



UNIVERZITA KOMENSKÉHO, BRATISLAVA  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

## EDITOR VOXELOVEJ GRAFIKY

Bakalárska práca

**Bratislava, 2013**

**Marek Kružliak**



UNIVERZITA KOMENSKÉHO, BRATISLAVA  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

## EDITOR VOXELOVEJ GRAFIKY

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Martin Samuelčík, PhD.

**Bratislava, 2013**

**Marek Kružliak**



## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Marek Kružliak

**Študijný program:** aplikovaná informatika (Jednooborové štúdium, bakalársky I. st., denná forma)

**Študijný odbor:** 9.2.9. aplikovaná informatika

**Typ záverečnej práce:** bakalárska

**Jazyk záverečnej práce:** slovenský

**Názov:** Editor voxelovej grafiky

**Ciel:** Hlavným cieľom práce je vytvorenie aplikácie pre prácu v pravidelnej mriežke v trojrozmernom priestore. Prvou úlohou bude zadefinovanie základných editačných nástrojov pre prácu s bunkami (voxlami) a ich implementácia. Nástroje by mali zadávať a meniť základné atribúty buniek ako farba, obsadenosť, namapovaná textúra. Výsledná aplikácia by mala obsahovať aj import a export pomocou existujúceho formátu pre zálohovanie mriežky a atribútov. Zároveň by bolo vhodné pridať import trojuholníkového meshu a implementovať jeho voxelizáciu. Súčasťou práce bude aj prehľad existujúcich podobných riešení a ich porovnanie s vytvorenou aplikáciou.

**Anotácia:** Objemová grafika patrí popri rastrovej grafike k dôležitým oblastiam počítačovej grafiky. Jednou z jej dôležitých oblastí je voxelová grafika, ktorá sa zaobrá reprezentáciou objektov v trojrozmernej pravidelnej mriežke. Pre takto reprezentované objekty je vhodné mať nástroj pre ich vytváranie, modifikáciu a zmenu parametrov. Kedže voxelová grafika je len priamym rozšírením dvojrozmernej rastrovej grafiky, nástroje a postupy používane pre editáciu rastrovej grafiky (napr. Adobe Photoshop alebo Gimp) sa dajú ľahko rozšíriť aj do 3D. Vytvorenie práve takýchto nástrojov bude základným cieľom tejto práce. Ako aj pri rastrových editoroch, pre úplnosť editora bude potrebné pridať ďalšiu funkcionality ako import a export. Zaujímavým problémom bude pridanie mapovania textúr na voxelový objekt.

**Vedúci:** Mgr. Martin Samuelčík, PhD.

**Katedra:** FMFI.KAI - Katedra aplikovanej informatiky

**Vedúci katedry:** doc. PhDr. Ján Rybár, PhD.

**Dátum zadania:** 15.10.2012

**Dátum schválenia:** 15.10.2012

doc. RNDr. Mária Markošová, PhD.

garant študijného programu

študent

vedúci práce

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Ďakujem RNDr. Martinovi Samuelčíkovi, PhD. za  
to, že sa ujal vedenia mojej bakalárskej práce a za  
jeho cenné rady pri programovaní a pri písaní práce.

# Abstrakt

Primárnym cieľom práce je opísať návrh a implementáciu praktického nástroja na tvorbu a editovanie voxelových objektov a scén. V práci sa taktiež venujeme vysvetleniu základných pojmov objemovej grafiky a jej využitiu mimo vedeckej sféry, teda v moderných herných enginoch a podobne. Ďalej uvádzame príklady existujúcich riešení v danej oblasti, ich kladné a záporné stránky, a následné využitie získaných poznatkov z ich testovania. Práca taktiež opisuje zaujímavé algoritmické problémy, ktoré sa počas tvorby práce vyskytli, ich riešenie a implementáciu opísaného riešenia.

**Kľúčové slová:** voxel , objemová grafika, voxelizácia

# Abstract

Primary goal of this paper is to describe design and implementation of voxel editing tool. We also describe basic concept of volume graphics and its usage in commercial sphere, like modern game engines, animation etc. Next we mention examples of already existing solutions in this field, their strong and weak points and their influence on this work. In this paper we also analyze interesting algorithmic problems, which appeared during implementation phase.

**Keywords:** voxel, volume graphics, voxelization

# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Motivácia . . . . .	1
1.2	Cieľ . . . . .	2
<b>2</b>	<b>Objemová grafika</b>	<b>3</b>
2.1	Objemové dátá . . . . .	4
2.2	Voxel . . . . .	5
2.3	Porovnanie s povrchovou grafikou . . . . .	5
2.3.1	Nevýhody . . . . .	6
2.3.2	Výhody . . . . .	6
2.4	Komerčné využitie . . . . .	8
2.4.1	Herný priemysel . . . . .	8
2.4.2	Umenie . . . . .	9
<b>3</b>	<b>Existujúce riešenia</b>	<b>10</b>
3.1	Digitálne sochárstvo . . . . .	10
3.2	Editory voxelovej grafiky . . . . .	11
3.2.1	Paint3D . . . . .	11
3.2.2	Voxel3D . . . . .	13
3.2.3	3D Dot Game Hero Editor . . . . .	15
3.2.4	Zoxel . . . . .	17

3.2.5	Sproxel . . . . .	18
3.2.6	Qubicle . . . . .	20
3.3	2D grafické editory . . . . .	22
3.4	Zhrnutie poznatkov z testovania . . . . .	22
<b>4</b>	<b>Implementácia</b>	<b>24</b>
4.1	Použité technológie . . . . .	24
4.1.1	C++ . . . . .	24
4.1.2	OpenGL . . . . .	25
4.1.3	DevIL . . . . .	25
4.1.4	Microsoft Visual Studio 2012 . . . . .	26
4.1.5	Assimp . . . . .	26
4.1.6	pugixml . . . . .	26
4.2	Rendering . . . . .	27
4.2.1	Rendering voxelu . . . . .	27
4.2.2	Rendering objektu . . . . .	28
4.2.3	Rendering scény . . . . .	28
4.3	Voxelové Objekty . . . . .	28
4.3.1	Základné tvary . . . . .	29
4.3.2	Booleanovské operácie nad objektami . . . . .	32
4.3.3	Transformácie objektov . . . . .	33
4.4	Nástroje . . . . .	34
4.4.1	Selekcia . . . . .	35
4.4.2	Štetec . . . . .	37
4.4.3	Guma . . . . .	37
4.4.4	Kvapátko . . . . .	38
4.4.5	Výplň . . . . .	38
4.4.6	Kamera . . . . .	39

4.4.7	Posun objektov . . . . .	40
4.5	Import/Export . . . . .	40
4.5.1	Polygónový model . . . . .	40
4.5.2	Bitmapa . . . . .	44
4.5.3	Voxelové formáty . . . . .	44
4.5.4	XML . . . . .	45
<b>5</b>	<b>Výsledky</b>	<b>47</b>
5.1	Metodika merania . . . . .	47
5.2	Rendering . . . . .	48
5.3	Voxelizácia . . . . .	49
5.4	Voxelové objekty . . . . .	50
5.5	Nástroje . . . . .	51
5.6	Import/Export . . . . .	52
5.7	Užívateľské prostredie . . . . .	54

# Zoznam obrázkov

2.1	Voxelový tank s namapovanou textúrou . . . . .	7
2.2	Ukážka voxelového terénu z hry Comanche: Maximum Overkill . . . . .	9
3.1	Paint3D . . . . .	13
3.2	Voxel3D . . . . .	15
3.3	3D Dot Game Hero Editor . . . . .	16
3.4	Zoxel . . . . .	18
3.5	Sproxel . . . . .	20
3.6	Qubicle . . . . .	22
4.1	Elipsoid . . . . .	30
4.2	Triedny diagram pre nástroje . . . . .	35
4.3	Vizualizácia fungovania algoritmu getVoxel . . . . .	37
4.4	Vizualizácia algoritmu zistovania, či sa bod náchádza v objekte . . . . .	43
5.1	Vyrenderovaná guľa . . . . .	49
5.2	Výsledky voxelizácie . . . . .	50
5.3	Základné objekty . . . . .	51
5.4	Booleanovské operácie . . . . .	51
5.5	Transformácie obejktov . . . . .	52
5.6	Test exportu . . . . .	53
5.7	Import bitmapy . . . . .	54

5.8	Užívateľské prostredie . . . . .	55
-----	----------------------------------	----

## Zoznam tabuliek

5.1	Tabuľka počtov voxelov na objekt . . . . .	48
5.2	Rýchlosť renderovania objektov namerané v FPS . . . . .	48
5.3	Rýchlosť voxelizácie rôznych modelov . . . . .	50

# Zoznam skratiek

CSG	Constructive Solid Geometry
GUI	Graphical User Interface
API	Application Programming Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
IDE	Integrated Development Environment
DOM	Document Object Model
XML	Extensible Markup Language
RLE	Run Length Encoding

# Kapitola 1

## Úvod

Kocka sa stala hlavným symbolom tejto práce, a tak sme nami navrhnutú aplikáciu nazvali *Cubo*, čo vlastne aj znamená po španielsky kocka. Tento názov sa nápadne podobá aj na anglické slovo *cube*, od ktorého sme sa pôvodne odrazili. Takýmto spôsobom chceme výrazne naznačiť, čomu sa program venuje.

Táto kapitola v stručnosti popisuje existujúce skutočnosti, ktoré nás podnietili k vytvoreniu tejto aplikácie a následne stanovujeme ciele, ktoré chceme v práci dosiahnuť.

### 1.1 Motivácia

Objemová grafika, alebo inak tiež voxelová grafika, je pomerne mladá oblasť počítačovej grafiky a len nedávno našla svoje uplatnenie aj v komerčnej sfére, čo je prvý dôležitý krok k jej skutočnému rozvoju. Jej ďalšia budúcnosť stojí a padá na hardvérovej podpore a dúfame, že je len otázkou času, kedy vzniknú prostriedky umožňujúce vývojárom spracovávanie veľkého množstva voxelov v reálnom čase.

Ako som už spomenul vyššie, voxelová grafika našla svoje uplatnenie v komerčnej sfére ako napríklad v herných enginoch, animácii alebo tvorbe lego skulptúr a podobne. Jej ohromnou výhodou v hrách je, že sa objekty skladajú z veľkého

množstva častíc a navyše uchovávajú v sebe objem, čo sa dá využiť napríklad pri reálnejšej simulácii fyziky alebo úplnej deštrukcii herného prostredia. Poznanie, že voxelová grafika sa objavuje v hrách priamo implikuje potrebu nástroja na vytváranie herných modelov. Voxelový editor môže mať tiež iné využitie a to na tvorbu zaujímavých 3D kockatých renderov, ktoré sú v dnešnej dobe veľmi populárne, a tak sa často objavujú na rôznych plagátoch alebo internetových reklamách.

Krása kociek uchvátila mnohých a nie jeden baží po vytvorení si svôjho vlastného voxelového modelu. Rozhodli sme sa teda o zhotovenie praktického kresliaceho nástroja v 3D, uplatňujúc skúsenosti získané pri práci v 2D grafických programoch.

## 1.2 Ciel

Našim hlavným cieľom je vytvorenie editačného nástroja na úpravu voxelových objektov a scén. Tento cieľ sa skladá z viacerých podcieľov, ktoré sme si stanovili.

Jedným z cieľov práce je uviesť čitateľa do problematiky zadefinovaním základných pojmov objemovej grafiky, akými sú napríklad *voxel*, *objemové dátá* a podobne. V rámci tohto cieľa chceme čo najstručnejšie opísť výhody a nevýhody voxelovej grafiky, jej dôležité postavenie v rámci počítačovej grafiky a taktiež jej využitie mimo vedeckej sféry.

Ďalším cieľom je opísanie existujúce riešenia, ich silné a slabé stránky a ich vplyv na našu prácu. Následne chceme tieto skúsenosti aplikovať v implementačnej fáze, kde chceme podrobne opísť jednotlivé časti programu, ich možnosti a taktiež metódy, ktoré sme použili pri ich vytváraní.

Na záver chceme zhrnúť výsledky jednotlivých častí programu, ktoré sme v práci dosiahli a taktiež možnosti ďalšieho vývoja aplikácie v budúcnosti.

# Kapitola 2

## Objemová grafika

Vďaka pokroku vo vývoji hardvéru sa revolučné prístupy k počítačovej grafike postupne stávajú realitou, a to aj pre bežného majiteľa počítača. Dobrým ilustratívnym príkladom pre nás môže byť rastrová grafika, ktorá sa objavila v období, kedy hardvérové inovácie dopomohli k reálnemu presunu z matematicky reprezentovanej vektorovej grafiky k pamäťovo náročnejšej rastrovej grafike. Podobným príkladom je aj objemová grafika, ktorej rozvoj opísali vo svojom článku v roku 1993 Kaufman, Cohen a Yagel [KCY93].

Objemová grafika teda predstavuje prechod od takzvanej povrchovej grafiky, ktorá objekty reprezentuje pomocou spojитých primitív ako sú polygóny alebo free-form povrchy, k reprezentácii objektu pomocou diskrétnych objemových štruktúr [BSc02], ktorých základná dátová jednotka je voxel opísaná nižšie v tejto kapitole. Taktiež môžeme povedať, že objemová grafika je významným poľom počítačovej grafiky a je 3D analógiou rastrovej grafiky. Jej uplatnenie sa našlo nielen vo vizualizácii rôznorodých vedeckých dát, ale aj v komerčnom využití ako napríklad herné enginy, animácie alebo umenie.

Avšak objemová grafika nadalej zostáva menšinou na poli počítačovej grafiky, pretože jednou z hlavných prekážok jej širšieho použitia je fakt, že niektoré z fundamentalných operácií ešte potrebujú vylepšenie teoretických základov a taktiež

objemová grafika, na rozdiel od tej povrchovej, nemá podporu grafických akcelerátorov. Na druhej strane sa mnohé z techník objemovej grafiky stali populárnymi v radoch vývojárov najrôznejších grafických aplikácií, pretože objavili nesporné výhody tohto prístupu v určitých oblastiach. [Pal08]

## 2.1 Objemové dátá

Ako bolo spomenuté vyššie, najbežnejším prístupom k reprezentácii objektov v trojrozmernom priestore je povrchová reprezentácia objektov pomocou dvojrozmerných primitív. Tento prístup je však v mnohých prípadoch nedostačujúci, pretože sa takto strácajú informácie o vnútornej štruktúre 3D telies. Nastolený problém riešia objemové dátá, ktoré tieto informácie obsahujú. Vďaka tejto vlastnosti je objemová reprezentácia vhodný kandidát pre medicínske, geologické, biologické, archeologické a iné vedecké dátá získané napríklad v prípade medicíny z tomografu.[EHS06] V hennom priemysle je takáto reprezentácia vhodná na detajlnú deštrukciu okolitého prostredia alebo je taktiež vhodná na modelovanie amorfín objektov ako sú napríklad oblaky, dym a podobne.

Vizualizácia objemových dát sa markantne líši od vizualizácie povrchových telies, keďže objemové dátá neobsahujú žiadnu explicitnú informáciu o povrchu a taktiež je nevyhnutne potrebné nejako nahliadnuť do vnútra objektu. Cieľom vizualizácie objemových dát je teda vytvoriť mechanizmus, ktorý nám umožní extrahovať zmysluplné infomácie z objemových dát pomocou ich reprezentácie, manipulácie a renderingu.

Objemové dátá sa najčastejšie uchovávajú v trojrozmernom poli a teda sú zvyčajne reprezentované ako štvorica  $(x, y, z, v)$ , kde  $(x, y, z)$  sú poloha v 3D priestore  $v$  je *value*, teda hodnota danej objemovej jednotky, čo môže byť napríklad 0 alebo 1, kde hodnota 0 indikuje prázdne miesto a 1 indikuje objekt. Táto hodnota

môže byť aj komplexnejšia informácia, ako napríklad farba, hustota, tlak alebo vektor zrýchlenia v danej polohe  $(x, y, z)$ . [KCY93] Alternatívne dátové štruktúry na uchovávanie a manipuláciu objemových dát sú napríklad octrees, voxelové riedke matice, 'voxel runs' alebo nepravidelné mriežky. Tieto štruktúry sú zvyčajne vhodné na dobrý manažment pamäťových zdrojov alebo majú iné výhody napríklad v detekcii kolízií.

## 2.2 Voxel

Elementárnu jednotkou objemovej grafiky je voxel. Je priamou 3D analógiou pre jednotku dvojrozmerného priestoru v pravidelnej mriežke, teda pixel. Podobne ako slovo pixel (picture element) slovo voxel je len skratkou pre zložitejšie pomenovanie volume element. Ako bolo dostatočne opísané v predchádzajúcej kapitole, každý voxel okrem svojej polohy v priestore obsahuje aj nejakú merateľnú hodnotu.

Analógia, pre voxeli s pixelmi, sa zachováva aj pri uchovávaní voxelov v pravidelnej mriežke. Teda zatial čo sú pixely uchovávané vo *frame buffery*, voxelы sa uchovávajú vo *volume buffery* alebo tiež nazývanom *3D raster*, či *objemový frame buffer*.

Zadefinovaním tohto pojmu môžeme teraz hovoriť o voxelovom priestore alebo tiež o voxelovej grafike.

## 2.3 Porovnanie s povrchovou grafikou

Spomenuli sme už majoritné rozdiely medzi povrchovou a voxelovou grafikou, ktoré priamo vyplývajú z ich pomenovania. V tejto časti sa teda zameriam na porovnanie týchto dvoch prístupov detajlnejšie s dôrazom na výhody a nevýhody objemovej grafiky oproti povrchovej.

### 2.3.1 Nevýhody

Pri reprezentácii scény sú voxely usporiadane v pravidelnej trojrozmernej mriežke, teda hovoríme o diskrétnom priestore. Tento fakt spôsobuje mnohé komplikácie analogické ku komplikáciám v rastrovej grafike.

Jedným z hlavných problémov je poškodenie, teda modifikácia alebo strata informácií o objekte pri jeho transformácii. Pri rotácii objektu o uhol  $\alpha \neq k\frac{\pi}{2}, k \in \mathbb{Z}$  vznikajú diery. Podobne aj pri škálovaní o koeficient  $k > 1$  je očakávaný vznik dier a pri škálovaní o koeficient  $k < 1$  nastáva strata informácie. Kedže tieto problémy súvisia taktiež s rastrovou grafikou, vzniklo dostačujúce množstvo techník (supersampling, bilineárna interpolácia a pod.) ako tento problém riešiť s viac-menej uspokojivým výsledkom.

Kedže voxelové objekty sú získavané vzorkovaním, nevyhnutne vznikajú artefakty. Tie vidieť najmä pri bližšom zazoomovani na danú oblasť. Teda môžme povedať, že voxelové objekty sú nepresné a sú len diskrétnou aproximáciou pôvodných objektov, kde hustota 3D mriežky určuje ich presnosť.

Prítomnosť objemu vo voxelových objektoch spôsobuje ďalší problém charakteristický pre voxelovú grafiku, a tým je pamäťová a procesová náročnosť. Pomerne malý objekt s rozmermi  $512^3$  pozostáva z približne  $1.35 \cdot 10^8$  voxelov. Ak by sme každý voxel reprezentovali iba jediným bytom, tak pre takýto objekt by bolo potrebné 128MB pmäte. Tieto veľké objemy priamo spôsobujú, že voxelová grafika je náročná aj na processing. Príkladom môže byť rendering, transformácie alebo aj voxelizácia.

### 2.3.2 Výhody

Okrem niekoľkokrát spomenutého faktu, že objemová grafika obsahuje aj informácie o vnútornej štruktúre objektov, nám voxelový priestor ponúka ďalšie nesporné

benefity.

Jedným z nich je jednoduchosť rôznych operácií, ako je napríklad sekanie, orezávanie alebo aj booleanovské operácie medzi objektami. Zatiaľ čo pri povrchovej grafike sú tieto operácie pomerne komplexné a zložito implementovateľné, vo voxelovej grafike sa tieto operácie posúvajú až na úroveň voxelov a teda predstavujú triviálny problém. Ak si zvolíme Constructive Solid Geometry (CSG) ako modelovaciu paradigmu, tak je táto vlastnosť voxelových objektov ohromnou výhodou.

Ďalšou výhodou je, že voxelová grafika nie je citlivá na členitosť objektov. Pri väčzej členitosti objektu v povrhovej grafike narastá počet polygónov a teda aj požiadavky na zdroje, zatiaľ čo voxelový objekt si svoju zložitosť zachováva konštantnú. Podobne aj pri mapovaní textúr (znázornené na obrázku 2.1) má objemová grafika výhodu, pretože na voxelový objekt stačí namapovať textúru iba raz, kedy sa vypočíta farba pre každý voxel a v ňom sa aj nadalej uchováva, zatiaľ čo mapovanie textúr na polygónový respektíve povrchový objekt sa vykonáva zvyčajne až na konci renderovacej pipeline.



Obr. 2.1: Voxelový tank s namapovanou textúrou

## 2.4 Komerčné využitie

Technologický pokrok umožnil, že objemovú grafiku je možné realizovať aj na bežnom-širokodostupnom hardvéri. Toto otvorilo dverka pre komerčnú sféru technologickej rozvoja, a tak sa voxely objavujú nielen ako pomôcka pre zobrazenie pseudonáhodných čísel v teoretickej informatike, ale aj ako stavebné častice herného prostredia, či nástroj na tvorbu detajlných skulptúr.

### 2.4.1 Herný priemysel

Prvou firmou, ktorá sa odvážila využiť voxely ako stavebný prvak pre svoj herný engine bola spoločnosť *NovaLogic* v roku 1992 v hre *Comanche: Maximum Overkill* [Mob13]. Išlo o letecký simulátor, ktorý voxelovú technológiu použil na vytvorenie detajlného vykreslenia okolitého terénu, aký je na obrázku 2.2. V roku 1996 táto spoločnosť za podobným účelom vytvorila populárnu bojovú hru *Delta Force*.

V roku 2000 začal pracovať programátor Ken Silverman spolu s Tomom Dobrowolskym na svojom prvom plne voxelovom hernom engine s názvom *Voxlap*. [SD06] Ten je teraz spolu so zdrojovými kódmi voľne dostupný na svojej oficiálnej stránke a ponúka vykreslovanie voxelového priestoru v pomerne dobrom rozlíšení.

Hra, ktorá najviac spopularizovala voxelovú grafiku vo svojej upravenej podobe, vznikla v roku 2010 s názvom *Minecraft*. Princípom hry je dolovať v hlbokom podzemí bloky rôznych stavebných materálov. Takýmto spôsobom hra využíva najdôležitejšiu vlastnosť voxelovej grafiky, ktorou je uchovanie informácie o vnútornom obsahu objektu.



Obr. 2.2: Ukážka voxelového terénu z hry *Comanche: Maximum Overkill*

### 2.4.2 Umenie

Informatika a obzvlášť počítačová grafika je natoľko ohromujúca, že svojím vplyvom zasahuje i do umenia. Nie je teda prekvapením, že voxely si našli svoje miesto aj medzi umelcami.

Dobrým príkladom je napríklad aj digitálne sochárstvo (*Voxel sculpting*). To umožňuje pre umelca slobodu modelovania digitálnych sôch bez akýchkoľvek topologických obmedzení a vytvárať komplexné detajly takmer z ničoho. Táto sloboda je dosiahnutá vďaka tomu, že úprava sôch nie je založená na deformácii povrchu, ale na vytváraní a vyplňovaní objemu. [Shp11]

Voxelart je ďalším smerom digitálneho umenia, ktoré je 3D obnožou pre pixelart. Na rozdiel od digitálneho sochárstva vo voxelarte sa kladie dôraz na každý vložený voxel, ktorý dotvára celkový dojem z diela. Zvyčajne takéto výtvory bývajú aj omnoho jednoduchšie, kedže ich tvorba je náročnejšia.

Svojou jedinečnosťou zaujali voxely aj animátorov a na internete je možnosť nájsť pomerne veľké množstvo zaujímavých kockatých animácií.

# Kapitola 3

## Existujúce riešenia

Napriek vzrastajúcej popularite voxelovej grafiky neexistuje doposiaľ mnoho kvalitných nástrojov na ich editáciu. Preto sme vybrali niektoré z nich, ktoré nám svojim charakterom najviac vychovávajú. Celkovo môžeme rozdeliť takéto programy do dvoch skupín. Jednou sú programy slúžiace na digitálne sochárstvo s voxelovým základom a druhou sú nástroje zámerne zobrazujúce každý voxel.

### 3.1 Digitálne sochárstvo

Digitálne sochárstvo je pomerne nová a zaujímavá metóda 3D modelovania. Umožňuje pridávanie najjemnejších detajlov a v prípade voxelových programov aj bez ohľadu na topológiu modelu. Vďaka tomu aj napriek pomerne krátkej existencii sa táto metóda stala veľmi populárnou medzi tvorcami detajnej 3D grafiky. Voxelové sochárstvo sa prevažne využíva na tvorbu modelov s fotorealistickým až hyperrealistickým vzhľadom, ktoré slúžia ako predlohy do filmov, hier, priemyselného dizajnu a taktiež ako realistické ilustrácie a podobne. Nechceme sa však zamerať na tento smer modelovania, keďže nevyužíva voxely ako základnú stavebnú jednotku, ale len ako nástroj k zlepšeniu detajlov modelov alebo ako nástroj umožňujúci slobodnejšiu tvorbu.

Zoznam zaujímavých voxel-based programov pre digitálne sochárstvo:

- **3Dcoat** Program, ktorý zadefinoval pojem *Voxel Sculpting*.
- **Acropora** Procedurálny voxelový modeler.
- **Crysis Terrain editor** Program na tvorbu členitých terénov do počítačovej hry *Crysis*.

## 3.2 Editorы voxelovej grafiky

V tejto časti sa zameriavame na programy, ktoré na rozdiel od nástrojov na digitálne sochárstvo nevyužívajú voxely len ako pomôcku k dosiahnutiu detajlného výsledku s pomerne vysokým levelom slobody tvorby, ale ako primárnu stavebnú jednotku 3D objektov. Takéto nástroje neumožňujú veľké rozlíšenia voxelových priestorov a teda nie sú ani zamerané na detail modelov. Toto spôsobuje, že užívateľ má pod kontrolou každý voxel scény, ale aj to, že sú viditeľné rôzne artefakty, obzvlášť zubatý okraj objektov. Takýto stav nemožno nazvať nevýhodou, keďže mnoho fanúšikov voxelov ostré hrany obľubuje, pretože práve viditeľné kocky dodávajú voxelovej grafike to správne čaro.

Nasledujúci prehľad obsahuje vzorku voxelových editorov, ktoré sa nám podarilo nainštalovať a korektne spustiť. Taktiež obsahuje ich stručný opis s výhodami a nevýhodami, ktoré sme zaznamenali pri krátkom testovaní.

### 3.2.1 Paint3D

Daný program predstavuje jednoduchú 3D alternatívu pre program MS Paint [Cor10]. Interface aplikácie je zobrazený na obrázku 3.1.

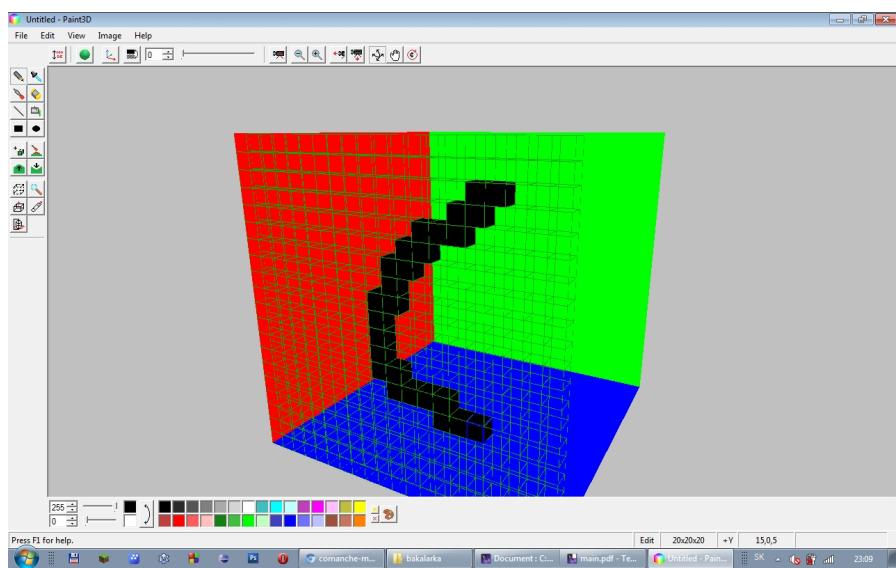
## Výhody:

- **Jednoduchosť.** Na webovej stránke tohto programu tvorcovia proklamujú, že je nástroj jednoduchý na používanie. Do istej miery je to aj pravda, avšak pri testovaní sme museli využiť niektoré internetové zdroje, aby sme pochopili, ako program funguje.
- **Nástroje.** V ponuke sú rôzne užitočné nástroje a je ich taktiež primerané množstvo.
- **Priehladnosť.** Program podporuje, ako jediný z testovaných, alfa transparentiu voxelov.
- **Spolupráca sa 2D grafickými programami.** Tento nástroj umožňuje funkcie copy/paste z iných 2D grafických programov, ako je napríklad MS Paint od spoločnosti *Microsoft*.
- **Kresliace módy.** Program ponúka viaceré módy kreslenia voxelov. Či už po rezoch v rôznych osiach alebo priestorovým kreslením štetcom s nastaviteľnou hrúbkou.
- **Export/Import.** Program podporuje široké množstvo voxelových formátov.
- **Skriptovanie.** Asi najväčšou výhodou tohto nástroja je skriptovanie, ktoré sa dá v grafike a vo voxelovej obzvlášť dobre využiť pri tvorbe inak pracne vytvorených modelov.

## Nevýhody:

- **Zahlcovanie procesora.** Program už pri spustení a počas celého behu vyťaží procesor na takmer 100%. Toto by nemal robiť žiadnen rozumný program pri pokojovom stave.

- **Plná verzia je platená.** Ak by ste chceli využívať všetky možnosti programu, musíte si priplatiť 20\$. Naštastie je trial verzia voľne stiahnuteľná a to, čo ponúka je dostačujúce na prácu.
- **Jeden objekt v scéne.** Možnosť pridávania viacerých objektov do scény neumožňuje väčšina programov.



Obr. 3.1: Screenshot z programu Paint3D

### 3.2.2 Voxel3D

Jedná sa o program od spoločnosti *Everygraph* [Eve08]. Interface aplikácie je zoobrazený na obrázku 3.2.

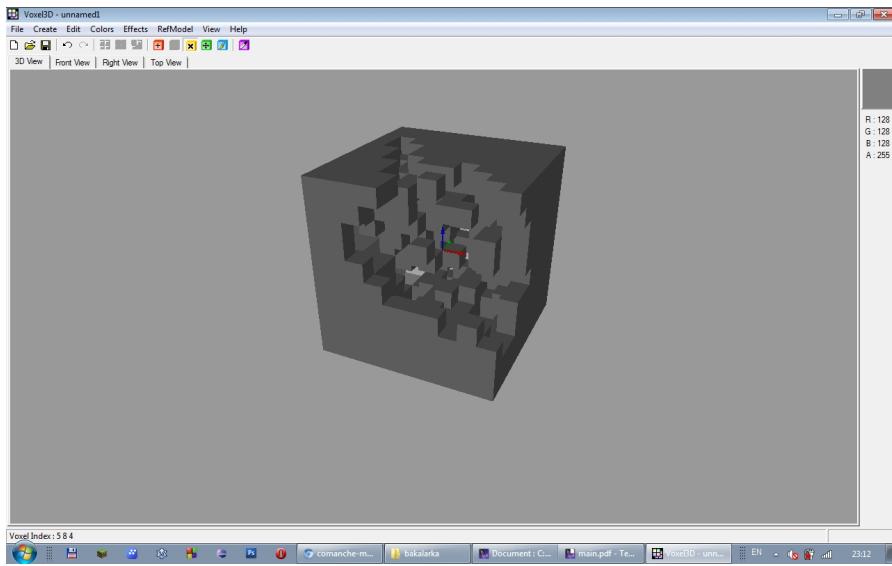
#### Výhody:

- **Voxelizácia.** Ako jediný z testovaných ponúka možnosť voxelizácie 3D meshu. Taktiež umožňuje počas tvorby zobraziť tento mesh spolu s voxelizovaným objektom a podľa ľubovôle ho umiestňovať.

- **Jednoduchý interface.** V programe sa dá ľahko zorientovať, a preto je vhodný aj pre neskúsených užívateľov.
- **4 rôzne náhľady.** Táto funkcia je bežná v 3D modelovacích nástrojoch, avšak toto je jediný z testovaných voxelových editorov, ktorý ju obsahuje.
- **Import/Export.** Výhodou programu je, že má vlastný binárny a aj ASCII formát na ukladanie dát. Taktiež program ponúka aj export do .obj formátu. Nevýhodou však je, že program nepodporuje žiadne iné bežne používané voxelové formáty.

### Nevýhody:

- **Rýchlosť.** Tento parameter dosahuje nedostatočné hodnoty najmä pri vytváraní prázdnego objektu. Už pri pomerne malých rozmeroch vytvorenie objektu trvá iritujúco dlho. Pri testovaní sme vytvárali objekt s počtom voxelov  $32^3$  a ďalší objekt mal počet voxelov  $64^3$ . Vytvorenie prvého objektu trvalo približne 6 sekúnd, zatiaľ čo vytvorenie druhého trvalo približne 48 sekúnd.
- **Neošetrené vstupy.** Napriek tomu, že je program pomalý aj pri malých rozmeroch, jeho interface umožňuje vytvoriť objekt s počtom voxelov  $(2^{32})^3$ .
- **Zdĺhavé modelovanie.** Očakávame, že tvorba voxelových objektov nebude najrýchlejšia, avšak kreslenie voxelov v tomto programe je o to zdĺhavejšie, že umožňuje kresliť iba po rezoch objektom.
- **Jeden objekt v scéne.** Možnosť pridávania viacerých objektov do scény neumožňuje väčšina programov.



Obr. 3.2: Screenshot z programu Voxel3D

### 3.2.3 3D Dot Game Hero Editor

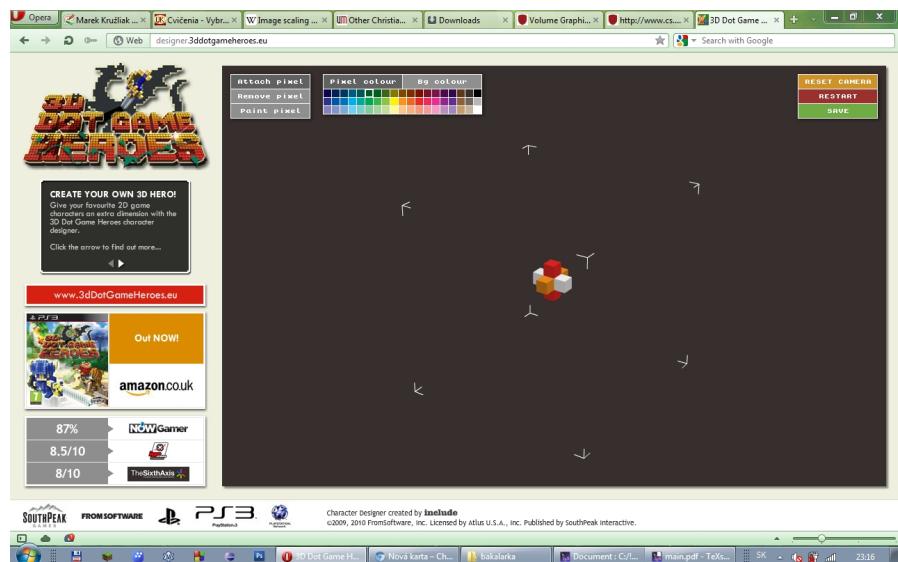
Táto aplikácia je skôr online editor, pre tvorbu voxelových hrdinov do hry 3D Dot Game [Inc09]. Interface aplikácie je zobrazený na obrázku 3.3.

#### Výhody:

- **Dostupnosť cez prehliadač.** Tento fakt je aj výhodou, ale aj nevýhodou daného programu, pretože je dostupný výlučne cez prehliadač a neexistuje desktopová verzia.
- **Jednoduchosť použitia.** Táto vlastnosť je zaručená najmä vďaka malému množstvu nástrojov a možností softvéru.
- **Render a animácia.** Ak v programe zvolíme možnosť *save*, automaticky sa vyrenderuje balík výstupných súborov, ktoré sa nám ponúknu na stiahnutie. Tento balík obsahuje 2 rendery s pozadím a bez pozadia, jednu animáciu objektu (ide iba o rotáciu okolo vlastnej osi) a jeden wallpaper.

## Nevýhody:

- **Ponuka nástrojov.** Ako bolo spomenuté, program ponúka iba chudobné množstvo editovacích nástrojov v počte 3.
- **Obmedzenie farieb.** Taktiež aj farebná paleta je obmedzená na niečo viac ako tucet farebných odtieňov.
- **Obmedzené rozmery.** Tvorený objekt je ohraničený bounding boxom, ktorý nemožno nijak prekročiť, ani zmeniť jeho rozmery.
- **Export/Import.** V ponuke aplikácie nie je žiadnen import a exportovať je možné iba render, ako bolo spomenuté vyššie.
- **Jeden objekt v scéne.** Možnosť pridávania viacerých objektov do scény neumožňuje väčšina programov.



Obr. 3.3: Screenshot z programu 3D Dot Game Hero Editor

### 3.2.4 Zoxel

Zoxel je kvalitný výtvar od jediného človeka menom Graham King [Kin13]. Interface aplikácie je zobrazený na obrázku 3.4.

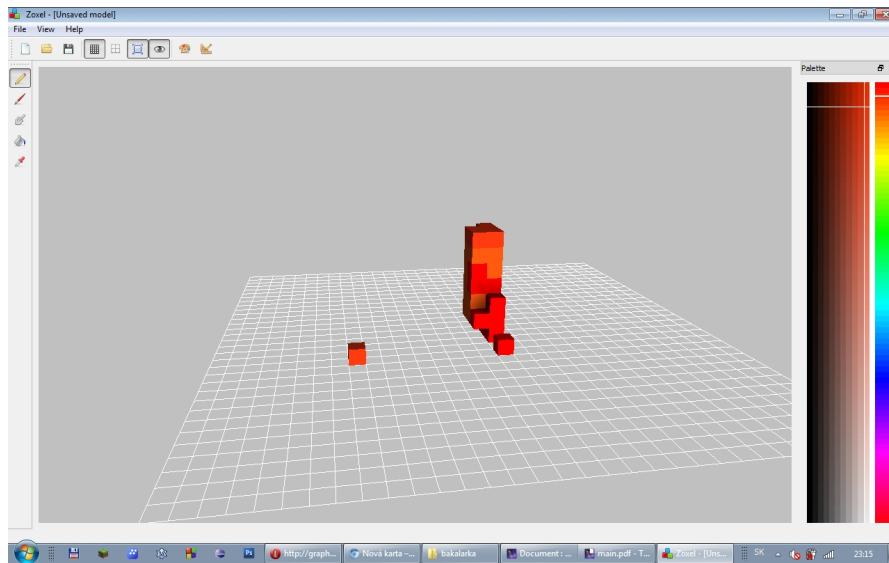
#### Výhody:

- **Lepší shading.** Väčšina programov používa pre tieňovanie jednoduchý flat-shading zatiaľ, čo Zoxel má implementovaný istý druh smooth-shadingu, resp. využíva základný z možností OpenGL knižnice.
- **Jednoduchý interface.** Interface je jednoduchý a intuitívny s možnosťou ľubovoľného premiestňovania niektorých komponentov, ako napríklad paleta farieb alebo nástrojov.
- **Wireframe.** Program umožňuje zobraziť model ako wireframe.
- **Interaktívne rozširovanie voxelového priestoru.**
- **Import/Export.** Výhodou programu je, že má vlastný binárny formát na ukladanie dát. Taktiež program ponúka aj export do .obj formátu a ukladanie a náčítanie súboru z programu Sproxel spomenutého nižšie.

#### Nevýhody:

- **Kamera.** Pri testovaní nám najväčšie problémy spôsobovala nemotorná kamera, ktorá občas prejavovala neočakávané správanie.
- **Kreslenie.** Kreslenie je vyriešené veľmi nešťastne, kedže sa dá pridať iba jeden voxel na klik. Teda nie je umožnené kreslenie na udalosť tahania myši.
- **Chudobná ponuka nástrojov.**

- **Jeden objekt v scéne.** Možnosť pridávania viacerých objektov do scény neumožňuje väčšina programov.



Obr. 3.4: Screenshot z programu Zoxel

### 3.2.5 Sproxel

Horlivý vyvíjaný voxelový editor, tak by sa dal charakterizovať Sproxel [VG12]. Interface aplikácie je zobrazený na obrázku 3.5.

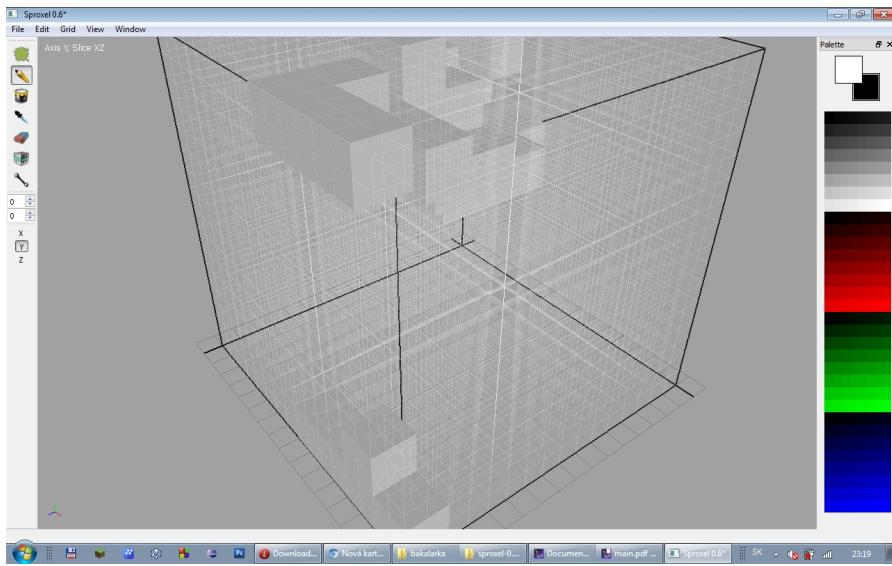
#### Výhody:

- **Jednoduché ovládanie.** Program mal jedno z najlepších ovládaní a nebolo vôbec zložité sa ho naučiť.
- **Jednoduchý interface.** Interface je jednoduchý a intuitívny s možnosťou ľubovoľného premiestňovania niektorých komponentov, ako napríklad paleta farieb alebo nástrojov.

- **Ponuka nástrojov.** Ponuka nástrojov a rôznych úprav je dostačujúca a v porovnaní s niektorými programami až nadpriemerná.
- **Export/import.** Okrem vlastného binárneho súborového formátu na ukladanie a načítavanie objektov má aplikácia v ponuke aj rôzne druhy importu a exportu napríklad ako .obj alebo ako obrázok vo formáte .png a mnohé iné.
- **Medziplatformovosť.**
- **Aktívny vývoj.** Aj v súčasnosti sa aktívne pracuje na vývoji tohto softvéru. Najnovšia verzia je Sproxel 0.6, takže nám vývojári pravdepodobne slúbjujú ešte niekoľko ďalších verzií.

#### Nevýhody:

- **Z-fighting.** Problém z-fightingu nie je riešený ani na miestach, kde by to bolo pomerne jednoduché. Jeho neriešenie spôsobuje nie len neestetickosť výsledku ale aj neprehľadnosť pri práci s voxelmi.
- **Voxel grid.** Čo by mohlo byť výhodou, je však nevýhodou, pretože zapnutie zobrazenia voxel gridu spôsobuje v podstate zaseknutie programu už pri počte voxelov  $32^3$ .
- **Jeden objekt v scéne.** Možnosť pridávania viacerých objektov do scény neumožňuje väčšina programov.



Obr. 3.5: Screenshot z programu Sproxel

### 3.2.6 Qubicle

Qubicle je veľmi kvalitný softvér od spoločnosti minddesk [Min13]. Interface aplikácie je zobrazený na obrázku 3.6.

#### Výhody:

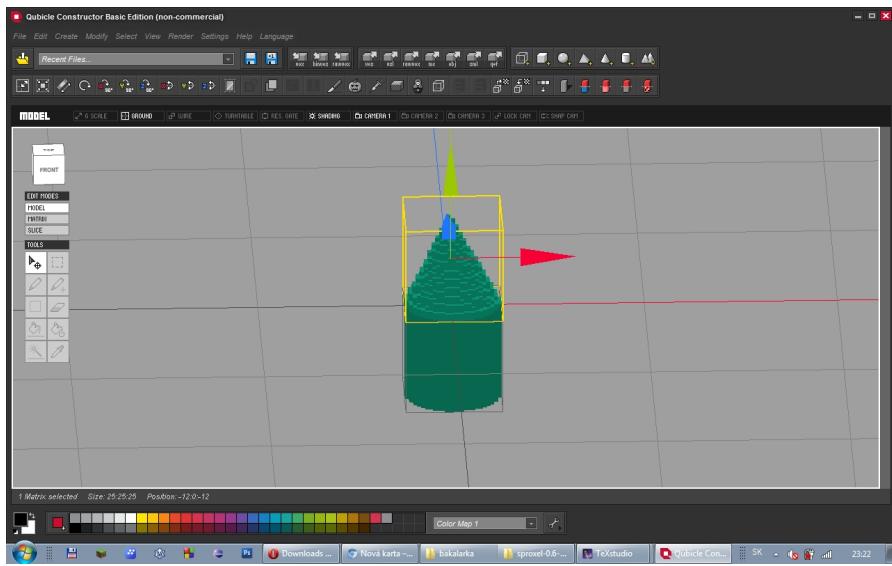
- **Ponuka nástrojov a funkcií.** Aplikácia sama o sebe neponúka veľké množstvo nástrojov, avšak ich počet je dostačujúci a sú doplnené ohromným množstvom funkcií a nastavení na úpravu objektov a scén.
- **Interface.** Prostredie je pomerne jednoduché s obzvlášť peknou grafikou.
- **Viacej objektov v scéne.** Tento program ako jediný z testovaných ponúka možnosť vytvárania viacerých nezávislých objektov v scéne, čo umožňuje lepšiu variabilnosť pri tvorbe.
- **Defaultné tvary.** Okrem prázdnego objektu je možné pridať základné ob-

jemové útvary ako napríklad kváder, ihlan, elipsoid, valec a kužel. Taktiež je možné vygenerovať náhodný voxelový terén. Táto možnosť je však až v platenej verzii, preto sme ju nemohli otestovať.

- **Editačné módy.** Rozličné módy editovania nám umožňujú tvoriť na troch rozličných leveloch. Na úrovni scény, na úrovni objektu a na úrovni rezu. Vďaka tomu máme pomerne veľkú kontrolu nad tým, čo robíme.
- **Import/Export.** Program má vlastný binárny formát na ukladanie a na načítavanie súborov. Taktiež je možný export do XML, .obj alebo do rôznych známych binárnych voxelových formátov.
- **Lokalizácia.** V nastaveniach programu je možné vybrať jeden zo štyroch svetových jazykov.
- **Rendering.** Ako jeden z mála programov, má *Quibicle* zabudovaný jednoduchý rendering objektov s obrázkovým súborom ako výstupom.

### Nevýhody:

- **Licencia.** Voľne stiahnuteľná je iba *Basic* verzia programu, ktorá sa dá použiť výlučne na nekomerčné účely. Táto verzia nezahŕňa širšiu funkcionality a export objektov. Vyššie verzie sú platené.
- **Komplikovanosť.** Veľké množstvo funkcií a nastavení vyžaduje viacej času naučiť sa dokonale ovládať tento nástroj. Avšak základy je veľmi ľahké pochopiť a sú dostačujúce na dosiahnutie pomerne dobrých výsledkov.



Obr. 3.6: Screenshot z programu Qubicle

### 3.3 2D grafické editory

Kedžže si predstavujeme prácu s voxelmi ako kreslenie objektov v priestore, inšpiráciou nám pre našu prácu bolo mnoho 2D kresliacich programov akými sú napríklad *Adobe Photoshop*, *Gimp* alebo jednoduchší *Microsoft Paint*. Tieto aplikácie poskytujú užívateľovi širokú sadu nástrojov, z ktorých sme sa rozhodli niektoré z nich použiť aj my. Taktiež sme sa snažili pri vytváraní GUI a nastavovaní spôsobu používania nástrojov vytvoriť podobný pocit, ako pri práci napríklad v programme MsPaint.

### 3.4 Zhrnutie poznatkov z testovania

Prieskum spomínaného softvéru nám umožnil vytvoriť si detajlnejší obraz o našich cieľoch a možnostiach pri implementácii práce. Najlepšie výsledky v testovaní dosiahol jednoznačne program Qubicle. Jeho filozofiou modelovania sme sa naj-

väčšmi stotožnili, kedže ako jediný umožňoval vytvárať samostatné objekty, ktoré bolo následne možné ľubovoľne umiestniť v scéne. Taktiež obsahoval najväčšie množstvo funkcií a aj napriek tomu mal rýchlo pochopiteľný interface. Jeho najslabšou stránkou teda zostáva, že jeho plná verzia je platená. Zvyšné programy nedosahovali podobné výsledky, avšak vytvorili spodnú hranicu kvality, ktorú by chcela dosiahnuť aj táto práca.

# Kapitola 4

## Implementácia

Začiatkom tejto kapitoly v stručnosti opisujeme použité technológie a taktiež uvádzame dôvody, prečo sme si ich zvolili. Jednotlivé časti programu sme zhrnuli do viacerých súvislých celkov, ktoré sme podrobne charakterizovali a preskúmali. Najdôkladnejšie sa venujeme časťam programu, ktoré považujeme za zaujímavé, z dôvodov ich zložitosti alebo kvôli metóde, ktorú sme pri ich riešení použili.

### 4.1 Použité technológie

#### 4.1.1 C++

Jazyk C++ vznikol rozšírením jazyka C o triedy a iné rysy objektovo orientovaného programovania. Jedná sa o jeden z najznámejších a najobľúbenejších programovacích jazykov za posledných pár dekád. Vďaka jeho obľúbenosti bolo vytvorených mnoho interných ale aj externých knižníc a nadstávb, čo je jeden z dôvodov, prečo sme si ho zvolili na prácu.

Vďaka tomu, že má tento jazyk nízkoúrovňové a zároveň aj vysokoúrovňové črty programovacích jazykov, umožňuje dobrý interface na efektívny manažment pamäťových a procesových zdrojov. Spomínaná vlastnosť jazyka je užitočná pri

pamäťovo a výpočtovo náročných algoritmov, s ktorými sa pri objemovej grafike zaručene stretнемe.

### 4.1.2 OpenGL

V práci sme si ako prvé uvedomili nevyhnutnosť vizualizácie voxelového priestoru a na túto úlohu sme vyбрали knižnicu OpenGL.

Jedná sa o jednu z najznámejších a najpoužívanejších open source multi-platformových API na interakciu s grafickými akcelerátormi (GPU). Dobre špecifikovaný OpenGL štandard má jazykové väzby na mnohé často používané programovacie jazyky [Gro13]. Funkcie OpenGL API sú dobre definované a sú volateľné aj z jazyka C++, ktorý sme si zvolili ako hlavnú implementačnú technológiu. Táto knižnica vo všeobecnosti ponúka široké množstvo výhod, akou je napríklad veľké množstvo podrobnych rád a návodov, ako s knižnicou narábať a zároveň má OpenGL veľmi podrobne spracovanú dokumentáciu. Aj toto bol jeden z dôvodov, prečo sme si ju zvolili pre nás účel.

### 4.1.3 DevIL

Počas implementácie sa ukázalo, že je potrebné načítať bitmapy, ktoré sa následne použili ako textúry, alebo sa pretransformovali na jednovrstvový voxelový objekt. Ako prvá sa nám ponúkla knižnica *DevIL* [WWD10], ktorú sme nakoniec aj v práci použili.

DevIL, pôvodne pomenovaná ako OpenIL, je programátorská knižnica, slúžiaca na načítavanie obrázkov širokej škály formátov do pamäti. Udržiava podobné menné konvencie ako OpenGL a veľmi dobre s ním aj spolupracuje.

#### **4.1.4 Microsoft Visual Studio 2012**

Spoločnosť *Microsoft* ponúka vysokokvalitné vývojové prostredie *Microsoft Visual Studio* [Mic13], ktoré je pre študentov našej fakulty dostupné zdarma a voľne použiteľné na nekomerčné účely. Jeho integrovanou súčasťou je aj prostredie na vytváranie GUI pre Win32, čo je jeho hlavná výhoda, ktorú sme zohľadňovali pri výbere toho správneho IDE. V prvých fázach práce sme volili verziu 2010 Express, ktorá však neobsahovala možnosť lepšieho syntax highlightu, čo bolo pri rozsiahlosti kódu pomerne dôležité, a tak sme nakoniec prešli na verziu 2012, ktorá nám plne vyhovovala.

#### **4.1.5 Assimp**

Na loadovanie povrchových modelov za účelom voxelizácie sme vybrali knižnicu *Assimp* [GSKN12], ktorá je taktiež voľne dostupná.

Jej hlavnými výhodami sú, že poskytuje pre a post-processing, má jednoduchý interface a podporuje tucty rôznych 3D súborových formátov.

Nevýhodami naopak sú, že má komplikovanejšiu inštaláciu, niektoré zdokumentované triedy nie sú správne implementované alebo nie su implementované vôbec a to často spôsobuje problémy, ktoré sa môžu objaviť pri práci.

#### **4.1.6 pugixml**

Na export a import dát z celej scény sme sa rozhodli použiť jazyk XML a nami navrhnutú podrobnú štruktúru opisujeme nižšie v tejto kapitole. Kedže parsovanie takýchto štruktúr nie je úplne triviálne a je vytvorených už mnoho existujúcich riešení, využili sme túto možnosť a siahli po knižnici *pugixml* [Kap12] na parsovanie XML súborov.

Dôvodom prečo sme si ju zvolili je, že sa jedná o extrémne rýchlu knižnicu, ktorá vytvorí DOM strom z XML súboru, takže zaručuje jednoduché použitie.

## 4.2 Rendering

Na rendering sme použili spomínanú knižnicu OpenGL s jej programovým rozšírením OpenGL Utility Toolkit (GLUT) [Gro96].

### 4.2.1 Rendering voxelu

Predpokladáme, že voxel ako základný element voxelového priestoru sa bude v scéne nachádzať v pomerne veľkých objemoch. Z tohto dôvodu bolo potrebné nájsť spôsob, ako jeden voxel čo najúspornejšie vyrenderovať. Na tento účel sme sa rozhodli použiť display listy. Display list je séria príkazov OpenGL, ktoré sú skompilované a uložené, spolu s inými dátami o objekte na GPU [Ahn08]. Vďaka tomu používanie displaylistov redukuje komunikáciu medzi GPU a CPU, ktorá je v procese renderovania časovo náročná. Display listy sú teda vhodné na rendering objektov s rovnakou topológiou, ktoré sa v scéne vyskytujú veľmi často. Keďže každý voxel je objekt v tvare kocky s rôznymi vlastnosťami akou je napríklad farba, môžeme použiť jedinú zabudovanú funkciu GLUTu:

```
void glutSolidCube(GLdouble size)
```

alebo tiež:

```
void glutWireCube(GLdouble size)
```

Spôsob renderovania voxelu ešte ovplyvňujú jeho vnútorné vlastnosti ako farba, flag určujúci označenie voxelu, a taktiež nastavenie renderingu ako napríklad shading, backface culling, zobrazenie wireframu alebo tiež veľkosť vzorky.

### **4.2.2 Rendering objektu**

Ako sme spomenuli o sekciu vyššie, bolo potrebné nájsť spôsob ako čo najviac ušetriť renderovací proces. Takéto vylepšenie je možné spraviť aj pri renderingu objektov, pretože veľké množstvo voxelov býva prekrytých inými. Použili sme preto veľmi jednoduchú, ale účinnú metódu, ktorá rozhoduje, či voxel zobrazí, podľa toho, či je ohraničený zo všetkých šiestich strán alebo nie je. Takto sme sa zbavili vykreslovania všetkých voxelov, ktoré nebolo možné nikdy vidieť.

### **4.2.3 Rendering scény**

Scéna sa skladá z viacerých voxelových objektov, ktoré je potrebné pri zobrazovaní posunúť správnu translačnou maticou na svoju pozíciu. Plávajúce objekty v priestore môžu na užívateľa programu pôsobiť dezorientujúco, a preto sme pridali do scény dva vizuálne elementy a tými sú podlaha a začiatok karteziánskej sústavy s troma osami.

## **4.3 Voxelové Objekty**

Sekundárny element v našom priestore po voxeloch je voxelový objekt, ktorý o sebe uchováva sadu základných informácií, z ktorých najdôležitejšie sú pozícia, dané rozmery a voxely, ktoré mu prislúchajú.

Každý objekt v scéne predstavuje samostatný ohraničený voxelový priestor. Na jeho reprezentáciu sme sa rozhodli použiť trojrozmerné pole, ktorého výhodou je priamy prístup k voxelom prostredníctvom ich indexov. Vďaka tejto vlastnosti je možné rýchlo a jednoducho procedurálne generovať voxelové objekty a ich rôzne transformácie.

### 4.3.1 Základné tvary

Pre jednoduchšie a rýchlejšie modelovanie väčšina 3D grafických nástrojov ponúka sadu základných telies, ktoré sú pri práci často využívané. Takýmto postupom sme sa inšporovali i my a zhovili sme sadu objektov<sup>1</sup>, ktoré sa generujú na základe jednoduchej matematiky. Takéto generovanie by sme mohli zovšeobecniť na nasledujúci algoritmus:

```
for ( $i, j, k = (0, 0, 0) \rightarrow (width, height, depth)$ ) do
    if  $v\_teleso(i, j, k)$  then // funkcia je špecifická pre každý útvar
        Pridaj voxel do Objektu na pozíciu (i,j,k)
    else
        Pridaj NULL do Objektu na pozíciu (i,j,k)
    end if
end for
```

### Kváder

Vytvorenie kvádra je veľmi jednoduché. Jeho podmienka v spomenutom algoritme je funkcia vracajúca vždy hodnotu *TRUE*. Takúto funkciu môžeme odstrániť a v jednoduchosti môžme povedať, že objekt inicializovaný na dané rozmeria iba naplníme voxelmi príslušnej farby.

### Elipsoid

Elipsoid je povrchové teleso dané rovnicou:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

---

<sup>1</sup>Kôli diskrétnosti voxelového priestoru, spomenuté objekty predstavujú iba aproximáciu konkrétnych geometrických útvarov.

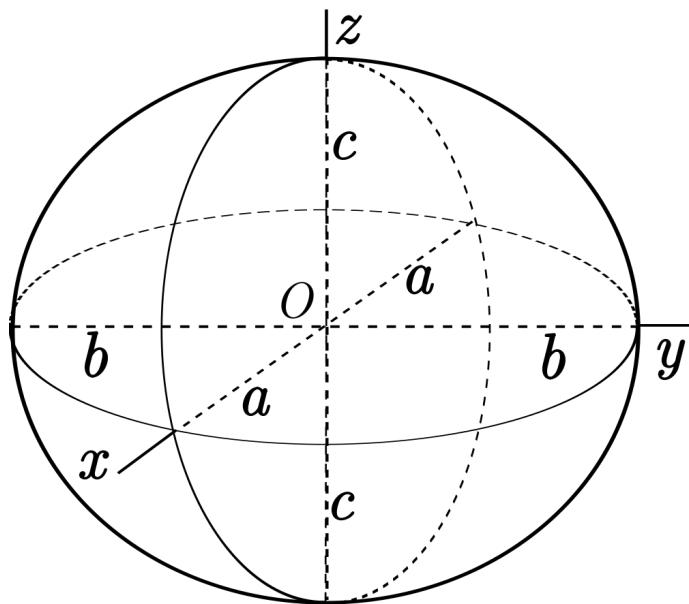
Kedže v našom prípade budeme potrebovať elipsoid so stredom v bode

$S = (s_1, s_2, s_3)$  k úžitku nám bude všeobecnejšia rovnica:

$$\frac{(x - s_1)^2}{a^2} + \frac{(y - s_2)^2}{b^2} + \frac{(z - s_3)^2}{c^2} = 1$$

Kde  $a, b, c$  určujú dĺžky poloosí v smere osí  $x, y, z$  ako možno vidieť na obrázku 4.1.

Vďaka tejto rovnici môžme ľahko určiť, ktoré body sa nachádzajú v telesu alebo mimo telesa, vytvorením nerovnice, ktorá vznikne nahradením znamienka = za  $<$  alebo  $>$ .



Obr. 4.1: Elipsoid

V našom prípade sú hodnoty  $a, b, c$  nahradené za  $width/2, height/2, depth/2$ , hodnoty  $x, y, z$  za  $i, j, k$  a stred  $S = (width/2, height/2, depth/2)$ . Naša funkcia rozhodujúca, či sa voxel nachádza v telesu by mohla vyzerať takto:

$$\frac{(i - (width/2))^2}{(width/2)^2} + \frac{(j - (height/2))^2}{(height/2)^2} + \frac{(k - (depth/2))^2}{(depth/2)^2} < 1$$

zjednodušene:

$$\frac{(i - s_1)^2}{s_1^2} + \frac{(j - s_2)^2}{s_2^2} + \frac{(k - s_3)^2}{s_3^2} < 1$$

a ešte jednoduchšie:

$$(\bar{p})^2 < 1, \text{ kde } \bar{p} = \left( \frac{i - s_1}{s_1}, \frac{j - s_2}{s_2}, \frac{k - s_3}{s_3} \right)$$

## Valec

Základ na vygenerovanie valca je podobný ako pre elipsoid, s tým rozdielom, že potrebujeme rovnicu posunutej elipsy do stredu  $S = (s_1, s_2)$ :

$$\frac{(x - s_1)^2}{a^2} + \frac{(y - s_2)^2}{b^2} = 1$$

Pre každú vrstvu objektu v našom prípade v smere osi  $y$  (zanedbávame teda  $y$ -ovú súradnicu respektíve súradnicu  $j$ ) zistujeme, či sa bod nachádza v elipse alebo mimo nej. Táto elipsa je určená stredom  $S = (width/2, depth/2)$  a rovnakými rozmermi, teda  $a = width/2, b = height/2$ . Výsledná nerovnica vyzerá nasledovne:

$$\frac{(i - (width/2))^2}{(width/2)^2} + \frac{(k - (depth/2))^2}{(depth/2)^2} < 1$$

Toto môžme zjednodušiť rovnako ako v predchádzajúcom prípade na:

$$(\bar{p})^2 < 1, \text{ kde } \bar{p} = \left( \frac{i - s_1}{s_1}, \frac{k - s_3}{s_3} \right)$$

## Ihlan

Pre pochopenie základného konceptu tvorby diskrétneho ihlanu, môžme problém zjednodušiť na vygenerovanie diskrétneho rovnoramenného trojuholníka. Počet bodov pre prvý a posledný riadok takéhoto trojuholníka vieme ľahko zistiť. Prvý riadok obsahuje 1 alebo 2 body podľa toho, či je šírka trojuholníka párna alebo nepárna, no a posledný riadok obsahuje počet bodov rovný šírke trojuholníka. Aby

sme získali počet bodov, ktoré sa nachádzajú v riadkoch medzi prvým a posledným, stačí nám použiť lineárnu interpoláciu bodov:  $X = (x, y)$ ,  $A = (x_0, y_0)$ ,  $B = (x_1, y_1)$

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

Pre nás je  $x$  hľadaný počet bodov na riadok  $j$ ,  $y$  je spomenutý riadok  $j$ ,  $A = (f, 1)$ , kde  $f$  je 1 alebo 2 a  $B = (width, height)$ . Z toho nám vyplýva:

$$x = \frac{(j - 1).(width - f)}{height - 1} + f$$

Teraz nám už stačí umiestniť body na stred dvojrozmernej mapy, čo je pomerne jednoduché. Túto myšlienku pre diskrétny trojuholník už ľahko rozšírime do priestoru.

## Kužeľ

Kužeľ je najkomplikovanejším tvarom, z pomedzi všetkých spomínaných. Avšak nie je potrebné vymýšlať nijaký nový spôsob na jeho vytvorenie, keďže tento tvar získame kombináciou už dvoch vyššie spomenutých metód. Vďaka metódam pre valec a pre ihlan môžme získať veľmi pekný voxelový kužeľ.

### 4.3.2 Booleanovské operácie nad objektami

Známa modelovacia paradigma CSG využíva na vytvorenie výsledného modelu booleanovské operácie. Potrebnosť tohto prvku vo voxelovom editore sme si uvedomili aj my, a tak sme sa rozhodli implementovať základné operácie akými sú *AND*, *OR*, *NOT* a *XOR*. Vo všeobecnosti by sa dal algoritmus, ktorý sme vytvorili, pre vstupné objekty  $A$ ,  $B$  a výstupný  $C$  opísť v troch krokoch:

1. Získanie bounding boxu pre  $C$ .
2. Zistenie pozície nového objektu  $C$ .

- Postupné vkladanie voxelov do objektu C z objektov A,B na základe danej logickej funkcie.

### 4.3.3 Transformácie objektov

#### Translácia

Jedná sa o základnú transformačnú funkciu objektov, ktorej úlohou je zmeniť polohu objektu v scéne. Vyznačuje sa jednoduchou implementáciou, kedže nie je potrebné nijakým spôsobom meniť štruktúru voxelov v objekte. V časti 4.4 *Nástroje* v podsekcii 4.4.7 *Posun objektov* je bližšie špecifikované ako môže užívateľ danú funkciu používať.

#### Škálovanie

Problém vzniku dier pri škálovaní sa z 2D rasterového priestoru prenáša aj do 3D voxelového priestoru. Na jeho riešenie sme použili algoritmus *Nearest-neighbor interpolation*, ktorý je veľmi jednoduchý, no napriek tomu dosahuje primerané výsledky.

Škálovať voxelový objekt je možné vo všetkých troch rozmeroch súčasne zadáním škálovacieho faktoru pre každý smer osi zvlášť.

#### Rotácia

Pri rotácii podobne ako pri škálovaní hrozí pri určitých hodnotách vznik dier. Tento problém sme riešili prepracovaním algoritmu na rotáciu bitmapy opísanom v článku od Michela Leunena [Leu03].

Základnou myšlienkovou je zistenie nových rozmerov obrázka a v našom prípade objektu pomocou danej rotačnej matice v priestore. Následne už len prechádzame všetkými bodmi nového objektu a snažíme sa získať voxely, pomocou spomenu-

tej matice, z pôvodného nezrotovaného objektu. Takýmto spôsobom sme dosiahli pomerne uspokojivé výsledky.

### Mapovanie textúr

Kôli jednoduchosti sme zvolili planárne mapovanie textúr na objekt podľa osí. Teda je možné namapovať textúru z celkovo 6 rôznych strán (2 pre každú os). Algoritmus mapovania textúry spočíva v štyroch krokoch:

1. Naškálovanie obrázka na potrebnú veľkosť. Toto zabezpečí knižnica DevIL.
2. Napasovanie obrázka na danú stenu objektu.
3. Vyslanie lúča z každého bodu obrázka smerom na objekt.  
(Takto získame potrebný voxel.)
4. Každý získaný voxel zafarbíme príslušnou farbou pixelu.

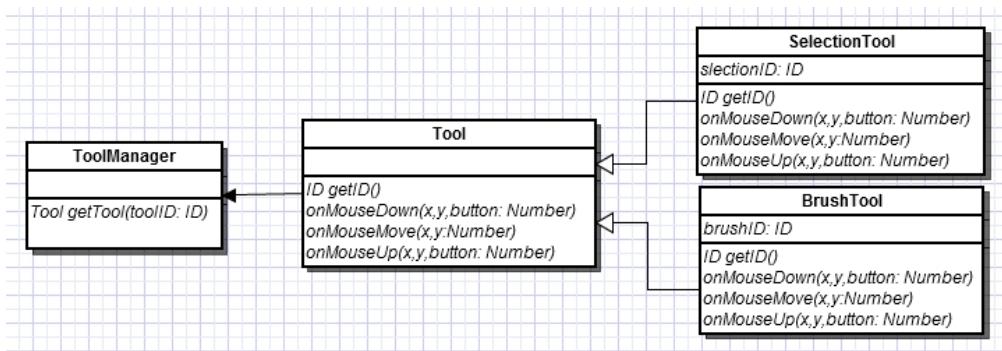
## 4.4 Nástroje

Akýkoľvek editor, ktorý si predstavíte, nemôže existovať bez editačných nástrojov. V tejto časti opíšeme, aké nástroje náš voxelový editor ponúka, ich stručný opis a v prípade zaujímavých nástrojov podrobnejší opis ich implementácie.

Správu nástrojov sme riešili podľa nasledujúceho UML diagramu 4.2.

Základom je abstraktná trieda Tool, ktorá obsahuje virtuálne metódy:

```
void mouseDown(int x, int y, int button)
void mouseMove(int x, int y)
void mouseUp(int x, int y, int button)
ID getID()
```



Obr. 4.2: Triedny diagram pre nástroje

Hlavnou spravujúcou triedou je **ToolManager**, ktorý uchováva všetky objekty odvodené od triedy **Tool**. Vďaka spôsobu akým je manažment nástrojov navrhnutý, môžeme rôzne typy nástrojov získať jednoducho z **ToolManagera** iba vďaka konkrétnemu ID.

#### 4.4.1 Selekcia

Tento nástroj má dve funkcionality. Prvou je selekcia objektov a druhou selekcia voxelov na udalosť kliknutia na daný objekt.

##### Selekcia objektov

Označovanie objektov prebieha tak, že sa z miesta obrazovky vyšle lúč a následne sa zistia priesečníky so všetkými objektami v scéne. Takto získame pole objektov, z ktorých vyberieme objekt najbližší ku začiatočnému bodu lúča. Kedže predpokladáme, že voxelových objektov nebude v scéne príliš veľa, táto metóda je pre nás dostačujúca.

## Selekcia voxelov

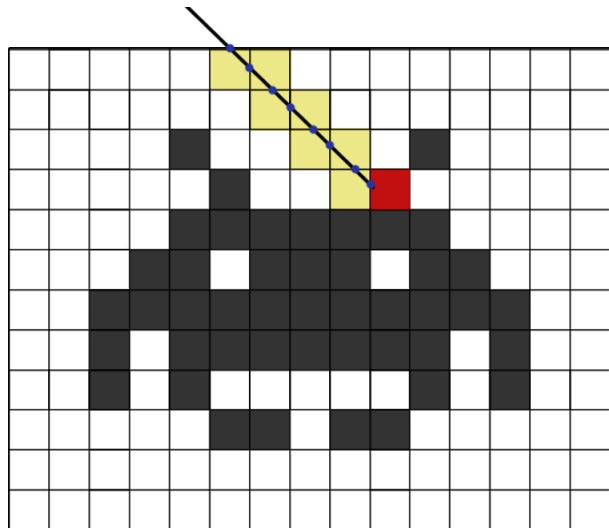
Od selekcie voxelov na kliknutie očakávame okamžitú spätnú väzbu. Ak by sme použili rovnakú metódu, ako na objekty, museli by sme zisťovať priesečníky s obrovským množstvom elementov, čo by spôsobovalo značné oneskorenie. Preto sme sa rozhodli využiť fakt, že každý voxelový objekt je umiestnený v pravidelnej trojrozmernej mriežke.

V prvej fáze zistujeme miesto vniknutia lúču do objektu. Na tomto mieste získame pozíciu bunky, kde by sa mohol nachádzať voxel. Ak sa na danej pozícii žiadnen voxel nenachádza, zistíme miesto, v ktorom lúč z bunky vychádza. Podľa tohto miesta zistíme susediacu bunku, do ktorej lúč vošiel. Ak sa ani v nej voxel nenachádza postupujeme ďalej v cykle, kým nenašramíme na nenullový voxel alebo lúč nevylezie z objektu von.

Opísaný algoritmus vyzerá asi takto:

```
function GETVOXEL(Luc, Objekt)
    Priesecnik  $\leftarrow$  Vypočítaj priesečník Luca a Objektu
    Pozicia  $\leftarrow$  Získaj pozíciu bunky v mieste Priesecnik
    while Pozicia sa nachádza v Objekte do
        Voxel  $\leftarrow$  Voxel na pozícii Pozicia v Objekte
        if Voxel  $\neq$  NULL then
            return Voxel
        end if
        Pozicia  $\leftarrow$  Získaj susednú pozíciu bunky, kam sa dostal Luc
    end while
end function
```

Na obrázku 4.3 je zobrazené fungovanie algoritmu.



Obr. 4.3: Vizualizácia fungovania algoritmu getVoxel

#### 4.4.2 Štetc

Taktiež aj tento nástroj poskytuje dve funkcionality. Tou prvou je pridávanie voxelov na povrch objektu a tou druhou je zafarbovanie zakliknutých voxelov. Základom tohto nástroja, ako aj mnohých nasledujúcich je opísaná funkcia na získavanie voxelov. Pri pridávaní voxelu sa získa kliknutý voxel a naň sa prilepí ďalší, zafarbený na natavenú farbu štetca. Pri zafarbovaní sa jednoducho zakliknutý voxel prefarbí podľa farby štetca.

#### 4.4.3 Guma

Tento nástroj využíva funkciu získavania voxelov na ich mazanie. Pri zmazaní voxelu sa lokálne upravia okolité voxely, tak aby boli viditeľné. Teda sa im nastaví flag *visible* na *TRUE*.

#### **4.4.4 Kvapátko**

Kvapátko je nástroj, ktorý už celé desaťročia mätie užívateľov svojím názvom. Slúži na získavanie farby z voxelu na kliknutie, preto bola aj pre tento nástroj potrebná funkcia na získavanie voxelu.

#### **4.4.5 Výplň**

Posledným nástrojom z rady implementovaných, ktorý používaja funkciu získavania voxelu pomocou lúča, je výplň. Tento nástroj zafarbí všetky voxely objektu, ktoré sú tranzitívne spojené so zakliknutým voxelom a zafarbí ich na farbu nastavenú v prostredí. To, že sú dva voxely spojené chápeme tak, že sú k sebe priamo susediace a zároveň sú rovnakej farby. Opísana procedúra priamo zachytáva algoritmus *flood fill*, ktorý je len potrebné preniesť do priestoru.

##### **3D flood fill**

S veľmi pochabými úmyslami sme sa pustili do implementácie algoritmu rovno dvoma spôsobmi.

Ako prvé sme zvolili rekurzívne riešenie a algoritmus prehľadávania do hĺbky. Funkciu bolo jednoduché implementovať, avšak už pri prvom testovaní sme narazili na veľký problém. Neuvedomili sme si, že pri veľkom počte voxelov sa funkcia vnára príliš hlboko do seba a teda nevyhnutne spôsobuje chybu pretečenia zásobníka

Nefungujúcemu funkciu bolo následne nutné prepísať tak, aby neobsahovala rekurziu. Tento problém sme vyriešili použitím radu a teda sme nahradili prehľadávanie do hĺbky, algoritmom prehľadávania do šírky. Naše riešenie teda vyzerá asi takto:

```

function FLOOD_FILL3D(indeces)
    Visited  $\leftarrow$  [ ]
    Queue  $\leftarrow$  [indeces]
    repeat
        position  $\leftarrow$  pop positon from Queue
        if  $\neg(\text{position in } \textit{Visited})$  then
            Change color of voxel at position
            for all position neighbours of same color do
                Queue  $\leftarrow$  Queue + [neighbour]
            end for
        end if
    until Queue is empty
end function

```

Finálne riešenie bolo ešte nutné pre urýchlenie upraviť tak, aby sme sa vyhli častému volaniu kopírovacích konštruktorov, použitím smerníkov.

#### 4.4.6 Kamera

Tento nástroj je veľmi dôležitý pre užívateľa, aby sa rozumným spôsobom mohol orientovať v scéne. Ohnisko kamery je defaultne nastavené na stred celej scény, teda na bod  $(0, 0, 0)$ . Pri selekcii objektu pravým tlačidlom myši sa však ohnisko kamery zmení na stred označeného objektu. Kamera nám umožňuje orientáciu v scéne dvoma spôsobmi. Prvým je rotácia kamery okolo ohniska a druhým je približovanie sa alebo oddaľovanie od objektu, inak povedané zoom.

#### **4.4.7 Posun objektov**

Označené objekty je možné posúvať na dragovaciu udalosť v troch rovinách.

Kliknutie na objekt určí rovinu, v ktorej je možné objekt pohybovať podľa toho, na ktorú stenu užívateľ klikol. Okolo objektu sa následne zobrazí obdĺžnik, ktorý túto rovinu vizualizuje ako pomôcku pre užívateľa. Takéto riešenie je asi jediné svojho druhu, no napriek tomu ho považujeme za celkom jednoduché a intuitívne.

### **4.5 Import/Export**

Modelovacie aplikácie akou je aj tá naša si vyžadujú import a export dát do a z programu. Rozhodli sme nájsť špecifikácie rôznych formátov voxelových súborov, ktoré už existujú, aby sme dosiahli čo najlepšiu variabilitu programu pri použití v praxi. Okrem ukladania dát do binárnych voxelových súborov je možné scénu exportovať do XML štruktúry alebo aj ako povrchový model vygenerovaný z voxelového.

#### **4.5.1 Polygónový model**

##### **Import**

Kedže všetky modely, ktoré sa nachádzajú v programe je možné uchovávať a zobrazovať iba v podobe voxelových objektov, import klasického povrchového modelu predstavuje nutnosť konverzie 3D meshu na voxely. Tento proces je tiež nazývaný *voxelizácia*.

## Voxelizácia

Vo všeobecnosti proces voxelizácie produkuje množinu hodnôt v pravidelnej troj-rozmernej mriežke z objektov určených povrchovou reprezentáciou. Cieľom voxelizácie je vytvoriť voxelový objekt, ktorý je čo najbližšou možnou aproximáciou k originálnemu objektu. [PTTK06] Presnosť výsledku samozrejme závisí aj od rozmerov mriežky, ktoré si zvolíme. Kritickými miestami tohto procesu môžu byť objekty s nulovou hrúbkou, alebo objekty s nekonvexnými dierami.

Riešenia opísané Georgom Passalisom [PTTK06] využívajú možnosti GPU prostredníctvom funkcií OpenGL. Základná myšlienka spočíva v použití z-bufferov, vyprodukovaných povrchovými objektami, na voxelizáciu. Algoritmus, ktorý prezentuje Karabassi potrebuje 3 páry z-bufferov, kde každá dvojica je určená pre jednu os. Kamera sa umiestni na všetkých 6 plôch pomyselného bounding boxu voxelizovaného objektu a s použitím ortogonálnej projekcie sa zapíšu dáta do bufferov. [EKT99]

Toto riešenie je pomerne jednoduché, avšak v istých prípadoch produkuje neželané výsledky. Ďalšia implementácia voxelizácie spomenutá v tomto článku rieši nedostatky tohto algoritmu použitím komplikovaných shaderov, a preto sme sa rozhodli túto metódu nepoužiť a zvolili sme priamočiarejší prístup k problému.

Naše riešenie napasuje voxelizovaný objekt do mriežky s danými rozmermi a pre každú bunku mriežky určí, či sa jej stred nachádza v objekte alebo nie. Ak sa daný bod v objekte nachádza, pridáme voxel do objektu, ak sa nenachádza, tak nepridáme nič, respektíve pridáme nullovú hodnotu.

Približne takto vyzerá náš program:

- Vytvor trojrozmernú mriežku  $M$  s rozmermi ( $width, height, depth$ )
- Naškáluj objekt  $O$  do danej mriežky (tak aby sa objekt nedeformoval)

**for all**  $bunka \in M$  **do**

**if**  $bunka$  sa nachádza v  $O$  **then**

- Pridaj voxel do objektu.

**else**

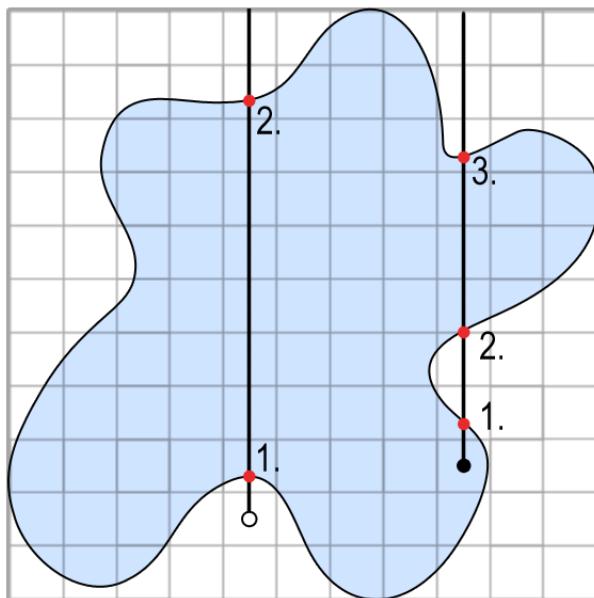
- Pridaj  $NULL$  do objektu.

**end if**

**end for**

Takýto prístup je intuitívny a ľahko pochopiteľný aj pre laika. Rozhodujúcu úlohu v algoritme zohráva metóda zobrazená na obrázku 4.4, ktorá determinuje, či sa bod nachádza v objekte alebo nie. Túto úlohu sme vyriešili tak, že sa z daného bodu vyšle lúč (polpriamka), a spočítajú sa priesčníky lúča s povrhom objektu. Ak ich počet je nepárny znamená to, že bod sa v objekte nachádza, inak sa v objekte nenachádza. Takto sa nám úloha posunula na ďalší podproblém zisťovania priesčníku polpriamky s polygónom. Kedže nájsť priesčník priamky s trojuholníkom je ľahšie ako s ľubovoľným polygónom, triangulizovali sme objekt už pri načítavaní zo súboru a potom sme mohli nájsť priesčník polpriamky s rovinou trojhlníka a následne pomocou barycentrických súradníc sme rozhodli, či sa v ňom daný priesčník nachádza.

Na obrázku 4.4 vidno fungovanie algoritmu na zistenie, či sa bod nachádza v objekte. Môžete vidieť dva príklady, kedy je lúč vyslaný z daného bodu na hor. V prvom prípade je nájdený párný počet priesčníkov, teda sa bod nenachádza v objekte a v druhom prípade ich je nájdených nepárny počet, teda sa bod v objekte nachádza.



Obr. 4.4: Vizualizácia algoritmu zisťovania, či sa bod náchádza v objekte

## Export

Z každého voxelového objektu je možné extrahovať jeho povrchovú informáciu a vytvoriť z nej klasický 3D model. Táto vlastnosť je užitočná napríklad pri postprocessingu, ak by sme napríklad chceli vytvoriť kvalitný render voxelových objektov. Nás algoritmus prechádza všetky voxely v objekte, vypočíta pozíciu všetkých krajných 8 bodov voxelu a pre každý určí, či sa nachádza na povrchu objektu alebo nie. Následne tieto body pridá do poľa tak, aby sa v ňom nenachádzali duplicitne a do ďalšieho poľa si zapamätá poradie bodov, aby bolo možné danú viditeľnú stenu rekonštruovať.

### 4.5.2 Bitmapa

Do programu sme zahrnuli aj možnosť importovania bitmapy do prostredia. Spôsob, akým to robíme, je pomerne jednoduchý. Načítame si bitmapu zo súboru do pamäte a následne si vytvoríme voxelový objekt s hĺbkou 1 a šírkou a výškou rovnajúcimi sa rozmerom obrázka. Teraz nám stačí už len prejsť všetky body bitmapy a pridať voxel príslušnej farby do objektu.

### 4.5.3 Voxelové formáty

Špecifikáciu bežne používaných voxelových formátov sme získali zo zdrojových kódov opensource programu *binvox* [HZM12]. Vďaka tomu sa nám podarilo pridať do ponuky aplikácie pomerne veľké množstvo rôznych výstupných a vstupných formátov.

Výstupné formáty:

- Binvox
- RawVox
- Hips
- Mira
- Vt
- Vtk

Vstupné formáty:

- Binvox
- Vox
- Mira
- Vt
- Vtk

Tieto formáty sa štruktúrou na seba dosť podobajú. Zaujímavým z nich je napríklad Binvox, ktorý používa RLE kódovanie na kompresiu dát a najjednoduchším z nich je RawVox, ktorý v sebe uchováva surové dáta tak, že na mieste voxelu ukladá 1 a na mieste prázdnego miesta 0. Ich veľkou nevýhodou je, že nie je

možné v nich ukladať viacej objektov naraz a taktiež neuchovávajú informáciu o farbe voxelov. Aby sme mohli ukladať celé scény pred začatím zapisovania dát do súboru, zlúčime všetky objekty v scéne a následne takto vzniknutý objekt uložíme.

#### 4.5.4 XML

Kedže všetky známe formáty na ukladanie voxelov do súborov sú binárne a neumožňujú ukladanie farieb a ani celých scén s väčším množstvom objektov. Rozhodli sme sa ukladať scény do nami navrhнутej XML štruktúry. Takýto výstup je nielen viacej čitateľný, ale môžeme si dovoliť uložiť akékoľvek informácie navyše, ktoré potrebujeme. Jeho nevýhodou oproti binárnym súborom je však jeho veľkosť. Import aj export XML sme implementovali prostredníctvom vyššie spomenutej knižnice *pugixml*.

Takto vyzerá nami navrhnutá XML štruktúra:

```
<?xml version="1.0" encoding="utf-8"?>
<cubo>
    <scene width="" height="" depth="">
        <positions>
            <position index="" x="" y="" z="" />
            ...
        </positions>
        <objects>
            <object width="" height="" depth="" position="">
                <colors>
                    <color index="" red="" green="" blue="" alpha="" />
                    ...
                </colors>
                <voxels sample="">
                    <voxel i="" j="" k="" color="" />
                    ...
                </voxels>
            </object>
            ...
        </objects>
    </scene>
</cubo>
```

# Kapitola 5

## Výsledky

V tejto kapitole uvádzame merania výkonu aplikácie a taktiež niektoré vizuálne výsledky, ktoré sme získali na výstupe z aplikácie.

### 5.1 Metodika merania

Merania sme vykonávali na verzii programu skompilovanej v *release* móde, od čoho sme si sľubovali lepšie výsledky. Merania FPS sme vykonali pomocou programu *Fraps* [Pty13] a na merania času sme použili zabudovanú funkciu *clock()*, ktorá meria čas v milisekundách. Na odtestovanie voxelových súborov sme použili program *viewvox* [Min12], ktorý je spustiteľný cez konzolu, avšak nedokáže otvoriť všetky implementované súborové formáty.

Testy prebiehali pod 32-bitovým operačným systémom Microsoft Windows 7 Ultimate. Testovací hardvér bol AMD Turion X2 Dual-Core Mobile RM-72 2.1 GHz, 3GB DDR2 pamäte, zabudovaná grafická karta ATI Mobility Radeon HD 3400 Series 512MB. Na hardvéri boli nainštalované grafické ovládače Catalyst 8.97.100.7 16-XI-12 podporujúce OpenGL 3.3.

## 5.2 Rendering

Rýchlosť renderovania je závislá od počtu voxelov v scéne a tvaru samotného objektu. Ucelené tvary dosahujú lepšie výsledky, ako tvary s veľkým množstvom dier vo vnútri objektu, keďže program je nastavený tak, že skrýva voxely obkolesené zo všetkých strán. V tabuľke 5.1 je zobrazený počet voxelov pre rôzne objekty a rôzne rozmery. Na objektoch s týmito rozmermi sme následne vykonali testy FPS. Tabuľka 5.2 zobrazuje výsledky testovania pre dané objekty.

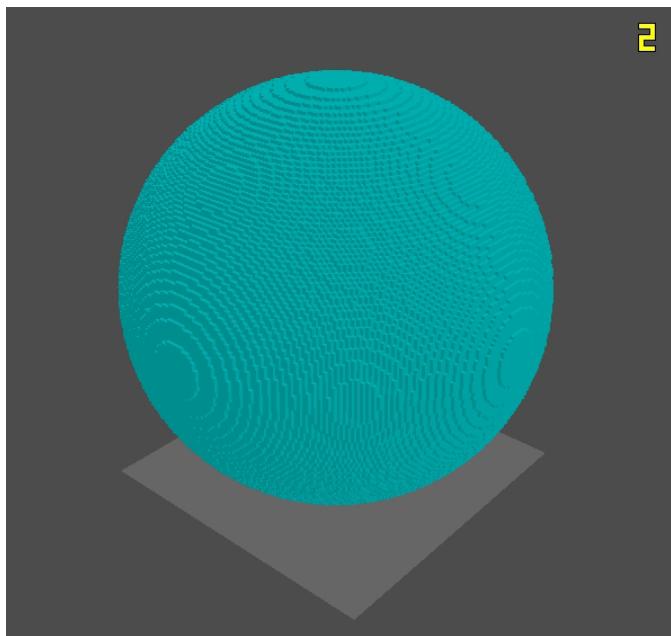
Tvar Rozmery	$16^3$	$32^3$	$64^3$	$128^3$
Kocka	4096	32 768	262 144	2 097 152
Guľa	2176	17 256	137 376	1 099 136
Kužeľ	1108	8680	68 672	549 760
Náhodne generovaný objekt	2058	16 490	131 072	—

Tabuľka 5.1: Tabuľka počtov voxelov na objekt

Tvar Rozmery	$16^3$	$32^3$	$64^3$	$128^3$
Kocka	62 FPS	19 FPS	5 FPS	1 FPS
Guľa	105 FPS	35 FPS	9 FPS	2 FPS
Kužeľ	110 FPS	40 FPS	11 FPS	3 FPS
Náhodne generovaný objekt	48 FPS	7 FPS	1 FPS	—

Tabuľka 5.2: Rýchlosťi renderovania objektov namerané v FPS

Na obrázku 5.1 možno vidieť vyrenderovanú guľu o rozmeroch  $128^3$  voxelov a počet framov za sekundu nameraných programom Fraps.



Obr. 5.1: Vyrenderovaná guľa zložená z 1 099 136 voxelov

### 5.3 Voxelizácia

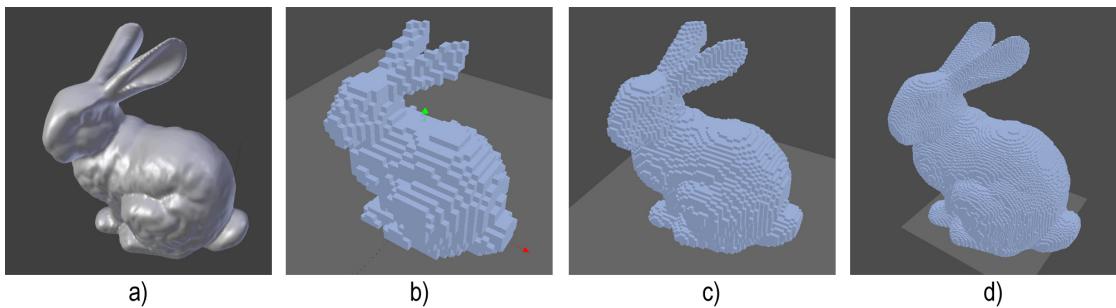
Vybrali sme niekoľko známych povrchových modelov s rôznym počtom polygónov respektíve trojuholníkov, ktoré sme voxelizovali do mriežky rôznych rozmerov. Počas voxelizácie sme testovali čas, ktorý uplynie od začiatku funkcie voxelizácie po jej koniec. Nezapomňali sme teda časový úsek, ktorý ubehne počas načítania polygónových modelov do pamäte.

Taktiež sme vizuálne kontrolovali tvar výsledného voxelového objektu. V tabuľke 5.3 sú zobrazené namerané údaje pre konkrétny model a rozmery. Celkovo voxelizácia dosahovala uspokojivé výsledky.

Na obrázku 5.2 je vidno výsledky voxelizácie pre model *Standford bunny* s 69666 trojuholníkmi v rozmeroch  $32^3, 64^3, 128^3$  voxelov.

Objekt(počet trojuholníkov)	Rozmer	16	32	64	128
Suzanne (968)		0.022 s	0.125 s	0.92 s	6.88 s
Teapot (6321)		0.078 s	0.421 s	2.418 s	16.24 s
Pumpkin (10 000)		0.123 s	0.577 s	3.074 s	18.486 s
Standford bunny (69 666)		1.17 s	5.678 s	30.467 s	184.76 s

Tabuľka 5.3: Rýchlosť voxelizácie rôznych modelov

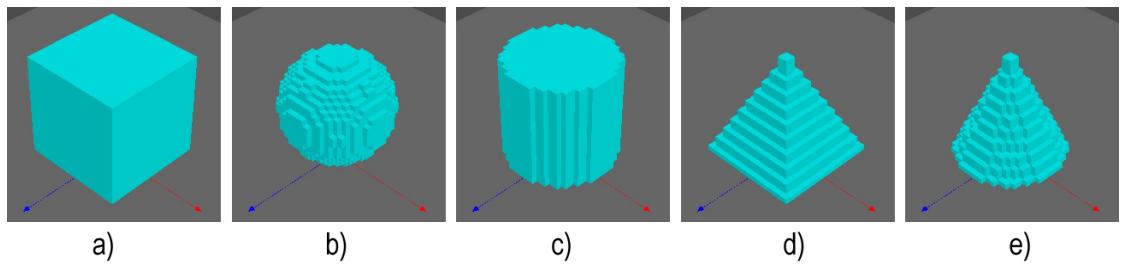


Obr. 5.2: Výsledky voxelizácie. a) Originálny polygónový objekt b)  $32^3$  c)  $64^3$  d)  $128^3$

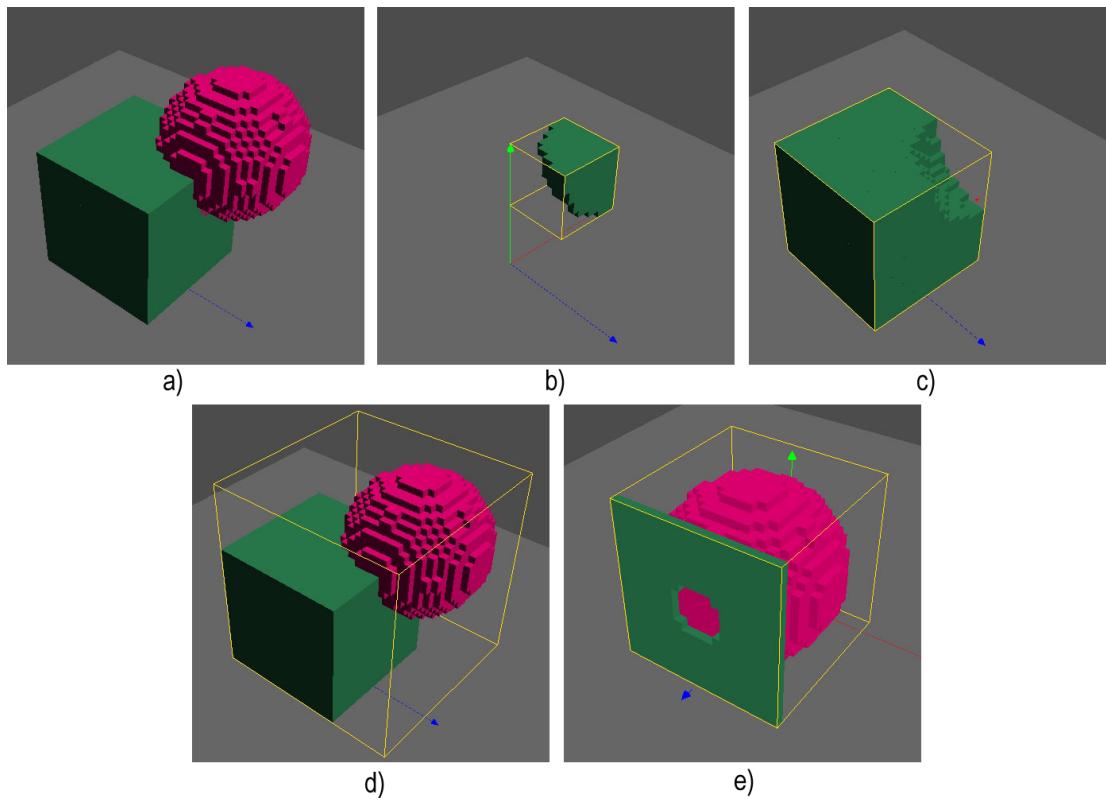
## 5.4 Voxelové objekty

Ako bolo spomenuté v kapitole 4, implementovali sme 5 základných voxelových tvarov: kváder, elipsoid, valec, ihlan a kužeľ. Výsledky implementácie môžete vidieť na obrázku 5.3. Na tomto obrázku vidno objekty v rozmeroch  $20 \times 20 \times 20$  voxelov, avšak je možné ich vytvoriť v ľubovoľných rozmeroch od 1 po 50. Aspoň takéto obmedzenia ponúka nami vytvorený interface na tvorbu voxelových tvarov.

Taktiež sme nad objektami implementovali boolovské operácie, ktoré v programe dosahovali očakávané výsledky ako je možné vidieť na obrázku 5.4.



Obr. 5.3: Základné tvary. a) kocka(kváder) b) guľa(elipsoid) c) valec d) ihlan e) kužeľ



Obr. 5.4: Výsledok booleanovských operácií. a) Originálne objekty b) AND c) NOT d) OR e) XOR (nevznikol z objektov na obrazku a) )

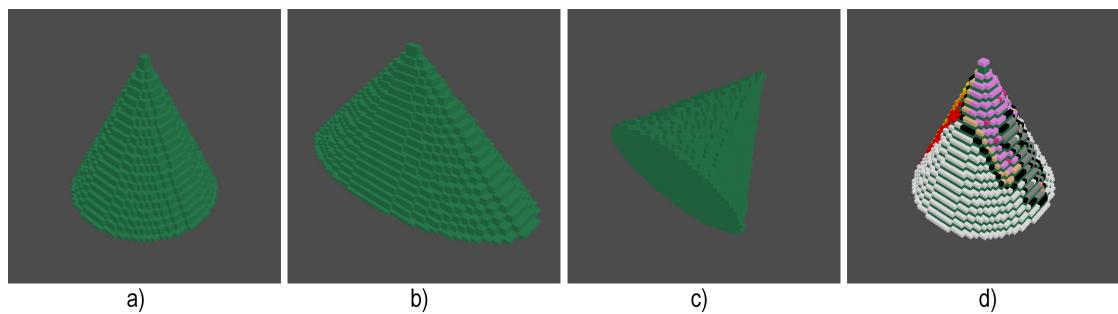
## 5.5 Nástroje

Získavanie voxelov z objektu na kliknutie je dostatočne rýchle a teda základné nástroje, ktoré využívajú túto funkciu, ako napríklad *štetec*, *guma*, *kvapkátko* a

podobne, dosahujú želané výsledky a nestretli sme sa so žiadnymi závažnejšími problémami pri ich navrhovaní a ani používaní.

Zaujímavejším z nástrojov je napríklad nástroj *výplň*, ktorého fungovanie je zložitejšie a časovo náročnejšie, keďže využíva algoritmus prehľadávania do šírky a teda v mnohých prípadoch prechádza cez všetky voxely objektu. Tento nástroj sme po prvotnom testovaní museli optimalizovať s použitím smerníkov, a tak sme dosiahli niekoľkonásobne väčšiu rýchlosť, keďže sme sa vyhli zbytočnému volaniu kopírovacích konštruktorov.

Taktiež problémovými boli modifikačné nástroje s výnimkou translácie. V prípade škálovania a rotácie bolo potrebné vyriešiť vznikajúce diery, čo sa nám podarilo na základe spomenutých algoritmov v kapitole 4 *Implementácia* v sekcii 4.3 *Voxelové objekty*. Výsledky našej implementácie týchto postupov môžete vidieť na obrázku 5.5.



Obr. 5.5: Transformácie obejktov a textúrovanie. a) Pôvodný objekt b) Naškálovaný objekt na dvojnásobnú šírku c) Zrotovaný objekt o  $45^\circ$  okolo osi  $x$  d) Objekt s namapovanou textúrou

## 5.6 Import/Export

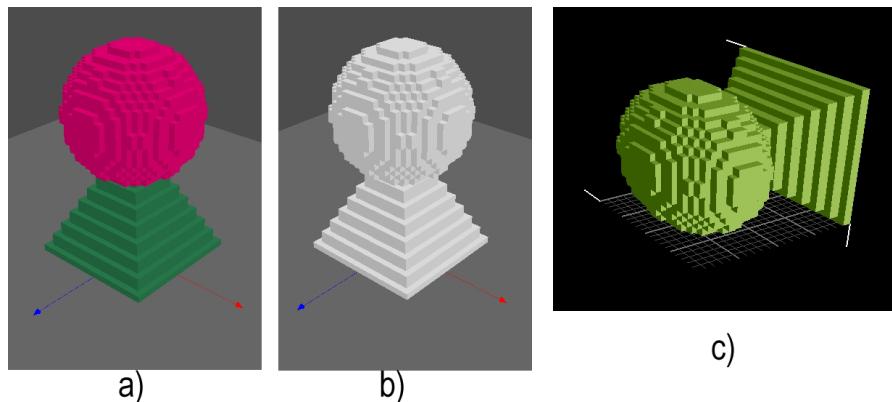
Testovanie importu a exportu voxelových súborov sme realizovali prostredníctvom programu na zobrazovanie voxelových objektov *viewvox* a programu na voxelizáciu

3D modelov *binvox* [HZM12]. Oba programy nepodporujú všetky nami implementované súborové formáty, preto bolo môžné vyskúšať ukladanie scény do formátov: .binvox, .mira a loadovanie súborov vygenerovaných programom binvox: .binvox, .mira, .vtk.

Ako možno vidieť na obrázku 5.6, program viewvox pri zobrazovaní naškáluje objekt na rozmery  $1 \times 1 \times 1$ , čo spôsobuje deformáciu objektov, ktoré nemajú totožné všetky tri dimenzie. Ďalším rozdielom pri zobrazovaní je umiestnenie osí  $z$  a  $y$ , ktoré sú v našom prípade vymenené.

Pomocou programu binvox sme voxelizovali trojrozmerný polygónový objekt na voxelový o rozmeroch  $64^3$  a uložili vo vyššie spomenutých formátoch. Takéto súbory sme sa potom snažili načítať do nášho prostredia. Prvý pokus nebol úspešný, ale po malých úpravách kódu sa nám súbory podarilo korektne spracovať a voxelové objekty načítať s vymenenou osou  $z$  podobne ako pri programe viewvox. Ako bolo spomenuté v kapitole 4 v sekciu 4.5 a ako možno vidieť na obrázku 5.6 tieto formáty neumožňujú ukladanie farieb, a tak sa zobrazujú v defaultnej bielej farbe.

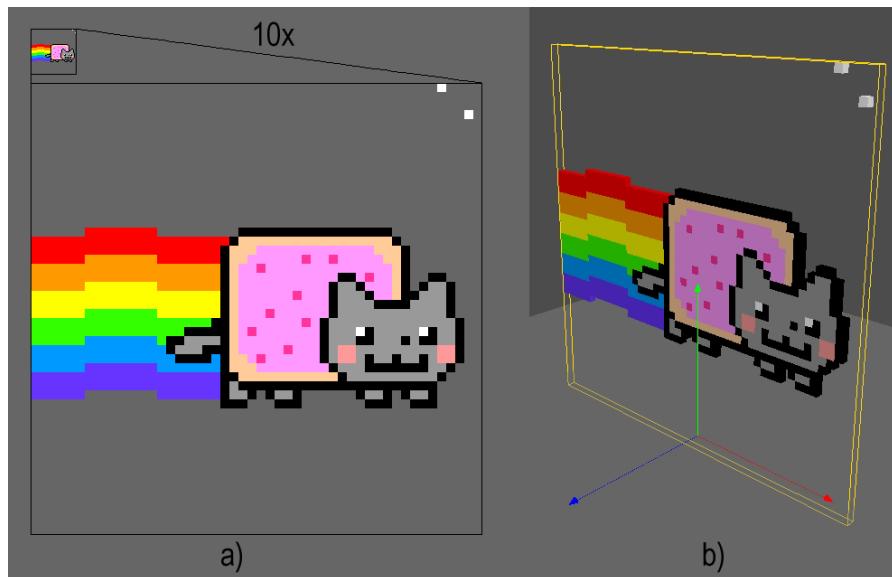
Uskutočnili sme aj testy exportu a importu pre XML štruktúru. Výsledky boli očakávané a ničím zaujímavé, a preto ich tu ani neuvádzame.



Obr. 5.6: Výsledky testovania exportu do binárnych súborov. a) Exportovaný objekt b) Načítaný objekt v našom prostredí c) Načítaný objekt v programe viewvox

Okrem importu meshu, teda voxelizácie, ktorej výsledky sme vyhodnotili výšie, je možné do programu importovať rôzne obrázkové súbory.

Import bitmapy do prostredia bol taktiež zaujímavý. Podarilo sa nám vložiť akýkoľvek obrázkový formát, ktorý je schopná otvoriť knižnica DevIL, s výnimkou formátov, ktoré majú indexované farby ako napríklad gif. Na obrázku 5.7 možno vidieť bitmapu mačky vo formáte PNG s transparentnými pixelmi (na obrázku ich vidno ako šedé) o rozmeroch  $50 \times 50$  pixelov a objekt, ktorý vznikol importovaním tohto obrázka do programu. Program zachováva transparenciu pixelov, ktoré majú hodnotu *alpha* nulovú.



Obr. 5.7: Výsledok importu bitmapy. a) 10 násobne zväčšená bitmapa mačky b) importovaná bitmapa do prostredia

## 5.7 Užívateľské prostredie

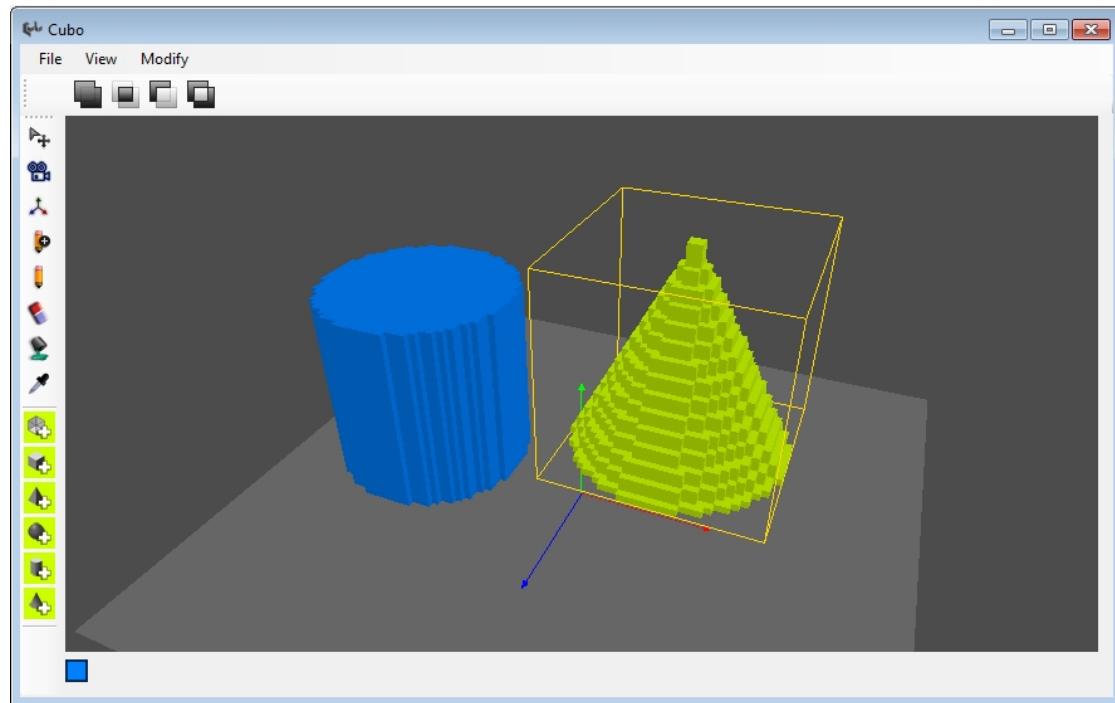
Pri tvorbe užívateľského prostredia sme kládli dôraz na vytvorenie čo najväčšieho pracovného priestoru. Výsledkom je, že najrozmernejšia časť okna aplikácie zabera

takzvané plátno, na ktoré je premietaná voxelová scéna.

V ľavom paneli sa nachádzajú dve súriny ikoniek predelené oddelovačom, z ktorých prvá sú základné editačné nástroje a druhá sú základné tvary, ktoré možno do prostredia vložiť.

Ďalej sa tu nachádzajú ešte dve horizontálne lišty zarovnané na vrchný okraj okna. V jednej sa nachádzajú štyri ikonky, ktoré slúžia na booleanovské operácie nad objektami a druhá lišta obsahuje zvyšnú funkcionality programu, teda import a export, možnosť prepínania zobrazenia podlahy, osí, tieňovania a podobne a taktiež možnosť transformácie označených objektov.

Na obrázku 5.8 možno vidieť výsledné prostredie nášho editora, tak ako sme ho opísali v tejto sekcií.



Obr. 5.8: Screenshot z programu *Cubo*

# Záver

V práci sme v stručnosti opísali základné poznatky z objemovej grafiky a dôležitosť nástroja na manipuláciu s voxelmi. Zadefinovali sme pojem *voxel* a preskúmali sme výhody a nevýhody tohto elementu i v komerčnej sfére.

Uviedli sme malú vzorku existujúcich voxelových editorov a opísali sme výhody a nevýhody, ktorými sme sa inšpirovali, jednotlivo pre každý program. Nadobudnuté poznatky sme ďalej aplikovali pri implementácii samotného voxelového editora.

Výsledkom práce je teda plne funkčný program s možnosťou vytvoriť takmer ľubovoľný voxelový objekt alebo scénu s obmedzením na rozmery. Vytvorená aplikácia ponúka uspokojivé množstvo modelovacích respektíve kresliacich nástrojov a taktiež možnosť importovať a exportovať scénu do XML, WaveFront(.obj) alebo aj rôznych binárnych voxelových formátov. V práci sme tiež opísali a implementovali algoritmus voxelizácie uzavretých polygónových objektov slúžiaci na import modelov do scény.

## Budúca práca

Prakticky je možné prerobiť akúkoľvek funkciu nad rastrami, aby pracovala nad voxelmi, a presne tam vidíme ďalšiu možnosť vývoja aplikácie. Vo všeobecnosti je ďalší postup obmedzený iba našou fantáziou a časom, a preto spomenieme iba pár majoritných vylepšení.

V prvom rade by bolo dôležité umožniť do prostredia vkladať a zobrazovať väčšie objemy dát. V momentálnom stave môžu mať objekty, bez vplyvu na rýchlosť, rozmery v ráde desiatok. Takýto stav je však nedostačujúci ale pri voxelových editoroch celkom bežný. Budúca snaha by bola optimalizovať uchovávanie dát a

rendering, aby bolo možné vytvárať objekty v ráde stoviek až tisícok ako to je pri bežných rastrových nástrojoch.

Kedže sme implementovali iba jednoduché planárne mapovanie textúry na objekt, bolo by zaujímavé vytvoriť zložitejšie spôsoby textúrovania objektov s rôznymi nastaveniami.

K lepšiemu ovládaniu nástroja a obzvlášť k lepšiemu manipulovaniu s obsahom objektov by dopomohla funkcia kreslenia po vrstvách. Takto by bolo možné sa postupne posúvať po vrstvách objektu a vyplňovať bunky príslušnou farbou, čím by sa značne uľahčila práca.

Ďalším dôležitým chýbajúcim prvkom v aplikácii je možnosť vyrenderovania objektu do obrázka. Toto by si vyžadovalo komplikovanejší prístup a použitie renderovacích algoritmov v diskrétnom priestore, ako je napríklad diskrétny raytracing, ktorý opísal Ronie Yagel [YKC92]. Okrem statického renderovania, by bolo veľmi zaujímavé vyrenderovať voxelovú animáciu, čo by vyžadovalo aj príslušné nástroje na správu framov, časovania, pohybu voxelov a podobne.

# Literatúra

- [Ahn08] Song Ho Ahn. [www.songho.ca/opengl/gl\\_displaylist.html](http://www.songho.ca/opengl/gl_displaylist.html). 2008.
- [BSc02] Andrew Steven Winter BSc. *Field-Based Modelling and Rendering*. PhD thesis, University of Wales, December 2002.
- [Cor10] Atomic Corporation. Paint3d <http://www.paint3d.net/>. 2010.
- [EHS06] Klaus Engel, Markus Hadwiger, and Christof Rezk Salama. Real-time volume graphics. 2006.
- [EKT99] G. Papaioannou E.A. Karabassi and T. Theoharis. A fast depth-buffer-based voxelization algorithm. 4:5–10, 1999.
- [Eve08] Everygraph. Voxel3d <http://www.everygraph.com/voxel3d/>. November 2008.
- [Gro96] The Khronos Group. Glut <http://www.opengl.org/resources/libraries/glut/>. 1996.
- [Gro13] The Khronos Group. [www.opengl.org/about/](http://www.opengl.org/about/). 2013.
- [GSKN12] Alexander Gessler, Thomas Schulze, Kim Kulling, and David Nadlinger. Assimp <http://assimp.sourceforge.net/>. Júl 2012.

- [HZM12] Eric Haines, Qingnan Zhou, and Patrick Min. Binvox 3d mesh voxelizer <http://www.cs.princeton.edu/~min/binvox/>. November 2012.
- [Inc09] Include. 3d dot game hero editor <http://designer.3ddotgameheroes.eu/>. 2009.
- [Kap12] Arseny Kapoulkine. pugixml <http://pugixml.org/>. Máj 2012.
- [KCY93] Arie Kaufman, Daniel Cohen, and Ronie Yagel. Volume graphics. 26(7):51–64, July 1993.
- [Kin13] Graham King. Zoxel <http://zoxel.blogspot.sk/>. Apríl 2013.
- [Leu03] Michel Leunen. Rotating a bitmap [www.leunen.com/cbuilder/rotbmp.html](http://www.leunen.com/cbuilder/rotbmp.html). 2003.
- [Mic13] Microsoft. Microsoft visual studio 2012 <http://www.microsoft.com/visualstudio/>. 2013.
- [Min12] Patrick Min. Viewvox 3d voxel model viewer <http://www.cs.princeton.edu/~min/viewvox/>. November 2012.
- [Min13] Minddesk. Qubicle <http://www.minddesk.com/>. Október 2005-2013.
- [Mob13] MobyGames. <http://www.mobygames.com/game-group/visual-technique-style-voxel-graphics/>. 1999-2013.
- [Pal08] Tomáš Palkovič. Web rozhranie pre voxelizáciu geometrických dát. Bakalárská práca, Univerzita Komenského v Bratislave Fakulta matematiky, fyziky a informatiky, 2008.

- [PTTK06] George Passalis, George Toderici, Theoharis Theoharis, and Ioannis A. Kakadiaris. General voxelization algorithm with scalable gpu implementation. Marec 2006.
- [Pty13] Beepa Pty. Fraps <http://www.fraps.com/>. 2013.
- [SD06] Ken Silverman and Tom Dobrowolski.  
<http://advsys.net/ken/voxlap.htm>. 2000-2006.
- [Shp11] Andrew Shpagin. <http://3d-coat.com/voxel-sculpting/>. 2011.
- [VG12] Alexey Volynskov and Andrw Gardner. Sproxel  
<https://code.google.com/p/sproxel/>. Október 2012.
- [WWD10] Denton Woods, Nicolas Weber, and Meloni Dario. Devil  
<http://openil.sourceforge.net/>. Jún 2010.
- [YKC92] Ronie Yagel, Arie Kaufman, and Daniel Cohen. Discrete ray tracing. 12:19 – 28, 1992.