zatím nevyužije
  glTexCoordPointer(2, GL_FLOAT, 0, prismTexCoord);
  // výpočet stěn, délka = 10, výška = 5, šířka = 5
  calculateCuboid(0, 0, 0, 10, 5, 5, cuboidVertex, cuboidNormal);
  // vykreslení stěn
  drawCuboid(cuboidNormal);

  // nastavení ukazatele pole souřadnic
  glVertexPointer(3, GL_FLOAT, 0, prismVertex);
  // aktivní textura bude textura střechy
  glBindTexture(GL_TEXTURE_2D,texid);
  // výpočet střechy, délka = 10, výška = 3, šířka = 5,
  // alfa = 0, beta = 2
  calculatePrism(0, 5, 0, 10, 3, 5, 0, 2, prismVertex,
                 prismNormal,prismTexCoord);
  // vykreslí se střecha, 3 stěny otexturované
  drawPrismTex(prismNormal, prismTexture, 3);

  // nastavení ukazatele pole souřadnic
  glVertexPointer(3, GL_FLOAT, 0, octagonVertex);
  // výpočet stěn, poloměr = 1.5, výška = 5
  calculateOctagon(-3, 0, 1.5, 5, octagonVertex, octagonNormal);
  // vykreslení stěn věže
```

```
  drawOctagon(octagonNormal);

  // nastavení ukazatele pole souřadnic
  glVertexPointer(3, GL_FLOAT, 0, pyramid);
  // nastavení ukazatele pole texturovacích souřadnic
  glTexCoordPointer(2, GL_FLOAT, 0, pyramidTexCoord);
  // výpočet střechy věže, poloměr = 1.7, výška = 2
  calculatePyramid(-3, 5, 0, 1.7, 2, pyramid,
                   pyramidNormal, pyramidTexCoord);
  // vykreslení střechy věže
  drawPyramid(pyramidNormal);

  // nastavení ukazatele pole souřadnic
  glVertexPointer(3, GL_FLOAT, 0, windowVertex);
  // nastavení ukazatele pole texturovacích souřadnic
  glTexCoordPointer(2, GL_FLOAT, 0, windowTexCoord);
  // aktivní textura bude textura okna
  glBindTexture(GL_TEXTURE_2D,texid8);
  // zapnutí texturování
  glEnable(GL_TEXTURE_2D);
  // nastavení normály
  glNormal3f(0, 0, 1);
  // výpočet okna, šířka = 1.5, výška = 2
  calculateWindowX(2, 2, 5.5, 1.5, 2, windowVertex);
  for(int i = 0; i < 2; i++)
  {
    // vykreslení okna
    glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, windowBack);
    // posun souřadnic o 4 ve směru osy x
    myTranslate(4, 0, 0, windowVertex, 4);
  }
  // vypnutí texturování
  glDisable(GL_TEXTURE_2D);
// konec display listu
glEndList();
```

Obrázek C.1: Ukázka modelování