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

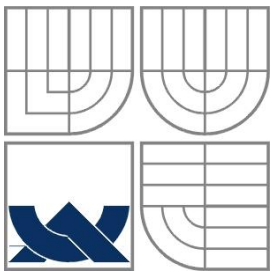
# MINIMALISTICKÁ REPREZENTACE MODELU AREÁLU BOŽETĚCHOVA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

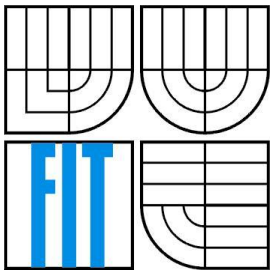
AUTOR PRÁCE
AUTHOR

RASTISLAV PIOVARČI

BRNO 2010



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

MINIMAL REPRESENTATION OF THE BOŽETĚCHOVA COMPLEX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RASTISLAV PIOVARČI

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM VLČEK

BRNO 2010

# Tu vložiť zadanie

(1 STRANA A4)

# Abstrakt

Tato práce si klade za cíl stručně seznámit čtenáře s fenoménem demoscény jako východiskové základny pro tvorbu minimalistické aplikace, kterou implementuje. Následně vysvětluje proces návrhu a implementace minimalistické aplikace zobrazující areál Božetěchova. Cílem této interaktivní aplikace je zachytit areál Božetěchova v co nejlepší kvalitě za omezené velikosti binárního souboru. Využívá k tomu prostředky, jako jsou procedurální textury na bázi Perlinova šumu či mnohonásobné komprimace a instancování modelových součástí.

# Abstract

The goal of this thesis is to briefly familiarize the reader with damoscene phenomena as a foundation for creation of minimalistic application, which it implements. Then it explains the process of designing and implementation of minimalistic application which displays the Božetechova complex. The aim of this interactive application is to reproduce the Božetechova complex in the best possible quality using a restricted size binary file. It uses means like procedural textures on the bases of Perlin noise, or multilevel comprimation and instantiating of model geometry to achieve this goal.



# Klíčová slova

Minimalistická reprezentace, demoscéna, OpenGL, C/C++, Python, 3DSMAX, VRML, komprese scény, SFX, procedurální textury, perlinova šumová funkce, instancování geometrie.

# Keywords

Minimal representation, demoscene, OpenGL, C/C++, Python, 3DSMAX, VRML, scene compression, SFX, procedural textures, perlin noise function, geometry instancing.


# Citace

Rastislav Piovarči:Minimalistická reprezentace modelu areálu Božetěchova, bakalářská práce, Brno, FIT VUT v Brně, 2010

# Minimalistická reprezentace modelu areálu Božetěchova

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. A.Vlčeka
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Rastislav Piovarči
Datum (19.05.2010)

## Poďakovanie

Rád by som sa týmto poďakoval Ing. Adamovi Vlčekovi za jeho neutíchajúcu trpezlivosť so mnou, priateľský prístup a ochotu stretnúť sa a poradiť hoci kedy to bolo nutné. Taktiež by som sa rád poďakoval všetkým kto ma počas tvorby podporovali a dodávali mi silu do ďalšej práce.

# Obsah

# 1 Úvod

V dnešnej dobe, kedy máme miesta na pevných diskoch osobných počítačov neúrekom je možno otázka minimalizácie zatláčaná do úzadia. Táto práca ako aj interaktívna aplikácia ktorej princípy a implementáciu popisuje sa snažia prezentovať iný spôsob pohľadu na klasické problémy s ktorými sa potýka 3D grafika v dnešných dňoch.

Jedným z hlavných problémov je enormné množstvo dát ktoré zaberajú textúry, či model na pevnom disku osobného počítača a ktoré je užívateľ nútený skladovať temer hoc ktorou klasickou 3D aplikáciou na svojom počítači. Načrtáva prístup pomocou ktorého je možné generovať veľmi kvalitné a obrazovo zaujímavé textúry ktoré na pevnom disku užívateľa zaberajú nepatrný priestor oproti ich bitmapovým ekvivalentom.

Opisuje generovanie textúr za pomoci perlinovho šumu. Taktiež sa zaoberá metóda my na minimalizáciu rozmerov dát reprezentujúcich 3D objekty zobrazované programom. Či už je to za pomoci redukcie samotných modelových dát, alebo za pomocou programátorskej cesty, použitím pre náš účel vhodnejších dátových typov. Následne predstavuje najčastejšie používané metódy zobrazovania, ich výhody, či nevýhody a ich implementáciu v nami zostrojenej aplikácií.

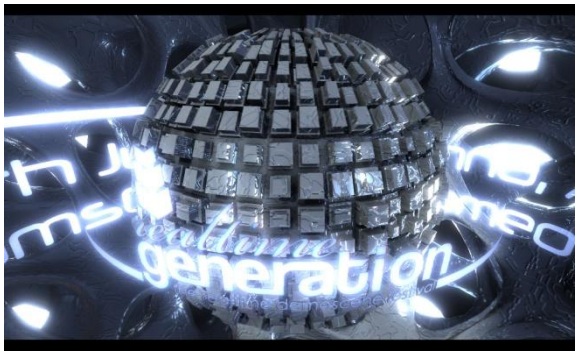
Nakoniec sa venuje spôsobu implementácie vybraných častí programu. Nástrojom použitým počas vývoja aplikácie a zhodnoteniu dosiahnutých výsledkov, ktoré sú reprezentované porovnaniami skutočných fotografií s obrázkami generovanými aplikáciou.

# 2 Minimalistické aplikácie a Demoscéna

Príkladom minimalistického programu je počítačová hra s menom Roboblitz. V tejto hre sú všetky textúry generované procedurálne, animácie postáv a interakcie s okolím sú tiež popísané funkčnými zákonitosťami známymi z reálneho sveta, namiesto pevnej animácie, používajúcej kľúčové snímky[1] a opakovanie určitých cyklov, či interpoláciu medzi bodmi.

*Obrázok 1: Ukážka z hry Roboblitz*

Ďalšou a od tejto témy neoddeliteľnou súčasťou je demoscéna. Slovo demo pochádza zo slova demonštrovať. Jedná sa teda o určitý druh umenia a zároveň prezentácie vlastných schopností, znalostí a kreativity. Pri tvorení takýchto diel ide o to aby autor, či zväčša tým autorov vytvoril pokiaľ možno čo najzaujímavejší obsah – z hľadiska dejového, grafického a hudobného.




*Obrázok 2: Ukážka z dema Realtime Genretation(19MB)*

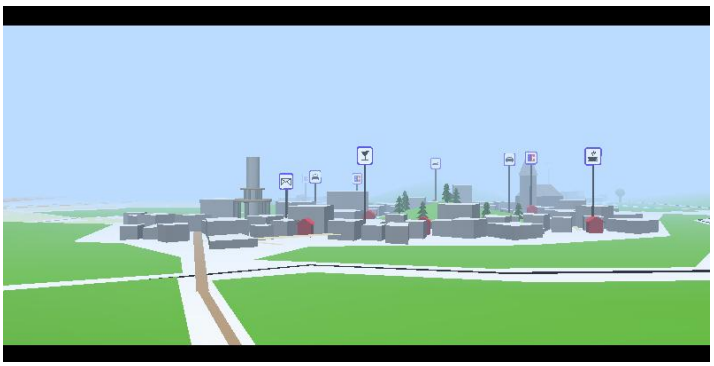
Vznik demoscény sa datuje na začiatok 80-tych rokov. Prvé demá sa rodili v časoch, keď neexistovali grafické akcelerátory a Comodore 64 bola novinka. Začali ako malé prídavky – podpisy softwarových pirátov k odomknutým (crackovaným) programom, či hrám. Nakoľko samotné dátové

médiá mali veľmi obmedzenú kapacitu, bolo nutnosťou, aby aj samotné intrá mali nepatrnú veľkosť. Ak chcel autor vytvoriť podpis a trvácny dojem, muselo intro vyzerať, pokiaľ možno čo najlepšie [2].

Demá sú často malými – veľkými umeleckými dielami, prezentovanými na digitálnej platforme, najčastejšie je to osobný počítač, môžu to však byť aj iné platformy ako napríklad C64, MSX, ZX Spectrum, Amstrad CPC, Amiga, ATARI, Dreamscast, či Game Boy Advance[3].

Ich rozmanitosť má rôzne podoby, od krátkych filmov vyjadrujúcich nejakú myšlienku/dej, cez abstraktné, zobrazujúce dych berúce scenérie, za pomoci fraktálov generované útvary, až po hudobné, v ktorých je obrazová stránka len akýmsi doplnkom, rozvíjajúcim fantáziu pozorovateľa na základe presnej synchronizácie hudby a obrazu.

Špecifickou časťou demoscény sú súťaže a do nich vytvárané intrá. Jediný rozdiel medzi demom a introm je ten, že intro je obmedzené veľkosťou. Dôvod tohto obmedzenia je jasný, je to práve súťaž. Intrá sú vytvárané zväčša za účelom súťaže vo svojej kategórií. Najčastejšie sa vyskytujú dve súťažné kategórie, ktorými sú 64 KB - teda 65536 B alebo 4 KB demo - 4096 B. Tieto súťaže prebiehajú zväčša ako súčasť demo party, čo je stretnutie rôznych skupín vytvárajúcich demá. Na stretnutiach tohto typu sa okrem voľby najlepšieho dema konajú aj iné aktivity, akými sú výmena informácií, či rôzne iné súťaže podobného typu.



*Obrázok 3: Ukážka z intra Ausflug (64KB)*

I keď to nie je v zadaní práce explicitne uvedené, našou snahou je vytvorenie niečoho, čo má niekoľko hlavných charakteristík intra a to sú : vytvoriť pokiaľ možno vizuálne čo najpôsobivejšiu reprezentáciu areálu Božetechova a neprekročiť pritom veľkosť spustiteľného súboru 64KB.

# 3 Analýza problémov

Problém vytvorenia systému takejto komplexnosti nie je možné riešiť bez rozkladu na podproblémy. Po naštudovaní základných informácií vyvstala otázka platformy na ktorej program pobeží. Riešenia boli dve a to OpenGL a DirectX. Po preskúmaní materiálov dostupných pre učenie od úplných začiatkov som sa priklonil k variante OpenGL, ktorá nám bola predstavená aj v rámci výučby.

Pre začiatok som skúsil pracovať s cvičeniami ktoré nám boli poskytnuté v rámci výučby, ale nakoniec som z dôvodu priamejšieho zamerania na moju tematiku použil ako základ návod z portálu[4] - NeHe Tutoriály ktoré sa zaoberajú tvorbou návodov pre OpenGL. Ako kostru programu som použil návod [5]. Tento návod implementuje najzákladnejšie zobrazenie scény a pohyby v nej. Kostra tohto programu a jej časti ako tvorba okien do WinApi, či zmeny ich veľkostí sú prevzaté, ostatné súčasti však bolo treba odznova prepísať pre náš účel. Návod mi pomohol pochopiť a implementovať základnú verziu zobrazovacieho modulu, ktorý som následne rozvíjal a ktorému sa podrobnejšie budem venovať v kapitole 4.

Nasledoval problém voľby WISIWG editora na vymodelovanie komplexu fakulty. Na tento účel by bolo možné použiť známe editory ako SoftimageXSI, Rhinoceros, Blender a mnoho iných. Voľba však padla na mnou známy a pomerne dobre ovládaný software 3DSMAX, ktorý poskytuje rozsiahle zázemie, pre modelovanie, optimalizáciu až po obrovský rozsah priamo exportovateľných formátov.

Zvolenie správneho formátu pre export dát bolo veľmi dôležité, nakoľko výber podstatných dát z binárneho formátu je samo o sebe je netriviálnou záležitosťou. Zvolil som z pomedzi širokého výberu formátov, ktoré 3D štúdio ponúka formát VRML – (virtual reality modelling language). Bolo to z dôvodu, podpory pre export dát v tomto formáte, temer všetkými známymi editormi tohto typu od CAD –systémov cez polygonálne WISIWIG editory, ako napríklad Blender, XSI, až po nami použitý 3DS MAX (buď priamo alebo za pomoci do inštalovateľného prídavného modulu). VRML je textový formát, ktorý bol uznaný za štandard Medzinárodnou Organizáciou pre Štandardizáciu [6] v roku 1997. Výhody tohto formátu – ako prenositeľnosť a textová forma prevážili nad ostatnými.

Jednou zo špecifikácií, ktorá vyplýva zo zadania, je že výsledkom môjho snaženia má byť len jeden binárny súbor – je tento prechod do textovej podoby len polovičkou cesty, ktorú musia dáta prejsť, aby sa stali súčasťou programu.

Logickou cestou je použitie ďalšieho nástroja na spracovanie textu, ktorý by vytvoril spracovateľnú položku programu a teda prikompilovateľný dátový model. Na túto úlohu sú mimoriadne vhodné skriptovacie jazyky ako Python, Ruby či Perl. Rozhodovanie bolo opäť ovplyvnené mojimi predošlými skúsenosťami. Na vytvorenie dátového modelu som zvolil jazyk Python. Problémom, s ktorými som sa potýkal pri tvorbe tohto skriptu, sa venujem v kapitole 6.1.

Toto by riešilo stránku základného zobrazenia modelu, nie však jeho vzhľad a veľkosť. Tu prichádzame k riešeniu, ktoré je nám naporúdzi a ktoré vyplýva z koncepcie riešenia a tým je pridanie zdrojových modulov, ktoré obsahujú funkcie implementujúce inštancovanie a textúrovanie. Na veľkosť daného binárneho súboru je braný zreteľ počas celého vývoja a vo všetkých fázach, ale konkrétnym vybraným spôsobom sa budem venovať v kapitole 3.

# 4 Redukcia veľkosti

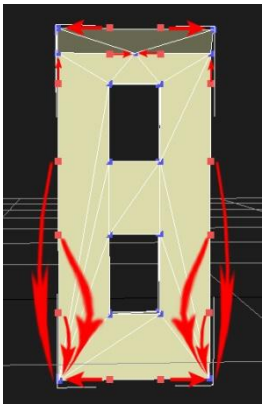
## 4.1 Optimalizácia modelu

Model je miesto kde všetky dáta ktoré budú následne spracúvane, upravované a komprimované vznikli a teda je to aj miesto kde sa dá ušetriť veľké množstvo dát. Počas modelovania som použil niekoľko najznámejších modelovacích techník. Takzvané box modelovanie a low poly modelovanie. Tieto techniky sú opísané v knihe Praktické postupy pre 3DS MAX[7]

Optimalizácia modelu spočíva najmä v odstránení neviditeľných súčastí, zjednodušení reality na úroveň ktorá dostačuje zadaniu našej úlohy. Celý projekt je hlavne o minimalizácií, bude teda našou prioritou snaha o dodržanie pokiaľ možno čo najvyššej kvality a detailnosti modelu za použitia čo najmenšieho počtu vrcholov a polygónov.

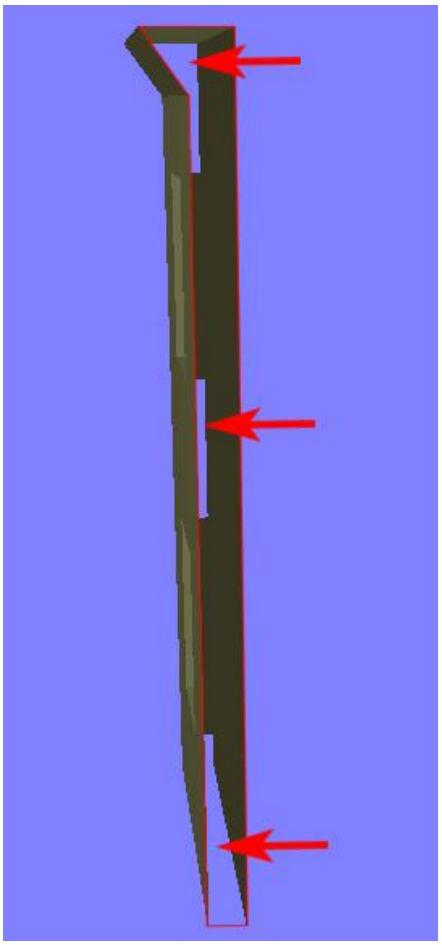
V niektorých prípadoch je možné bez straty vizuálnej príťažlivosti modelu odstrániť veľké množstvo vrcholov. Je to napríklad v prípade keď viacero vrcholov jedného objektu leží v rovine a sú navzájom prepojené trojuholníkmi.

Ako môžeme vidieť na obrázku 4, zo začiatočného počtu vrcholov prednej steny - 28 nám zostalo 15 vrcholov čo je skoro polovica. Berúc v úvahu fakt že táto časť sa v komplexe vyskytne desiatky možno až cez sto krát, je táto úspora značná. Ak by sme predpokladali 100 výskytov pri maximálnej komprimácií by sa jednalo o 100*13 teda 1300 vrcholov každý s troma súradnicami.



*Obrázok 4:Ukážka redukcie počtu vrcholov*
*na jednoduchom modely kusu steny*

Iným prípadom je odstránenie celých polygónov, ktoré v zásade nemôžu byť zobrazené – napríklad vnútorná strana steny, či vnútorné lemovanie okien.



*Obrázok 5:Červenou linkou sú obtiahnuté hranice polygónu ktorý bol zmazaný, šípky ukazujú na miesta ktoré sú síce z tohto profilového pohľadu priehľadné, ale vo výslednom zobrazení nebude vidieť*

Ďalšej, i keď minimálnej optimalizácie veľkosti exportovaných dát môžeme dosiahnuť spájaním objektov do jedného vačšieho objektu. Má to tú výhodu, že odpadne export mnohých dát pre každý objekt, akými sú napríklad otočenie, posun, farba a iné. Nevýhodou tejto metódy je však strata presnosti. Ak pridáme do jedného objektu body z viacerých objektov, pri optimalizácii bodov nám ostane presnosť 256 hodnôt (dôvod viz kapitola 3.3.2), týmto môže dôjsť ku skresleniu polohy niektorých bodov, poretže presnosť originálneho modelu nebude možné bez straty pri kompresii zobraziť. Je teda potrebné používať túto metódu s uvážením.

Všetky tieto úpravy prispevajú nielen k zníženiu veľkosti exportovaných dát, ale aj ku zmenšeniu výpočetnej náročnosti a teda k zrýchleniu vykreslovania scény.

## 4.2 Inštancovanie

Metóda inštancovania objektov je veľmi dobre známa a často používaná snáď v akomkoľvek počítačovom programe – v objektovom programovaní sa používa ako vytváranie objektov- inštancií danej triedy. Ide teda o snahu ušetriť čo najviac miesta zabraného v programe práve recykláciou a znovu použitím toho, čo už máme. V našom prípade sa jedná o inštancovanie už použitých dát v zdrojovom súbore, čím šetríme predovšetkým miesto v binárnom súbore.

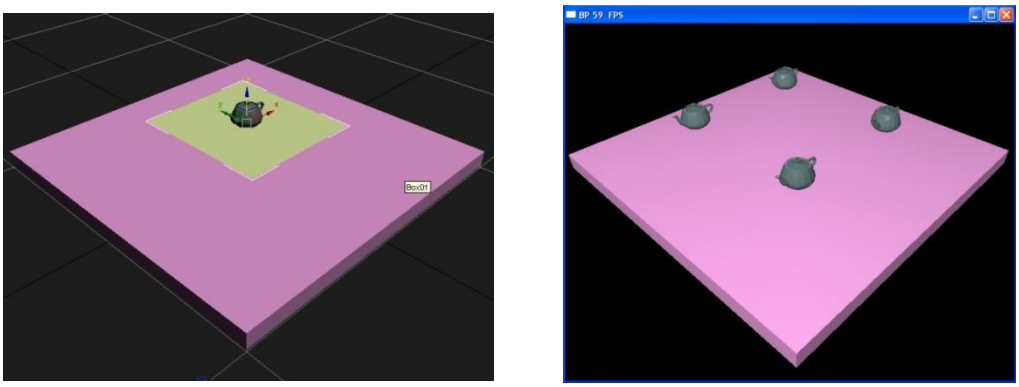
### 4.2.1 Inštancovanie za pomoci polí

Od začiatku existovali aspoň dva postupy ktoré som zvažoval :

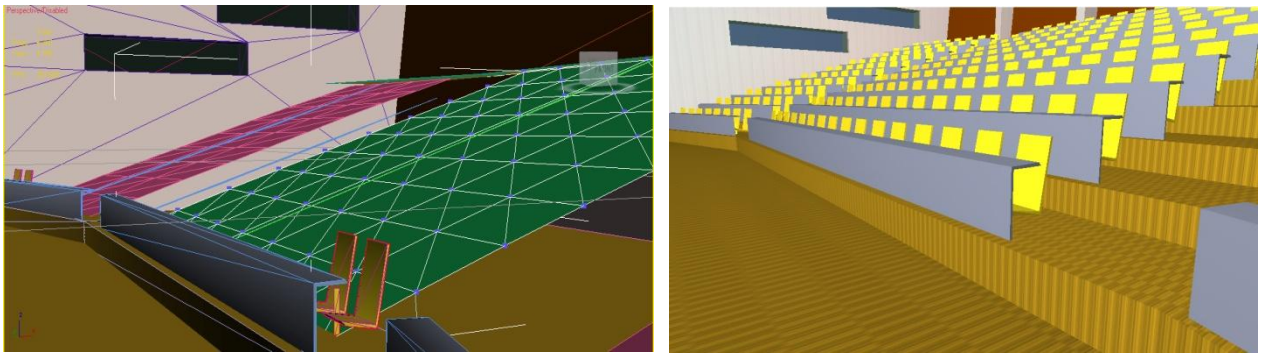
- Prvým bolo použitie objektu v scéne už použitého, vyrátanie jeho ťažiska. Následne jeho nakopírovanie na body z poľa za pomoci posunutia. Existovalo tu však riziko, že v editore človek ovplyvní takzvaný pivot, je to referenčný bod objektu. Posunie ho inam ako do ťažiska, preto som sa rozhodol použiť inú metódu.

- Metódu kopírovania objektov s použitím posunutia pivota, ktorý nám bude dodaný a to len za pomoci transformácií na miesta v poli. Kde poľom sa rozumie objekt zvlášť vytvorený pre tento účel. Je to objekt všeobecnej geometrie, ktorého vrcholy budú nahradené práve pivotmi novo rozkopírovaných objektov. Na záver kopírovania odstránim oba používané objekty (ako originál, tak aj pole ) z pamäte, ak je to treba, originál je jednoducho nahraditeľný ďalším bodom poľa.

  Pre lepšiu názornosť si celý princíp ukážeme na jednoduchej scéne. Na obrázku 1 môžeme vidieť ako vyzerá jednoduchá scéna pred a po úprave kopírovania. Objekt ktorého vrcholy budú použité na rozgenerovanie je v editore nutné nazvať menom kopírovaného objektu a pridať príponu *: array*.



*Obrázok 6:Pohľad na oba operandy kopírovania Vo WISIWIG editore(vľavo) a už rozkopírované objekty vo výslednom programe(vpravo). Poloha pôvodného objektu nie je podstatná.*



*Obrázok 7:Pohľad na časť scény pripravenú na kopírovania Vo WISIWIG editore(vľavo) a už rozkopírované objekty vo výslednom programe(vpravo).*

## 4.3 Výber programovacích nástrojov

Na začiatku vývoja vyvstala otázka pre ktorú platformu sa rozhodnúť. Na vytvorenie programu v jazyku C++ existovalo viacero mne známych variant. Použitie GLUT som vylúčil, veľkosť základného okna vytvoreného za pomoci GLUT sa šplhá na hranicu 32 KB v nekomprimovanej podobe. Použitie knižníc GL a GLU bolo pre tento projekt dostatočné. Použil som teda variantu WinApi.

Ako kompilátor som použil MinGW v spolupráci s vývojovým prostredím DevC++. Toto prostredie je síce mŕtvym projektom, ale pre našu prácu postačuje. Umožňuje aj použitie optimalizácií a nástroja strip executable, ktorý slúži na odstránenie ladiacich informácií a iných nadbytočných dát zo spustiteľného binárneho súboru. V úvahu pripadal ešte nástroj MS Visual Studio, v ktorom však po nastavení kompilátora na najmenšiu veľkosť výstupnej aplikácie som sa dopracoval k najlepšiemu výsledku, taktiež okolo 32 KB, kde DevC++ a MinGW dosiahli 11,5 KB. Tieto údaje sú pre aplikáciu zobrazujúce základné okno a v ňom 1 trojuholník.

## 4.4 Redukcia veľkosti premenných

Prostredie 3DS MAX ako aj OpenGL pracujú primárne s premennými z množiny reálnych čísiel. Je to z dôvodu povahy problematiky ktorej časť postihujú – problematiky 3D zobrazovania. Najjednoduchšie algoritmy na 3D transformácie sú za pomoci násobenia matíc. Tomuto trendu sa postupne prispôsobil aj hardware v podobe grafických kariet, ktoré nám na tento účel dnes primárne slúžia a preto sa mu musíme prispôsobiť aj my. Tento postup však ide proti zadaniu našej práce a to v ohľade veľkosti. Nakoľko reálne čísla sú v jazyku c a c++ reprezentované premennými typu float

a double. Pričom rozlíšenie (množstvo rôznych zobraziteľných hodnôt) je pre náš účel nepoužiteľné. Preto sme zvolili kompromis v podobe komprimácie dát do menších premenných. Ako bude ďalej opísané, týmto spôsobom sme ušetrili veľké množstvo dát.

### 4.4.1 Export farieb

Nástroj export do formátu VRML vyexportuje farbu objektu ako trojicu reálnych čísiel, pre náš účel je použitie tohto typu dát nevhodné, pretože by sme museli použiť dátový typ float, ktorý má veľkosť na bežnej 32 bitovej architektúre 4 B [8]. Pre náš účel bude vyhovujúca premenná typu unsigned char ktorá má rozsah 256 hodnôt. A zaberá 1B. Ušetríme tak : *Počet_objektov * 3 B*
Táto transformácia sa odohrá počas vyberania pre nás podstatných údajov v Python skripte.
Obmedzenie tejto metódy nie je žiadne, samotný WISIWIG editor a väčšina grafických editorov nepodporuje väčšie farebné rozlíšenie ako $2^{24}$ (256*256*256) farieb, ak pominieme alfa kanál danej farby.

### 4.4.2 Export bodov

Rovnako ako pri farbách je to aj s bodmi, situácia je tu však o poznanie zložitejšia. Ako sme si už povedali, reálne čísla sú v prostredí C++ reprezentované ako float/double. Už pri menšom počte vrcholov obsiahnutých v scéne bol zrejmý problém veľkosti takto vyexportovaných dát.

V prvej verzií programu bola na ukladanie dát použitá knižnica <string>, zabezpečovala ukladanie nekonečne dlhých reťazcov, ktoré Python skript upravil do podoby, ktorá bola prijateľnejšia voči vyberaniu nami potrebných dát v programe za pomoci C++. Toto však bolo nevyhovujúce riešenie. Dáta takto zaberali oveľa viac miesta, ako by tomu bolo pri ich uložení vo forme premenných a samotná réžia knižnice aj po odstránení nepotrebných súčastí v nastaveniach kompilátora sa šplhala až k 60 KB.

Druhá verzia programu preniesla implementáciu a teda aj riešenie tohto problému čiastočne na stranu Python skriptu, v ktorom prebieha výber podstatných premenných a ich transformácia do menšej podoby. Tento problém je riešený nasledovne :

- Výber potrebných dát z dokumentu VRML.
- Odstránenie nadbytočných formátovacích znakov.
- Načítanie dát vo forme reálnych čísiel do asociatívneho poľa jazyka Python.
- Vytvorenie dvoch premenných, ktoré nám následne poslúžia pri určení obálky telesa – takzvaného bounding boxu.

- Naplnenie týchto premenných najmenšími a najväčšími súradnicami bodov, nájdených spomedzi všetkých bodov objektu a to tak, že v prvom bode je minimálna hodnota spomedzi všetkých súradníc a každej z osí a v druhej maximálna.
- Tieto dva body tvoria protiľahlé rohy kvádra, ktorý nám obaľuje celé teleso,
- Tento obalový kváder, takzvaný bounding box, nám tvorí uzavretý systém, v ktorom zaokrúhlime každý bod tak, že jeho súradnice prevedieme do hodnôt 0-255 a kde minimálny bod má súradnice [0;0;0] a maximálny [255;255;255] nasledovne:

  Prišli do úvahy 2 algoritmy pre lepšiu presnosť na úkor veľkosti som zvolil algoritmus načrtnutý v[9].

  Prepočet súradníc do bounding boxu prebehne pre všetky súradnice bodu($x_{new}$, $y_{new}$, $z_{new}$) analogicky, zobrazuje ho rovnica (1)

  $$x_{new} = \text{round}\left( \left( x_{old} - x_{min} \right) * 255 / \left( x_{max} - x_{min} \right) \right) \quad (1)$$

  V tomto prípade je potrebné exportovať oba body bounding boxu teda 3*2*4 (24)B na objekt. Druhý algoritmus používa na prepočet uhlopriečku bounding boxu. Je treba preniesť len bod minima a dĺžku uhlopriečky D(vzdialenosť oboch bodov).
  Prenesie sa teda len 3*1*4+1 (12) B na objekt, úspora je mizivá a rozdiel je pri zložitejšej scéne na prvý pohľad zrejmý.

  Prepočet súradníc do bounding boxu prebehne pre všetky súradnice bodu($x_{new}$, $y_{new}$, $z_{new}$) analogicky, zobrazuje ho rovnica (2)

  $$x_{new} = \text{round}\left( \left( x_{old} - x_{min} \right) / \left( d / 255 \right) \right) \quad (2)$$

  K výpočtu potrebujeme vedieť dĺžku telesovej uhlopriečky bounding boxu, ktorej výpočet zobrazuje rovnica (3)

  $$\text{kde } d = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2} \quad (3)$$

- Dáta vyexportované do dátového modelu sú následne funkciami rozgenerované do pôvodného stavu do pamäte. Počas celého tohto procesu sa ušetrí *3*počet_vrcholov_v_scéne*4* B.

K ušetreniu priestoru prispelo aj prepísanie typu poľa indexov(do poľa vrcholov) z premennej typu *int* na premennú typu *unsigned short int,* ktorej veľkosť je polovičná (2B) oproti pôvodným 4B na *int*.

Už len použitie poľa indexov nám šetrí miesto nakoľko viacnásobne použité vertexy sú reprezentované len indexom – vytvorenie trojuholníkov zabezpečuje správne poradie indexov. Je to bezstratová kompresia oproti naivnému postupu, pri ktorom by sme vyberali pre náš program zoznam všetkých (aj duplicitných) vrcholov. Ušetríme teda ešte polovicu na veľkosti použitej premennej.

Toto má však aj nevýhodu, pretože jeden element (či objekt) bude obmedzený na počet vrcholov, ktorý je maximálny rozsah tejto premennej, čo činí 65536 = ($2^{16}$) vrcholov. Toto obmedzenie je však pre naše potreby dostačujúce.

## 4.5 Samorozbaľovací archív - SFX / EXE paker

EXE paker je program ktorý dokáže zabaliť dáta aplikácie. V našom prípade binárneho exe súboru, ku ktorému pridá dáta potrebné na jeho rozbalenie, oreže ho o nadbytočné dáta a súčasti, či opakujúce sa údaje. Následne zbaliť všetko potrebné čím ušetrí podstatnú časť miesta, ktorú takáto aplikácia zaberá na pevnom disku.

Takáto aplikácia sa od klasickej aplikácie odlišuje len veľmi málo a to v dvoch veciach. Jej veľkosť je oveľa menšia ako veľkosť pôvodnej aplikácie a čas na jej spustenie sa o málo predĺži, nakoľko pred jej spustením je potrebné aby bola aplikácia rozbalená do pamäte počítača. Je to posledná z rady úprav, či optimalizácií, ktorú na programe vykonáme.

V našom prípade sme použili program – paker kkrunchy, ktorý z testov vyšiel najlepšie. Je primárne vytvorený na tvorbu intier. Čo je v dnešnej dobre aj hlavné použitie exe pakerov. Porovnania výsledkov jednotlivých pakerov sú v nasledujúcej kapitole 4.5.1.

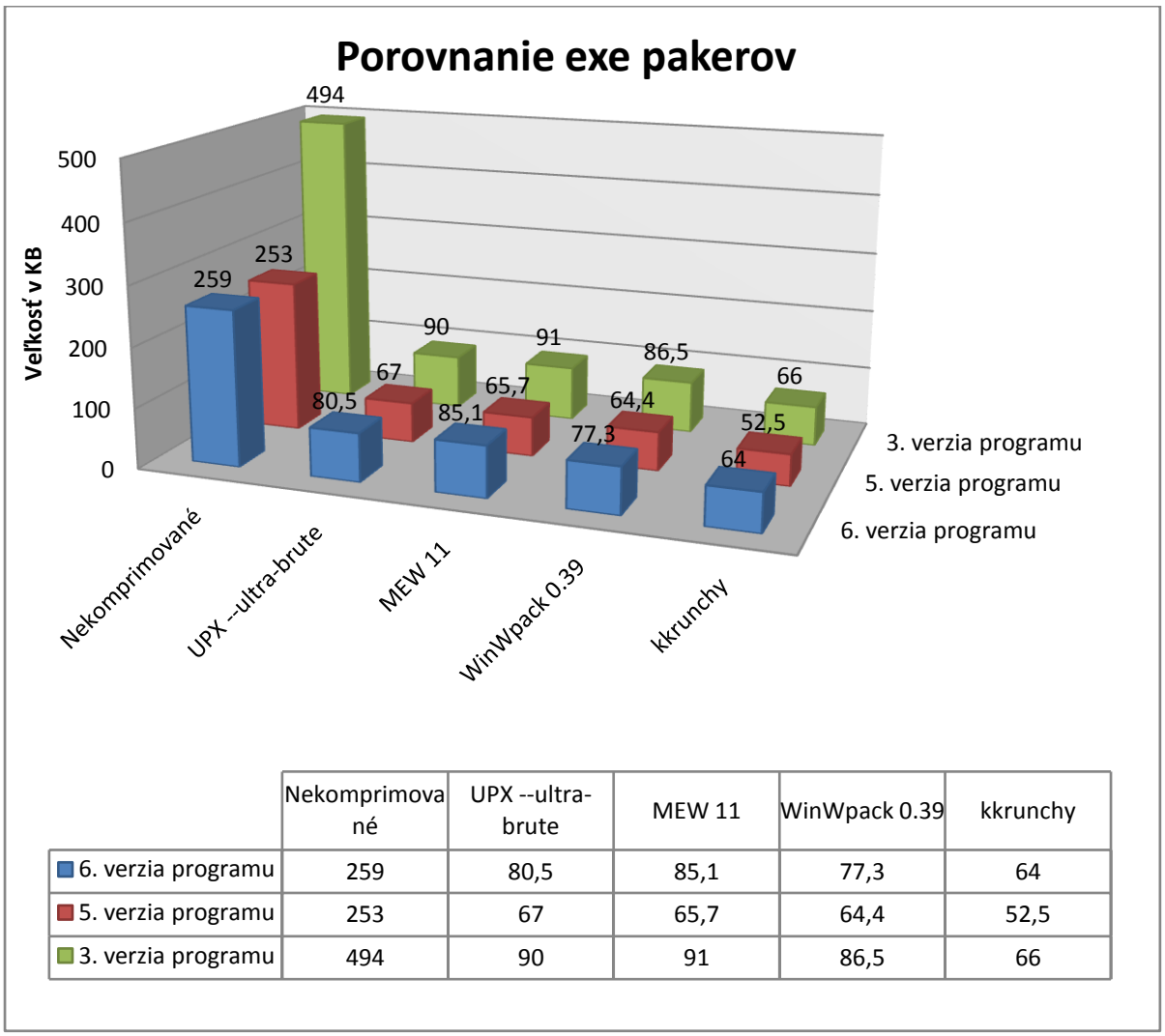
### 4.5.1 Porovnanie exe pakerov na troch verziách programu:

Nasledovné verzie boli zvolené z dôvodu jednoznačnej demonštrácie rozdielov vo veľkosti ktoré nami vykonané úpravy spôsobili.

3. verzia programu používa ako základný dátový typ float, na uskladnenie všetkých priestorových dát scény.

5. verzia používa kompresiu dát do premenných typu unsigned char a je rozšírená o použitie polí a ďalších prvkov vykresľovania.

6. finálna verzia programu – dokončený model, prezentačný mód a ďalšie textúry.




| | Nekomprimované | UPX --ultra-brute | MEW 11 | WinWpack 0.39 | kkrunchy |
|---|---|---|---|---|---|
| 6. verzia programu | 259 | 80,5 | 85,1 | 77,3 | 64 |
| 5. verzia programu | 253 | 67 | 65,7 | 64,4 | 52,5 |
| 3. verzia programu | 494 | 90 | 91 | 86,5 | 66 |

*Obrázok 8: Porovnanie exe pakerov.*

Z grafu je jasne zrejmé že techniky ktoré sme použili nám ušetrila 20,45 % miesta vo výslednom binárnom súbore pri použití najlepšej kompresie(3. a 5 verzia sa dátovo zhodujú) – exe paker kkrunchy.

# 5 Zobrazovací modul

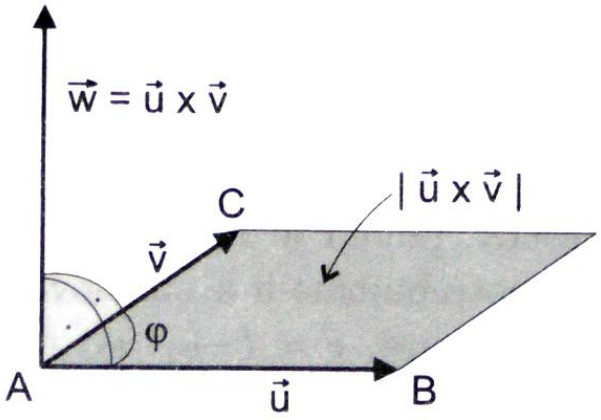
## 5.1 Zobrazovacie módy

OpenGL poskytuje viacero možností vykresľovania objektov - od bodovej reprezentácie, cez sieťovú tzv. wireframe až po polygonálnu. Naše jadro je schopné prepínať medzi týmito tromi. OpenGL taktiež podporuje dva typy základného vykresľovania tieňovania. Sú nimi : flat (ploché) tieňovanie a Gouraudovo tieňovanie.

## 5.2 Vytvorenie normálových vektorov

Na to, aby bolo možné začať počítať osvetlenie, bolo treba najprv vyrátať normálové vektory v každom z vrcholov všetkých trojuholníkov. Normálové vektory sme dopočítali za pomoci vektorového súčinu[10]. Vektorový súčin $\vec{w}$ vektorov $\vec{u}$ a $\vec{v}$ môžeme zapísať pomocou bázových vektorov kartézskej sústavy súradníc. Popisuje ho rovnica (4).

$$\vec{w} = \vec{u} \times \vec{v} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ u0 & u1 & u2 \\ v1 & v2 & v3 \end{bmatrix} = ( u1v2 - u2v1 )\,\vec{i} + ( u2v0 - u0v2 )\,\vec{j} + ( u0v1 - u1v0)\,\vec{k} \qquad (4)$$



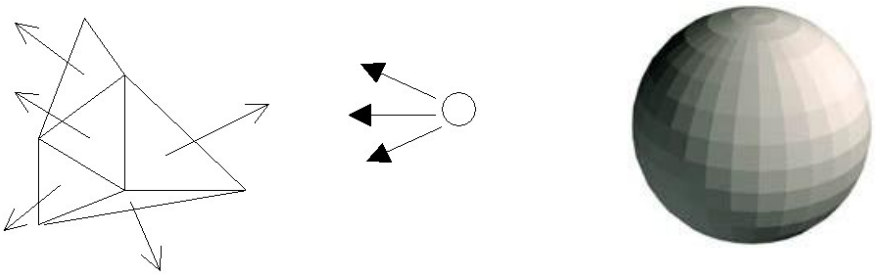

*Obrázok 9:Výsledkom vektorového súčinu je vektor $\vec{w}$ kolmý na rovinu určenú vektormy $\vec{u}$ a $\vec{v}$.*

Pre správne vykresľovanie tieňovania je potrebné aby mali normálové vektory dĺžku 1, nakoľko v našom programe používame a spracúvame neuniformnú zmenu mierky objektov [11], použijeme na toto vstavanú funkciu glEnable(GL_NORMALIZE), ktorá nám toto zabezpečí.
Nevýhodou tejto metódy je vyššia náročnosť pri vykresľovaní, rozdiel však nie je nijak citeľný a čas ušetrený počas modelovania, ktorý by sme museli inak venovať neustálej kontrole počas modelovania, za to stojí.

## 5.3 Tieňovania

### 5.3.1 Ploché tieňovanie

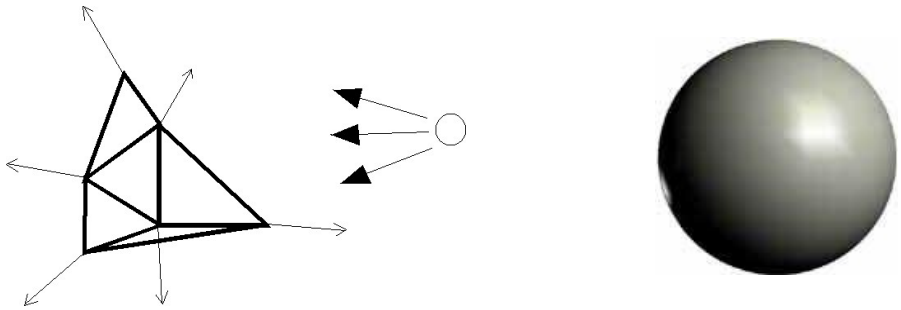
Ploché tieňovanie v porovnaní s Gouraudovým prináša vo väčšine prípadov podstatne horšie vizuálne výsledky čo sa týka estetickej stránky.
Jeho princípom je, že používa normálu tieňovaného trojuholníka na zistenie farby v jednom centrálnom bode trojuholníka, touto farbou následne vykreslí celý trojuholník.



*Obrázok 10:Ukážka princípu tieňovania (vľavo) a jednoduchého objektu vytieňovaného plochým tieňovaním v pravo.*

### 5.3.2 Gouraudovo tieňovanie

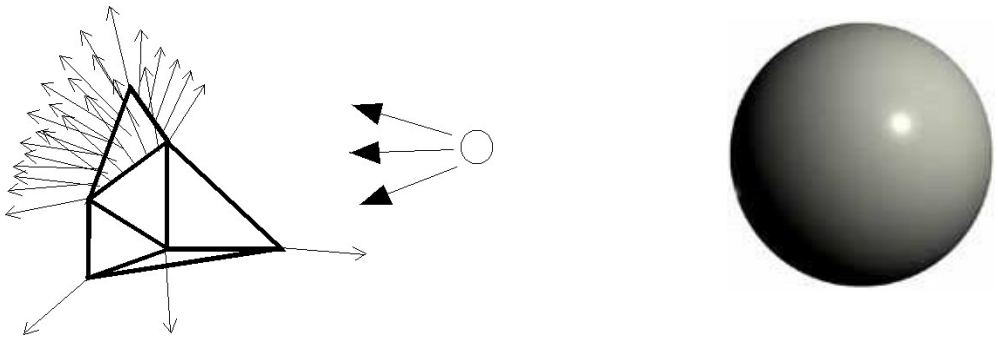
Naše zobrazenie používa primárne Gouraudovo tieňovanie, tak ako ho implementuje OpenGL.
Pre každý trojuholník sa vyhodnotia na základe osvetľovacieho modelu ( ktorý používa nami prepočítané normálové vektory vo vrcholoch), farby v každom vrchole daného trojuholníka či polygónu. Farba medzi nimi je následne vypočítaná ako bilineárna interpolácia [12] v jednotlivých farebných bodoch.



*Obrázok 11:Ukážka princípu tieňovania (vľavo) a jednoduchého objektu vytieňovaného Gouraudovým tieňovaním v pravo.*

### 5.3.3 Ďalšie možnosti - Phongovo tieňovanie

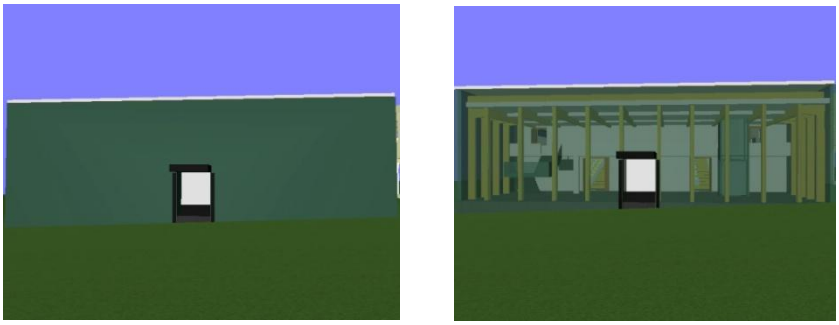
Phongovo tieňovanie je pokročilý spôsob tieňovania, ktorý je však veľmi náročný na výkon hardware, z čoho vyplýva že je pomalšie na výpočet, ale výsledok takto tieňovaného objektu je oveľa kvalitnejší. Výpočet prebieha tak, že farba sa vypočíta za pomoci osvetľovacieho modelu v každom bode vykresľovanej plochy (trojuholníka). K tomuto výpočtu je potrebné poznať normály vo vrcholoch. Pre každý bod plochy sa za pomoci lineárnej interpolácie z vrcholov [13] získa normála na výpočet farby. Zohľadňuje sa zakrivenie plochy, ktoré skoro odpovedá skutočnému zakriveniu. Implementácia tohto riešenia je náročnejšia oproti vyššie opisovaným metódam a v OpenGL sa implementuje za pomoci shaderov.



*Obrázok 12:Ukážka princípu tieňovania (vľavo) a jednoduchého objektu vytieňovaného Phongovým tieňovaním v pravo.*

## 5.4 Alfa blending - priehľadnosť

Táto technika používa prídavný alfa kanál ku trom klasickým farebným kanálom RGB (červenej zelenej a modrej). V modelovanom areáli je značné množstvo priehľadných objektov tvorených zväčša jedným typom skla, obsahujúcim zelený nádych. Aby bolo možné zobraziť scénu realistickejšie, bolo potreba využiť techniky alfa blendingu. Toto si vyžiadalo malú úpravu v štruktúre vykresľovanej funkcie, kde došlo k rozdeleniu vykreslenia najprv na nepriehľadné objekty a následne vykreslenie priehľadných objektov s použitím nastavení depth buffera (hĺbkového zásobníka). Výsledkom tohto procesu je efekt, pri ktorom dôjde ku sčítaniu farebných zložiek priehľadných objektov s ostatnými v pomere, ktorý im určuje práve alfa kanál.



*Obrázok 13:Ukážka vypnutého (vľavo)a zapnutého(vpravo) blendingu.*

## 5.5 FPS

### 5.5.1 Výpočet

V záhlaví okna sa ako meno programu zobrazuje aj FPS – frames per seccond teda počet snímkov za sekundu, (ďalej len FPS). Ako preklad napovedá je to počet snímkov ktoré sa zobrazia za jednu sekundu.

Výpočet tejto hodnoty prebieha tak, že za každý raz čo je zavolaná zobrazovacia funkcia zvýši sa hodnota počítadlo a skontroluje sa či neprebehla sekunda od poslednej zmeny záhlavia, ak áno, zapíše sa hodnota do záhlavia, počítadlo sa vynuluje a proces prebieha znova. Táto hodnota sa používa ako informačný údaj pri ladení. Je to najrýchlejší spôsob ako zistiť či po zmene nejakej súčasti programu došlo ku zmene rýchlosti vykresľovania a teda k urýchleniu, či spomaleniu behu celej aplikácie.

Vyvstáva otázka, prečo nám hodnota po spustení aplikácie pravdepodobne preskakuje medzi hodnotami 59 – 61 fps(odlišuje sa od použitého zobrazovacieho zariadenia, pre CRT môže byť viac). Je to z dôvodu že počítač na ktorom aplikácia beží je pravdepodobne dostatočne rýchli na to aby stíhal zobrazovať aplikáciu v obnovovacej frekvencií monitora. Využíva vertikálnu synchronizáciu[14]. Táto funkcia je primárne zapnutá a zabezpečuje že každá vykreslená snímka je

synchronizovaná so snímkou ktorú vykreslí monitor. Toto zabraňuje vzniku artefaktov v podobe vodorovných čiar napríklad pri otáčaní v programe okolo vlastnej osi.

### 5.5.2 Prepínanie vertikálnej synchronizácie

Údaj FPS je použitý pre prispôsobivú zmenu rýchlosti prekresľovania, nakoľko pri prechode časťami scény ktoré sú náročnejšia na výpočet môže FPS klesnúť pod prípustnú hodnotu, ktorú som v našom prípade zvolil na 35 FPS. Pri tejto FPS sa automaticky vypne vertikálna synchronizácia, čím stúpne rýchlosť prekresľovania približne o 30-50 percent (hodnota odhadnutá na základe viacerých experimentov).

Tým pádom sa dostane aplikácia do oblasti štandardnej kvality rýchlosti prekresľovania čo je približne 45 – 60 FPS. Ak sa FPS vyšplhá nad 65 FPS vertikálna synchronizácia sa zasa automaticky zapne. Pričom vodorovné artefakty pri týchto rýchlostiach nie sú až na toľko výrazné aby prebili prínos zrýchlenia programu. Veľký rozdiel 35 – 65 FPS bol zvolený aby dochádzalo k čo najmenšej šanci na preskakovanie medzi týmito dvoma stavmi, kde by sa táto funkcia striedavo zapínala a vypínala.

Vypínanie vertikálnej synchronizácie je v programe zabudované primárne na tlačidlo 4. Je však potrebné ho držať nakoľko program si sám riadi prepínanie tohto nastavenia.

### 5.5.3 Konštantná rýchlosť animácie

FPS sa nám mení v závislosti s náročnosťou časti vypočítavanej scény. Ak by sme animovali pohyb postavy o rovnaké posunutie či otočenie, len na základe počtu vykreslených snímkov, došlo by k efektu pri ktorom by postava mimovoľne zrýchľovala v jednoduchých častiach scény a spomaľovala v zložitých. Aby sme tomu zabránili, implementovali sme mechanizmus, ktorý zabezpečuje to, že postava prejde rovnaký úsek, či sa otočí o rovnaký uhol, za rovnaký čas nezávisle na FPS.

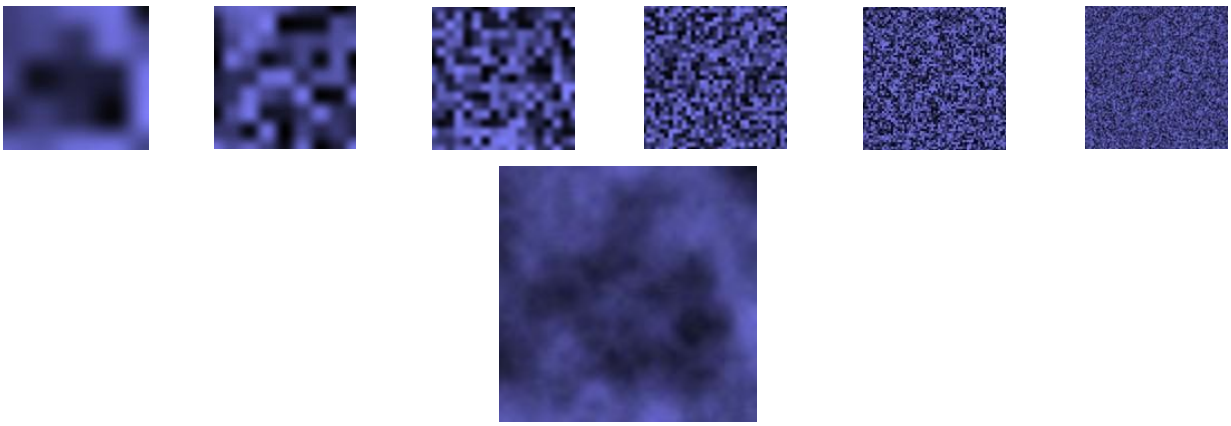
Využijeme k tomu FPS pri ktorom sa postava pohybuje prirodzenou rýchlosťou a vydelíme toto číslo aktuálnym číslom FPS. Získaným koeficientom násobíme všetky transformácie a rotácie postavy, medzi dvoma snímkami. Získaným efektom je plynulý pohyb bez zmeny rýchlosti. Nevýhodou tejto metódy je to že FPS sa aktualizuje len raz za 1 sekundu, pričom rozdiel medzi FPS v predchádzajúcej sekunde a terajšej môže byť dosť značný a kým sa rýchlosť postavy ustáli môže dôjsť k dočasnému trhaniu. Toto sa však dá ešte viac obmedziť s pomocou zníženia času za ktorý aktualizujeme FPS na 500ms. Tento čas je pre náš účel vyhovujúci, avšak obmedzí FPS na čísla deliteľné dvomi.

# 6 Textúrovací modul

Textúrovací modul zabezpečuje vytvorenie textúr. Základom textúry je farba ktorá sa získava z dát spolu s objektom. Táto je následne nanesená na objekt a na jej základe sú v textúrovacom module nadefinované textúry, následne pridávané na povrch objektov. Tento modul využíva voľne dostupných funkcií pre generovanie perlinovho šumu z internetového článku [15]. Tieto funkcie poskytujú základ pre kvalitné textúry bez použitia iných vonkajších zdrojov. Keď vezmeme v úvahu, že každý obrázok vo forme bitmapy o veľkosti 512 x 512 farebných bodov presahuje sám o sebe veľkosť nášho binárneho súboru, je použitie generovaných textúr jedinou možnosťou.

## 6.1 Perlinov šum

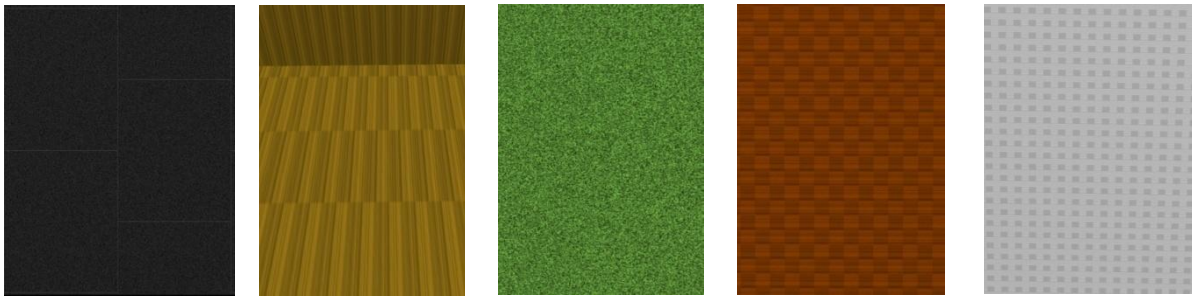
Perlinov šum je pseudonáhodná funkcia, ktorá môže byť vyjadrená v n rozmeroch, nás bude však zaujímať hlavne jeho dvojrozmerná podoba. Jeho výhodou je že pri určitých podmienkach vytvára v sebe vždy rovnako veľké útvary a taktiež to, že je možné ho opakovať. Tieto útvary majú pri iných parametroch inú veľkosť. Ich skladaním môžeme dosiahnuť funkciu, ktorá obsahuje prvky všetkých súčastí. Toto je najlepšie vidieť na obrázkoch dole.



*Obrázok 14 : Horný rad obrázkov znázorňuje Perlinovu šumovú funkciu ako 2D textúru v rôznych frekvenciách a dolný obrázok je súčtom všetkých týchto súčastí(oktáv).Obsahuje v sebe všetky ich zložky a teda aj detailnosť na viacerých úrovniach priblíženia.*

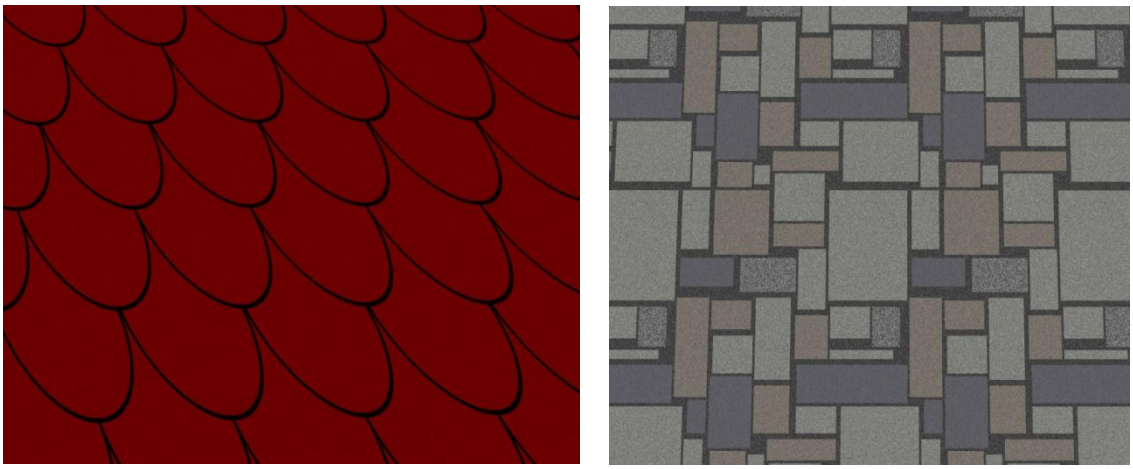
Vo svojej aplikácií využívam Perlinov šum hlavne na pridanie detailnosti textúram, pretože jednofarebné textúry, alebo textúry obohatené len o jednoduché tvary nie sú veľmi vizuálne príťažlivé. Ak by sme použili Perlinov šum ako trojdimenzionálnu textúru mohli by sme dostať až neuveriteľne detailnú textúru mramoru, či s použitím alfa kanála (viz kapitola 4.4) textúru mrakov či ohňa. Ak by sme postúpili ešte o krok ďalej do štvrtého rozmeru, mohli by sme tieto mraky, či oheň

aj animovať. V našom programe je jeho použitie hlavne na jednoduchú dvojdimenzionálnu textúru kamenných podláh, či dreveného obloženia. U všetkých týchto textúr bolo treba vyriešiť a vymyslieť nadväznosť textúry tak programovo, ako aj následne za pomoci automatického generovania mapovacích koordinátov.



*Obrázok 15:Ukážka textúr generovaných aplikáciou, ich veľkosť je meniteľná minimálnou úpravou zdrojového kódu, základom je však rozlíšenie 512 x 512 x32b.*

Rôzne tvary v textúrach boli dosiahnuté jednoduchými podmienkami a matematickými funkciami.



*Obrázok 16:Textúry napodobujúcej tvar škridiel sme dosiahli pomocou jednoduchej rovnice(5), ktorá popisuje kružnicu(vľavo) a textúra vykladanej dlažby je kombinácia podmienok obmedzujúcich zápis určitého odtieňa farieb do textúry.*

$$d = \sqrt{x^2 + y^2} \qquad (5)$$

Ako môžeme vidieť , textúry sa skladajú z mnohých malých súčastí ktoré sa opakujú. Bez intervencie by dochádzalo k jasnému a viditeľnému aliasingu[16]. V tento problém sa dá riešiť dvoma priamočiarymi metódami. Prvou z nich by bol antialiasing, ktorý je však výpočetne pomerne náročný a platformovo závislý. Je to metóda, ktorá vypočítava scénu v dvojnásobnom rozlíšení a na jeho základe vytvorí obraz v našom nižšom rozlíšení.

Druhou metódou ktorá aj bola použitá a je implementovaná je metóda mipmapingu[16]. Pri tejto metóde sa textúry uložia v dvojnásobnej veľkosti keď polovica je pôvodná textúra a ostatok tvoria menšie verzie tejto textúry. OpenGL potom automaticky mení túto textúru v závislosti na vzdialenosti od kamery. Táto metóda je oveľa menej výpočetne náročná ako antialiasing.

# 7 Implementácia

## 7.1 Python skript:

Celý projekt sa skladá z viacerých programových častí, prvou z nich je skript v jazyku Python, ktorý slúži na vytvorenie zdrojového modulu zo zdrojového súboru v jazyku VRML. Tento VRML súbor dostaneme ako výstup WISIWIG editora, akým je napríklad 3D Štúdio MAX. Ja som použil verziu 2009. Tento program je poskytovaný na stránkach výrobcu (Autodesk) ako 30 dňová trial verzia, alebo ako študentská licencia na dobu 6 mesiacov. Skript v jazyku Python zabezpečuje hlavne výber nami potrebných kľúčových dát a ich následnú premenu do menšej podoby (zabalenie do bounding boxov –kapitola 3.3.2). Ako svoj jediný parameter prijíma meno zdrojového súboru typu VRML buď ako relatívnu, alebo ako absolútnu cestu k nemu. Dáta pochádzajúce z WISIWIG editora musia byť vo forme trojuholníkov, teda zoznamov bodov a indexov do nich. Jeho výstupom je zdrojový prikompilovateľný modul v jazyku C++ s názvom model.cpp, ktorý využívame hlavným programom.

## 7.2 Hlavný program:

Hlavný program je rozdelený do niekoľkých zdrojových súborov. Je napísaný v jazyku C/C++. Tento jazyk som si zvolil kvôli jeho modularite, širokej použiteľnosti štandardných knižníc, rozšíriteľnosti, malých rozmerov výsledného binárneho súboru a v neposlednom rade aj priamej podpory OpenGL.

Aplikácia je postavená na platforme WinApi, ktorého kostra je temer rovnaká s malými odchýlkami pre všetky podobné aplikácie, má dobrú podporu pre užívateľské rozhranie a teda je najprijateľnejšie pre naše potreby.

Hlavný modul s názvom **main.cpp** neprijíma žiadne parametre, inicializuje okná a OpenGL. Je v ňom umiestnená hlavná vykresľovacia slučka a z neho sa volajú ostatné funkcie umiestnené v iných moduloch. Dá sa charakterizovať aj ako zobrazovací modul.

Hlavičkový modul **main.h** zabezpečuje prepojenie modulov, vkladanie štandardných knižníc a knižníc OpenGL, zároveň sú v ňom definované zložitejšie dátové štruktúry použité pre celý program a niekoľko makier použitých na sprehľadnenie zdrojového kódu.

Textúrovací modul **textures.cpp** je bližšie popísaný v kapitole 5 avšak obsahuje aj ďalšie súčasti. Práve tu sú definované textúry a ich rôznorodosť na základe operácií s poľami farebných hodnôt. A je tu aj funkcia ktorá rozhoduje akému telesu má byť priradená aká textúra.

Dátový modul **model.cpp** je vytvorený za pomoci Python skriptu a obsahuje dáta celej scény. Tieto dáta sú načítané a uložené do štruktúry vhodnej pre naše ďalšie spracovanie, výpočet normálových vektorov v zobrazovacom module a následné vykreslenie. Prebieha tu taktiež proces rozgenerovania inštancií ktorý je popísaný bližšie v kapitole 3.2.1

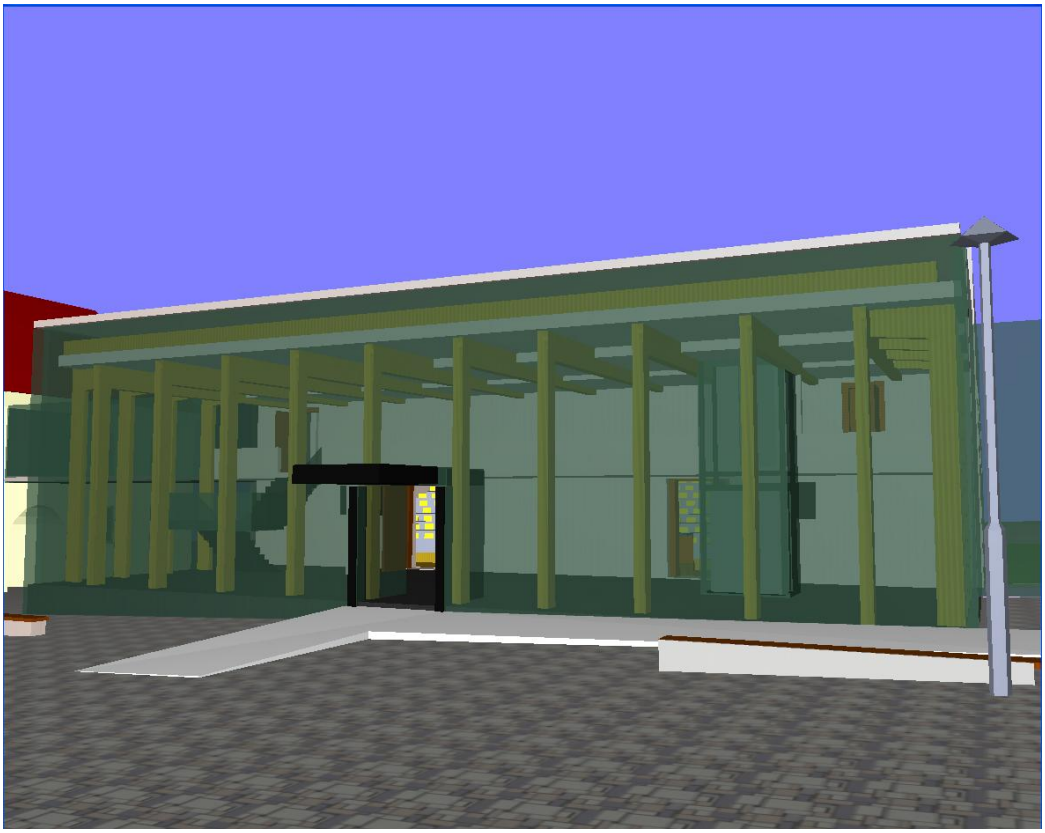
Dátový modul **camera.cpp** je modul v ktorom sú uložené dáta generickej kamery. Obsahuje súradnice bodov a natočenia kamery, ktoré sa načítajú pri každom spustení programu. Program generuje výstupný súbor camera.txt do ktorého ukladá súradnice bodov ktoré si užívateľ sám zadá. Je teda jednoducho možné zmeniť generickú cestu kamery – jednoduchou úpravou kódu modulu camera.cpp na základe dát camera.txt a znovu zostavením programu.

# 8 Dosiahnuté výsledky

*Obrázok 17: Skutočná fotografia zo zadných lavíc prednáškovej miestnosti D205 areálu Božetěchova.*

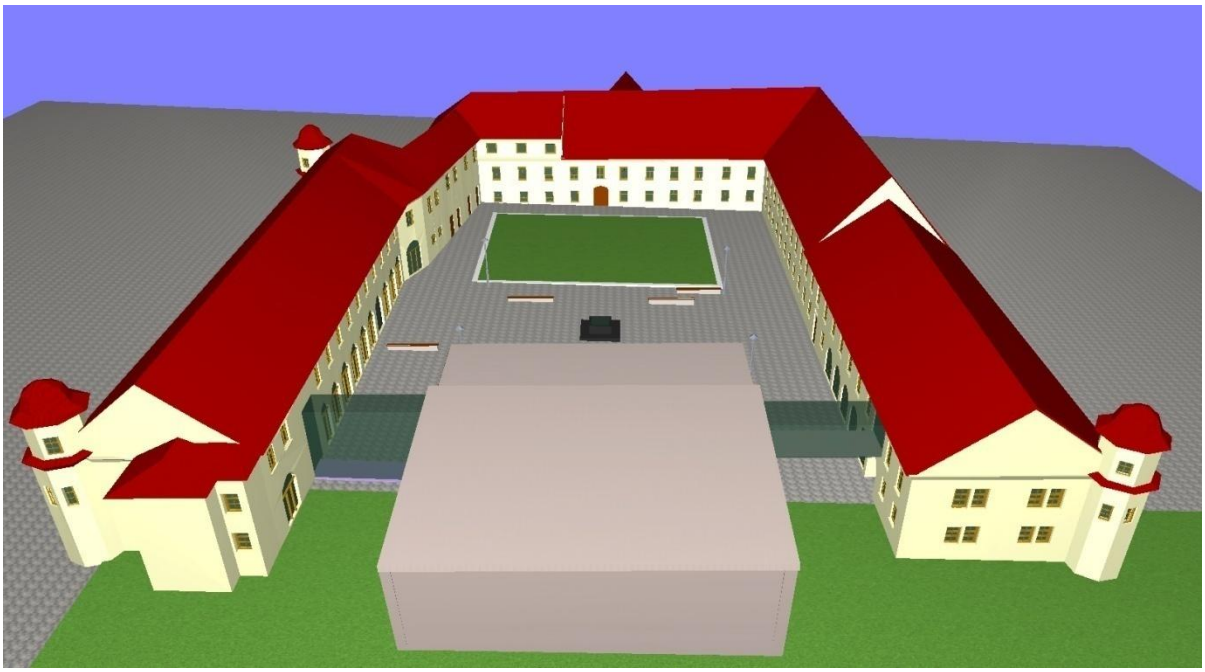
*Obrázok 18: Pohľad z môjho programu umiestnený tak aby zobrazoval rovnakú časť ako fotografia.*

*Obrázok 19: Fotografia prednáškovej miestnosti D105*



*Obrázok 20: Pohľad na D105 v našej aplikácii.*

*Obrázok 21:Porovnanie pohľadu na vstupnú bránu vľavo fotografia, v pravo naša aplikácia.*



*Obrázok 22:Pohľad na vymodelovanú časť areálu.*

Pohľady vyššie sú len vybranými časťami ktoré sa najviac podobajú realite. Ostatné časti areálu by sa do nami stanovenej hranice 64 KB už nevošli. Preto ostali vnútorné priestory fakulty, okrem D105 prázdne.

# 9 Záver

Hlavnou úlohou tohto projektu bolo zoznámiť sa s problematikou tvorby grafických aplikácií s obmedzenou veľkosťou a implementovať takúto aplikáciu pokiaľ možno v čo najlepšej kvalite zobrazenia a najnižším dátovým objemom.

Postupne sme si ukázali najpoužívanejšie a najzákladnejšie techniky minimalizácie modelu, generovania inštancií a vytvárania procedurálnych textúr. Načrtli sme si použitie niektorých prvkov ktoré zlepšujú vizuálnu kvalitu výstupu a ktoré sú súčasťou zobrazovacieho modulu. V neposlednom rade som sa počas procesu tvorby aplikácie zoznámili s mnohými prvkami a úskaliami počítačovej grafiky a to hlavne grafickej knižnice OpenGL, pri ktorých bolo vždy treba dbať na ich veľkosť.

Za hlavný prínos tejto práce považujem skúsenosti nadobudnuté tvorbou aplikácie, i keď samotná aplikácia nie je samoúčelná. Detailnosť modelu je obmedzená hranicou 64 KB, ktorá bola dodržaná. Ak by tomu tak nebolo, do budúcnosti by bolo možné použiť ju mnohými spôsobmi. Okrem virtuálnej prechádzky po fakulte, by bolo možné implementovať aj databázu ciest či ich generovanie a tak napomôcť prípadným návštevníkom nájsť konkrétnu miestnosť či kanceláriu hľadaného vyučujúceho. Taktiež skript vytvárajúci dátový model nie je obmedzený množstvom dát a je možné ho použiť v spolupráci s ostatnou časťou projektu s malými úpravami, pre temer akúkoľvek vizualizáciu, zatiaľ však len staticky koncipovanej scény.

Okrem iného by bolo možné rozšíriť aplikáciu mnohými spôsobmi, od implementácie tieňov, cez zlepšenie kvality textúr, použitie shaderov a zmenu spôsobu vykresľovania z imediate módu na vertex arrays, prípadne vertex buffer objects, ktorá nebola implementovaná. Pôvodný návrh s ňou rátal len okrajovo a vyžadovala by si väčší zásah do zdrojového programu. Pre nami vykresľovaný objem dát nie je až tak kritická. Taktiež by bolo možné zrýchliť vykresľovanie orezávaním pohľadovej plochy(viewing frustum).

Bolo by možne uvažovať o rozšírení na 128 KB, čím by sa popustila uzda nárokov, ale zvýšila by sa detailnosť a možnosť implementovať nové prvky. Možností je neúrekom a každá z nich by mohla byť témou bakalárskej práce ako napríklad automatické generovanie nábytku a doplnkov v kanceláriách s prihliadnutím na pohyb osôb.

Práca v štádiu, v ktorom je teraz, spĺňa všetky body zadania.

# Literatúra


[1] Key frame In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 22 February 2007, 29 March 2010 [cit. 2010-04-10]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Key_frame>

[2] The Demoscene : Computers new realm. *Digitale Kultur* [online]. 2005, č.1, [cit. 2010-04-30].
Dostupný z WWW: <http://www.digitalekultur.org/files/dk_whatisthedemoscene.pdf>

[3] Demoscene#Demo types In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 22 January 2003, 25 April 2010 [cit. 2010-04-30]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Demoscene#Demo_types>

[4] TUREK, Michal, et al. *NeHe OpenGL Tutoriály* [online]. c2002-2008, aktualizováno 2008-3-30 [cit. 2010-04-04]. Dostupné z WWW: <http://nehe.ceske-hry.cz>

[5] MOLOFEE, Jeff. *NeHe OpenGL Tutoriály* [online]. c2000 [cit. 2010-04-04]. Lekce 10 - Vytvoření 3D světa a pohyb v něm. Dostupné z WWW: <http://nehe.ceske-hry.cz/tut_10.php>

[6] VRML In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 9 October 2002, 12 Mar. 2010 [cit. 2010-04-04]. Dostupné z WWW: <http://en.wikipedia.org/wiki/VRML>

[7] KŘÍŽ, Jan. *3ds max 6 : Praktické postupy*. Brno : Computer Press, 2004. 344 s. ISBN 80-251-0329-3.

[8] Single-precision In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 4 April 2001, 3 April 2010 [cit. 2010-04-04]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Single-precision>

[9] KRÁL, Tomáš. *Minimalistická reprezentace modelu areálu Božetěchova*. Brno, 2009. 33 s. Diplomová práce. FIT VUT v Brně. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=8772>

[10] KRŠEK, Přemysl ; ŠPANĚL, Michal. *Základy počítačové grafiky* [online]. Brno : Vysoké učení technické v Brně, Fakulta informačních technologií, Ústav počítačové grafiky a multimédií, 2008 [cit. 2010-04-05]. Osvětlení a stínování 3D objektů, Dostupné z WWW: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IZG-IT/lectures/izg_3d_osvetleni_stinovani.pdf>

[11] Scaling (geometry) In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 13 May 2004, 18 April 2010 [cit. 2010-05-01]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Scaling_(geometry)>

[12] Bilinear interpolation In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 23 May 2004, 5 April 2010 [cit. 2010-04-30]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Bilinear_interpolation>

[13] Linear interpolation In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 20 December 2002, 10 March 2010 [cit. 2010-04-30]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Linear_interpolation>

[14] RICHARD, Wright; SWEET, Michael. *OpenGL SuperBible*. second edition. Indianopolis, Indiana : Waite Group Press, 2000. 696 s. ISBN 1-57169-164-2

[15] *www.sorgonet.com* [online]. 27-01-2003 [cit. 2010-04-30].
PROCEDURAL TEXTURES using OPENGL.
Dostupné z WWW: <http://www.sorgonet.com/linux/noise_textures/>

[16] ŽÁRA, Jiří; BENEŠ, Bedřich; SOCHOR, Jiří. *Moderní počítačová grafika*. druhé. Brno : Computer Press, 2004. 609 s. ISBN 80-251-0454-0

# Zoznam príloh

Príloha 1. Manuál
Príloha 2. CD

# Príloha 1. Manuál

Aplikácia je napísaná pod rozhraním WinApi, tým pádom je obmedzená, len na činnosť pod operačným systémom Windows.

# Pohyb v scéne:

Pre pohyb v scéne som zvolil klasické ovládanie avšak len za pomoci klávesnice. Nič to však neuberá na dynamickosti a plynulosti pohybu v scéne.

**L – spustenie prezentačného módu**

**W – pohyb kamery dopredu**
**S – pohyb kamery vzad**
**A – úkrok kamery doľava**
**D – úkrok kamery doprava**
**Q – pohyb kamery nahor**
**E – pohyb kamery nadol**

**Šípka hore(up arrow) – otočenie kamery hore**
**Šípka dole(down arrow) – otočenie kamery dole**
**Šípka doľava(left arrow) – otočenie kamery doľava**
**Šípka doprava(right arrow) – otočenie kamery doprava**

# Zmena zobrazovania:

**1 – výplňové zobrazovanie - štandardné**
**2 – zobrazovanie hrán trojuholníkov (wireframe)**
**3 – bodové zobrazenie**
**4 – manuálne vypínanie vertikálnej synchronizácie (v programe je implementované automatické prepínanie tejto vlastnosti v závislosti na podpore hardware a rýchlosti vykresľovania – fps, teda pre manuálne vypnutie je potreba držanie tlačítka)**
**5 – manuálne zapínanie vertikálnj synchronizácie**
**6 – zapnutie Gouraudovho osvetlenia (štandardne zapnuté)**
**7 – zapnutie plochého osvetľovacieho modulu (flat shading)**
**8 – zapínanie textúr (štandardne zapnuté)**
**9 – vypínanie textúr**

**F1 –zapnutie/vypnutie celoobrazovkového módu defaultné rozlíšenie preň je 800 x 600**

**ESC – ukončenie programu**

**P – pridá do zoznamu kamerových bodov aktuálnu pozíciu a pohľad(smer pohľadu)**

**L – cyklí donekonečna medzi bodmi uloženými v zozname kamerových bodov (prezentačný mód). Dá sa prerušiť hoc akou klávesou ovládajúcou natočenie či pohyb, pri opätovnom stlačení klávesy L sa prezentácia pustí prechodom na posledný nezobrazený bod.**

**I – vynuluje počítadlo bodov, prezentácia začne od prvého bodu.**

**M – vymaže zoznam kamerových bodov.**

**K – zmena štýlu prezentácie kamery**

# Príloha 2. CD