Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Editor vizuálních vlastností objektů pro real-time grafiku

Plzeň 2012 Jan Vodička

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. května 2012

Jan Vodička

Abstract

The aim of this work is to acquaint the reader with tools for editing visual properties of objects for real-time graphics and it should be pointed out their weaknesses. The next part of this work is to design and implement custom software solution to control and modify the appearence of objects at both the texture and the shader level. This work should provide the reader an alternative procedure for creating visual properties of an objects.

Obsah

1 Úvod

Tato bakalářská práce by měla v první řadě seznámit čtenáře s existujícími nástroji pro úpravu vizuálních vlastností objektů, určených pro real-time grafiku. Vizuálními vlastnostmi objektů je myšlena textura a shader. Obě tyto složky ovlivňují výsledný vzhled objektů. Textura dodává modelu na detailu a s pomocí programovatelných shaderů se dá dosáhnout nevídané kvality 3D modelů.

Důležitou a hlavní částí této bakalářské práce by mělo být navrhnutí a naimplementování vlastního softwarového řešení pro editaci textury přímo na modelu a shaderů s okamžitým výstupem na obrazovku. Textury modelů by mělo jít kreslit podobně jako v programech typu Photoshop, návrh shaderů by měl být zpříjemněn obarvováním syntaxe a doplňováním kódu.

Práce by měla poskytnout čtenáři alternativní postup pro editaci vizuálních vlastností 3D modelů a výsledný editor by měl zaujmout místo v malém, pomyslném a prázdném prostoru mezi existujícími komerčními programy.

Vytvářený editor by měl být určen pro začínající i pokročilé grafiky, rovněž tak pro vývojáře počítačových her.

2 Než začnete číst

V tomto dokumentu budou v jeho průběhu využívány názvy a pojmy související s počítačovou grafikou. Je tedy dobré, pokud není čtenář obeznámen s termíny související s počítačovou grafikou, aby si o nich přečetl více v následujících publikacích[1].

- *Computer Graphics: Principles and Practice*, James D. Foley, Andries van Dam, Steven K. Feiner a John F. Hughes (Addison-Wesley, 1990) – tato kniha je encyklopedie počítačové grafiky. Je to bohatý zdroj informací, ale bude pravděpodobně lepší, když ji přečtete až s určitými zkušenostmi o tématu.

- *3D Computer Graphics*, Andrew S. Glassner (The Lyons Press, 1994) – tato kniha je netechnický, mírný úvod do počítačové grafiky. Soustřeďuje se spíše na vizuální efekty, které lze vytvořit, než na techniku, jak jich dosáhnout.

- *Moderní počítačová grafika*, Jiří Žára, Bedřich Beneš, Petr Felkel (Computer Press, 2005) – zahrnuje informace o nejvíce používaných metodách počítačové grafiky, jak dvourozměrné, tak trojrozměrné. Je určena pro každého se zájmem o počítačovou grafiku.

Rovněž je dobré přečíst si něco o renderovacích API[2] (OpenGL, Direct3D) z následujích internetových zdrojů.

- *OpenGL*, Wikipedia
  `http://en.wikipedia.org/wiki/OpenGL`

- *Direct3D*, Wikipedia
  `http://en.wikipedia.org/wiki/Microsoft_Direct3D`

[1] Uvedená literatura převzata z [opengl2006].

[2] API (Application Programming Interface) je rozhraní poskytující programátorovi sadu funkcí či tříd nějaké knihovny (popř. jiného programu).

3 Existující software

V této kapitole se zaměříme na existující software, který s navrhovaným editorem sdílí alespoň malou množinu funkcí. Účelem této kapitoly je seznámit čtenáře s různými druhy programů, používajících se při tvorbě 3D grafiky. Rovněž by se měl čtenář dozvědět o slabinách nebo drobných nedostatcích těchto editorů. Ke konci kapitoly by měl čtenář zjistit, v čem se navrhovaný editor bude lišit a jaké funkce, o proti konkurenčním programům, bude poskytovat.

3.1 3DS Max

Autodesk 3ds Max je profesionální program pro 3D grafiku, vizualizace a animace. Bývá používán v postprodukci, při výrobě reklam, filmů a v televizním průmyslu, pro architektonické a konstrukční vizualizace a často slouží i k tvorbě grafiky do počítačových her. 3ds Max je z velké části zaměřen na polygonální modelování, editaci texturových souřadnic, práci se světlem a stínem, tvorbu animací a fotorealistický rendering[1].

Kreslení textury přímo na 3D model nebylo v 3ds Max donedávna možné. Ve verzi 2012 se modul pro kreslení textury objevil (viz. Obr. 3.1), ale dosti komplikovaně se ovládá a podporuje jen malé procento operací, na které jsou lidé zvyklí z programů jako je třeba Photoshop nebo GIMP. Pro vytváření shaderů a zobrazování jejich výstupu v reálném čase není v 3ds Max k dispozici žádný modul.

Cena 3ds Max je opravdu velmi vysoká. Verze 2012 se nyní prodává přibližně za 3500 amerických dolarů, což je v přepočtu asi 64000 Kč.

3.2 Maya

Další profesionální program pro tvorbu 3D grafiky, 3D modelů a animací, který je vyvíjen společností Autodesk. Je hojně využíván při vývoji počítačových her, hlavně kvůli pokročilému systému pro tvorbu animací postav, obličejové mimiky atd.

Program Maya podporuje velice primitivní nástroje pro tvoření textury přímo na 3D modelu a stejně jako v programu 3ds Max není k dispozici žádný modul, který by umožňoval tvorbu shaderů.

[1] Převzato z [3sdmax].

Obrázek 3.1: Ukázka programu 3DS Max 2012.

Cena nejnovější verze je přibližně stejná jako cena 3ds Max, tj. 3500 dolarů.

3.3 Blender

Blender je multi-platformní open-source software pro vytváření 3D grafiky a animací. Je vyvíjen společností NaN (dříve NeoGeo). Blender se svými funkcemi vyrovná editorům jako je 3ds Max nebo Maya, bohužel jej znevýhodňuje komplikované uživatelské rozhraní, které se ale postupně, od verze k verzi, zjednodušuje. Narozdíl od konkurenčního softwaru obsahuje vestavěný game engine, díky kterému lze vytvářet velmi jednoduše interaktivní 3D aplikace, především hry.

Vytváření textury přímo na 3D modelu Blender podporuje, ale úroveň tohoto modulu jen lehce převyšuje možnosti konkurenčních programů jako je třeba 3ds Max nebo Maya. Vzhledem k vestavěnému game enginu lze poměrně jednoduše manipulovat s shadery a zobrazovat výstup v reálném čase.

Blender je zcela zdarma.

3.4 Lightwave

Lightwave je dalším programem pro tvorbu 3D grafiky, modelů a animací, tentokrát vyvíjený společností NewTek. V porovnání s programy 3ds Max a Maya neobsahuje tolik vymožeností, ale zakladní poskytované operace jsou ve všech programech téměř totožné.

Tento program umožňuje v omezeném měřítku tvořit texturu přímo na modelu a neobsahuje žádný modul pro tvorbu a zobrazování shaderů.

Ceny Lightwave se pohybují kolem 800 dolarů, ale záleží na verzi programu.

3.5 MARI

MARI je program zaměřený na vytváření textur přímo na 3D modelech. Je vyvíjen společností The Foundry a lze jej provozovat jak na operačním systému Windows, tak na systému Linux. Byl použit při vytváření textur postav, flóry a fauny pro film Avatar. Je poměrně náročný na výkon hardwaru počítače.

Aplikace MARI byla vytvořena především pro tvorbu textur přímo na modelu (viz. Obr. 3.2[2]). Samotná tvorba je velice propracovaná, což dělá MARI jeden z nejlepších produktů zaměřených na tuto činnost. Tento editor neumožňuje editovat shadery přímo, ale je tu možnost vybrat a použít již předdefinované shadery, které zvýší kvalitu výstupu.

Cena této aplikace se pohybuje kolem 1000 dolarů.

3.6 ZBrush

ZBrush je digitální nástroj pro sochařské 3D modelování, texturování a kreslení. Vyvíjí ho společnost Pixologic.

Stejně jako program MARI umožňuje ZBrush vytváření textury přímo na modelu na vysoké úrovni. Texturování je velice pohodlné. ZBrush nenabízí žádnou možnost pro vytváření shaderů, ale stejně jako MARI poskytuje předdefinované shadery pro různé typy povrchů a materiálů.

ZBrush se dá zakoupit přibližně za 700 dolarů.

[2]Obrázek převzat z [pmari].

Obrázek 3.2: Ukázka programu MARI.

3.7 Sculptris

Sculptris je volně šiřitelná aplikace, původně vyvíjená programátorem Tomasem Petterssonem, nyní se o jeho další vývoj stará společnost Pixologic. Tento editor se svými funkcemi nejvíce podobá aplikaci ZBrush. Stejně tak jako ZBrush je Scultpris určen pro sochařské modelování a texturování. Podobně jako předchozí aplikace neumožňuje tvorbu vlastních shaderů.

3.8 MudBox

Další sochařský modelovací software vyvíjený firmou Autodesk. Obsahuje téměř stejnou množinu operací jako ZBrush a Sculptris. V tomto programu je možno vytvářet modely s vysokými detaily a texturovat je. Aplikace neobsahuje žádný modul pro vytváření vlastních shaderů.

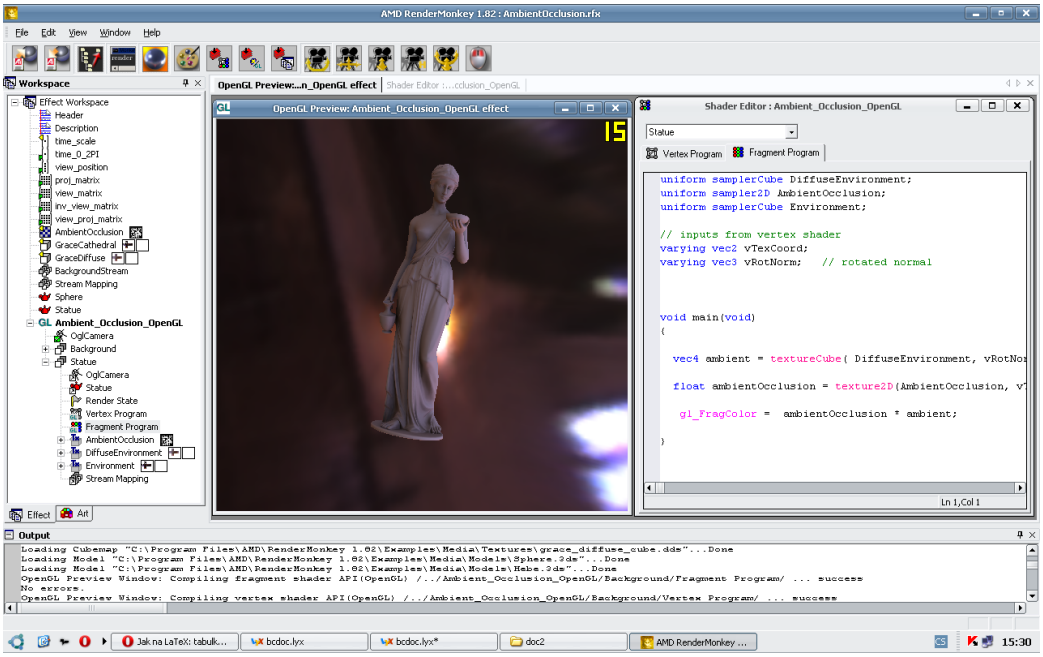
Mudbox lze zakoupit přibližně za 900 dolarů.

3.9 RenderMonkey

RenderMonkey (viz. Obr. 3.3) je prostředí pro vývoj shaderů a jejich vizualizaci. V této aplikaci lze tvořit shadery téměř ve všech jazycích určených

k programování shaderů. RenderMonkey neumožňuje kreslit vlastní textury, pouze využívat již hotové nebo je generovat proceduralně. Program vyvíjela společnost AMD, ale v této době už není dále podporován.

Aplikace je přístupná zcela zdarma.



Obrázek 3.3: Ukázka programu RenderMonkey.

3.10 FX Composer

Další program pro vývoj shaderů a materiálů pro 3D objekty od společnosti NVIDIA. Na rozdíl od aplikace RenderMonkey umožňuje vytvářet shadery jen v jazycích HLSL[3] a Cg[4]. V programu FX Composer je rovněž možné vytvářet částice. Lze jej využívat zdarma.

3.11 Lumina

Lumina je multiplatformní vývojové prostředí pro tvorbu shaderů v jazyce GLSL[5]. Díky vestavěnému skriptovacímu jazyku lze přímo řídit způsob vy-

[3]HLSL (High Level Shader Language) je jazyk pro psání shaderů od společnosti Microsoft, určen především pro rozhraní DirectX. Vychází z jazyka C.

[4]Cg (C for Graphics) je jazyk pro psání shaderů od společnosti NVIDIA.

[5]GLSL (OpenGL Shading Language) je jazyk pro tvorbu shaderů pro rozhraní OpenGL.

kreslování.

Lumina je přístupná zcela zdarma[6].

3.12 Srovnání s navrhovaným editorem

Většina z výše uvedených editorů byla vyvíjena několik let a jistě budou vyvíjeny i v letech dalších. Rovněž na nich pracují profesionálové. Tyto aplikace jsou tedy většinou velice stabilní, uživatelům poskytují vysoce kvalitní možnosti a funkce, ale také jsou většinou dosti drahé a ne každý si může takto drahý software dovolit.

Navrhovaný editor sice není na tak vysoké úrovni, jako uvedené aplikace, ale může uživatelům a především grafikům nabídnout alternativní postup vytváření textury pro 3D model. Ve většině uvedených editorech není implementována možnost editovat texturu přímo na 3D modelu nebo je tato funkce velice omezená. Navrhovaný editor také nabízí možnost vytvářet jednoduché shadery, přímo z editoru nastavovat jejich vstupy a vidět okamžitý výsledek. Editor rovněž umožňuje automaticky rozbalovat modely (unwrapping).

Srovnání možností navrhovaného editoru s výše uvedenými programy je vidět v tabulce 3.1.

Editor	Kreslení textury	Editace shaderů	Auto-unwrap	Cena v USD
Navrhovaný editor	✔	✔	✔	Zdarma
3DS Max 2012	✔	✖	✔	3500
Maya	✔	✖	✔	3500
Blender	✔	✔	✔	Zdarma
Lightwave	✔	✖	✔	800
MARI	✔	✖	✖	1000
ZBrush	✔	✖	✔	700
Sculptris	✔	✖	✔	Zdarma
MudBox	✔	✖	✔	900
RenderMonkey	✖	✔	✖	Zdarma
FX Composer	✖	✔	✖	Zdarma
Lumina	✖	✔	✖	Zdarma

Tabulka 3.1: Vzájemné srovnání editorů.

[6]Lumina lze stáhnout na [lumina].

4 Vlastní programové řešení

V následující kapitole se budeme věnovat navrhovanému editoru, popisu implementace a poskytovaným funkcím.

Editor byl programován v jazyce C#. Tento jazyk byl vybrán především z důvodu možnosti rychlého návrhu uživatelského rozhraní. Pro real-time vykreslování byla využita knihovna OpenGL[1], ale pro možnosti rozbalování modelů (unwrapping) byla zahrnuta i knihovna DirectX[2].

4.1 Požadavky pro spuštění

Aby bylo možné aplikaci spustit, musí být splněno několik požadavků:

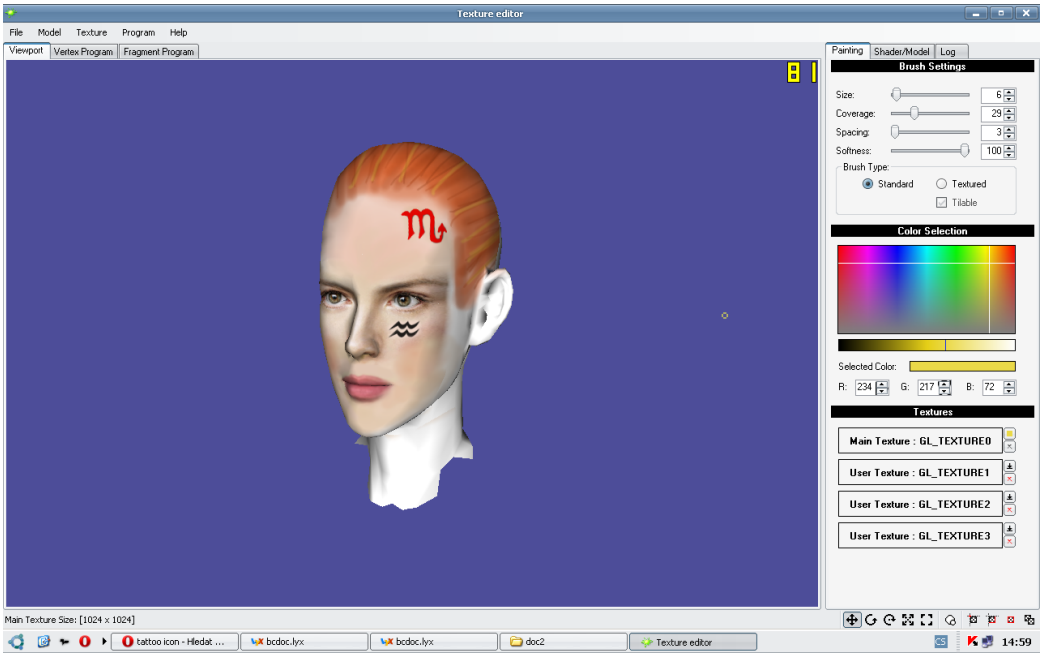
1. Operační systém Windows XP a vyšší (testováno na systému Windows XP a Windows 7).
2. Pro rychlý běh programu postačí průměrné PC kancelářského typu (CPU: 1,6+ GHz, 512 MB RAM).
3. Grafická karta musí podporovat OpenGL 1.5 nebo vyšší.
4. Na stanici (PC) je třeba mít nainstalováno .NET Framework verze 4 a všechny nižší.
5. Je potřeba mít v počítači nainstalovanou knihovnu SlimDX, konkrétně 32-bitovou verzi.

4.2 Vlastnosti editoru

Hlavní vlastností editoru by měla být možnost kreslení textury přímo na model bez použití programu třetí strany. Pro případ, že načtený model nebude obsahovat vlastní texturovací souřadnice, je třeba aby program umožňoval automatické přidělování texturovacích souřadnic (rozbalování modelu). Druhotná funkce spočívá v editaci GLSL shaderů a v real-time manipulaci se vstupními (uniformními) atributy shaderu. Editor musí umožňovat export modelu (pro případ automaticky vygenerovaných texturovacích souřadnic) a export nakreslené textury. Na obrázku 4.1 je ukázka navrhovaného editoru.

[1] Přesněji OpenGL4NET, podrobnější informace naleznete na [gl4net].

[2] Přesněji knihovna SlimDX, což je obalovací třída (wrapper), poskytující funkce z DirectX. Více na [slimDX].



Obrázek 4.1: Ukázka navrhovaného editoru.

4.3 Modely

Abychom mohli začít editovat vizuální vlastnosti objektů, je třeba nejprve nějaký objekt do editoru načíst. Objektem se pochopitelně myslí 3D model, složený ze skupiny geometrických primitiv. Většinou jsou těmito primitivy trojúhelníky, ale model můžou tvořit i například čtyřúhelníky nebo obecné polygony. Navrhovaný editor pracuje s modely jako se skupinou trojúhelníků.

4.3.1 Formát souboru modelů

Pro načítání a export modelů byl vybrán textový formát Wavefront OBJ. Obsahuje seznamy vrcholů, normál, texturovacích souřadnic a polygonů. Při načítání modelů ze souboru procházíme soubor po řádcích. Na jednotlivých řádcích jsou uložena data, navzájem oddělena mezerou (popř. více mezerami). Každý řádek může obsahovat jen jeden typ dat (vrchol, normálu, apod.), jednoznačně určený řetězcem znaků na začátku řádku. Editor podporuje tyto typy dat:

- Data vrcholů
  - Geometrické souřadnice vrcholu (v)

  - Texturovací souřadnice (vt)
  - Normála (vn)

- Elementy
  - Polygon/face (f)

- Skupiny
  - Název skupiny (g) – slouží k oddělení jednotlivých modelů v souboru

Na následujích řádcích je ukázka krátkého OBJ souboru s jednoduchým trojúhelníkem, bez texturovacích souřadnic a bez normál:

```
g trojuhelnik
v 0.0 0.0 0.0
v -1.0 1.0 0.0
v 1.0 1.0 0.0
f 1 2 3
```

Formát OBJ souboru umožňuje popsat data modelů (nebo celých scén) i dalšími způsoby, ale ty náš editor nepodporuje. V našem případě si vystačíme jen s daty, které jsou uvedeny výše.

Podrobnější popis formátu OBJ, použitého ve vytvářeném editoru, můžete najít na [obj1], úplný popis formátu Wavefront OBJ pak na [obj2].

4.3.2 Princip načítání modelů

Abychom odstranili drobné rozdíly (pořadí definic) v OBJ souborech, bylo třeba navrhnout takový postup čtení a zpracování dat, který by s rozdíly počítal. Řešení je kupodivu velice jednoduché. V první řadě je třeba připravit dynamické seznamy vrcholů, normál, texturovacích souřadnic a indexů, popisujících uspořádání trojúhelníků. Následně je třeba procházet soubor po řádcích. V případě, že řádek obsahuje příznak dat vrcholů (v, vt, vn) nebo příznak elementu (f), naparsujeme potřebná data a přidáme je do odpovídajícího seznamu. Pokud narazíme na příznak skupiny (g), uložíme si do pomocné proměné název této skupiny. Pokud se tak stalo poprvé, je tento příznak dále ignorován, pokud ne, došli jsme do fáze, kdy je třeba z načtených dat vytvořit nový model. Vždy, po vytvoření nového modelu, je potřeba smazat

seznam indexů. Ostatní seznamy musí být zachovány, protože podle specifikace OBJ souborů může jedna geometrie používat libovolná, již načtená data. Tento postup se opakuje do doby, než jsou všechny skupiny (modely) načteny. V případě, že jsme došli na konec souboru a přesto stále existují nějaké indexy trojúhelníků, vytvoříme z nich nový model.

Data jednotlivých modelů jsou uložena jak v běžné pameti počítače, tak i na grafické kartě. Data v běžné paměti jsou potřebná hlavně pro export do souboru, k samotnému vykreslování se nepoužívají. Naopak data na grafické kartě se používají pouze k vykreslování modelů.

4.4 Automatický unwrap

Automatické rozbalování modelů bylo do editoru implementováno ze dvou důvodů. V prvním případě se snažíme poskytnout uživateli možnost automaticky přidělit modelu texturovací souřadnice, který žádné nemá. Kdyby model při editaci textury texturovací souřadnice neměl, nedalo by se na něj nic nakreslit. Druhý důvod byl ten, že obvykle potřebujeme, aby každý trojúhelník modelu (popř. modelů) byl v UV mapě (mapa všech texturovacích souřadnic modelu) umístěn tak, aby neprotínal žádné jiné trojúhelníky. Kdyby se dva či více trojúhelníků překrývalo, nemuselo by kreslení na model fungovat správně. Problém spočívá v paralelním zpracování jednotlivých primitiv, což může mít za následek nepředvídatelné chování na úrovni pixelů.

Naprogramovat takový automatický unwrapper je poměrně složité. Je potřeba hlídat, aby se žádné dva trojúhelníky neprotínaly, zároveň je ale potřeba zachovávat co nejvíce sousednost trojúhelníků (sousedy jsou tehdy, sdílejí spolu hranu). Rozbalování modelů záleží také na estetickém cítění. To bohužel počítač většinou postrádá. V našem případě jsme nový unwrapper neimplementovali, protože by to svou obtížností vystačilo na samostatnou diplomovou práci. Raději byl použit již hotový unwrapper.

Do našeho editoru byl zahrnut rozbalovací systém z knihovny SlimDX (resp. DirectX), nazvaný UVAtlas[3]. Přes počáteční problémy, způsobené absencí rozumné dokumentace, se povedlo unwrapper uvést do chodu.

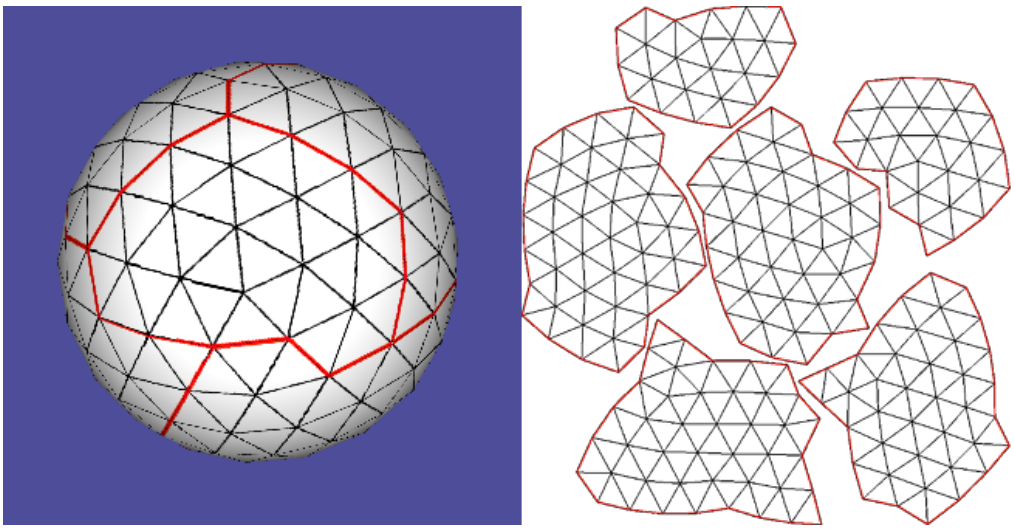
Doba výpočtu zavisí na frekvenci procesoru počítače, počtu trojúhelníků a složitosti rozbalovaného modelu.

4.4.1 Použití UVAtlasu v editoru

UVAtlas pracuje s daty modelů uložených ve speciální třídě z knihovny SlimDX s názvem `Mesh`. Jelikož se z dokumentace nedalo zjistit, jak přesně

[3]Více o UVAtlasu na [uvatlas].

nahrát potřebná data našich modelů do objektu této třídy, bylo potřeba vymyslet postup vlastní. Nejjednodušším řešením bylo vyexportovat naše uložená data modelu ve formátu X[4], ten následně načíst do objektu typu `Mesh` a zavolat metodu, která spustí rozbalování modelu. Po úspěšném rozbalení se data musela opět vyexportovat ve formátu X. Z tohoto souboru už stačilo naparsovat nové texturovací souřadnice a aktualizovat všechny modely načtené v editoru. Na obrázku 4.2 je ukázka rozbaleného modelu.



Obrázek 4.2: Ukázka rozbalené koule.

4.5 Editace textury

Tvorba vlastních textur je fundamentální funkce celého editoru. Proto si některé části programu popíšeme podrobněji.

V první řadě se musí správně navzorkovat tahy myší. Na místech těchto vzorků se vykreslí čtvercové polygony s požadovanou texturou. Tyto polygony se kreslí do zvláštní textury. V dalším kroku se tato textura vzorkuje a převádí do UV mapy. Vzniká přírustková textura modelu. V této textuře se musí opravit okraje trojúhelníkových skupin. Nakonec se finální přírustková textura přičte ke skutečné textuře modelu. V této fázi dochází k ukládání bitmap do historie změn. Tímto jsme popsali editaci textury v kostce. Nyní si rozebereme detaily jednotlivých částí.

[4]Popis formátu DirectX lze nalézt na [xfile].

4.5.1 Vzorkování tahů myši

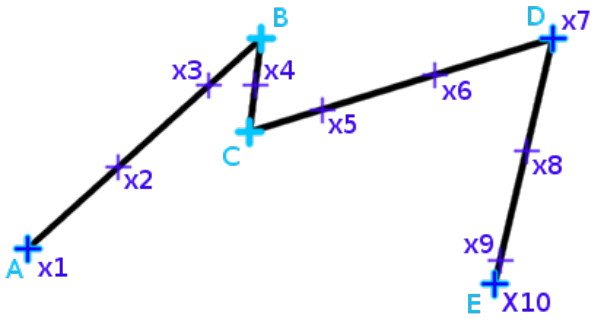
Když kreslíme tahy štětce, obvykle se nám hodí nějaký malý rozestup mezi vykreslovanými čtverci. Při správném rozestupu pak pro dosažení spojitých čar nepotřebujeme tolik vykreslovat. A čím méně vykreslujeme, tím svižněji aplikace běží.

Body `A` až `E` udávají jednotlivé pohyby myší. Body `x1` až `x10` jsou navzorkované tahy štětce (viz. Obr. 4.3). Následující kód v pseudojazyce ukazuje jak se vypočítají body `x1` až `x10`:

```
ArrayOfPoints pts[5] = { A, B, C, D, E }
AddNewPoint(A)
Float traveledDistance = 0.0

for I=1 to 4 do

    Float strokeDistance = PointDistance(pts[I-1], pts[I])
    Float newDistance = traveledDistance + strokeDistance
    Int pointCount = newDistance / spacing
    Float interpolationValue = (spacing - distance) / strokeDistance
    Float interpolationIncrement = spacing / strokeDistance

    for J=1 to pointCount do

        Point p = pts[I-1] + (pts[I] - pts[I-1]) * interpolationValue
        AddNewPoint(p)
        interpolationValue += interpolationIncrement
    end

    traveledDistance = newDistance % spacing
end
AddNewPoint(E)
```

Hodnota `spacing` určuje rozestup mezi body. Metoda `AddNewPoint` přidává nový bod (v našem případě `x1` až `x10`). Tento algoritmus se v editoru provádí v podstatě při každém pohybu myši (při stisklém tlačítku pro kreslení).

Výsledek tohoto algoritmu je vidět na obrázku 4.3.



Obrázek 4.3: Ukázka vzorkování.

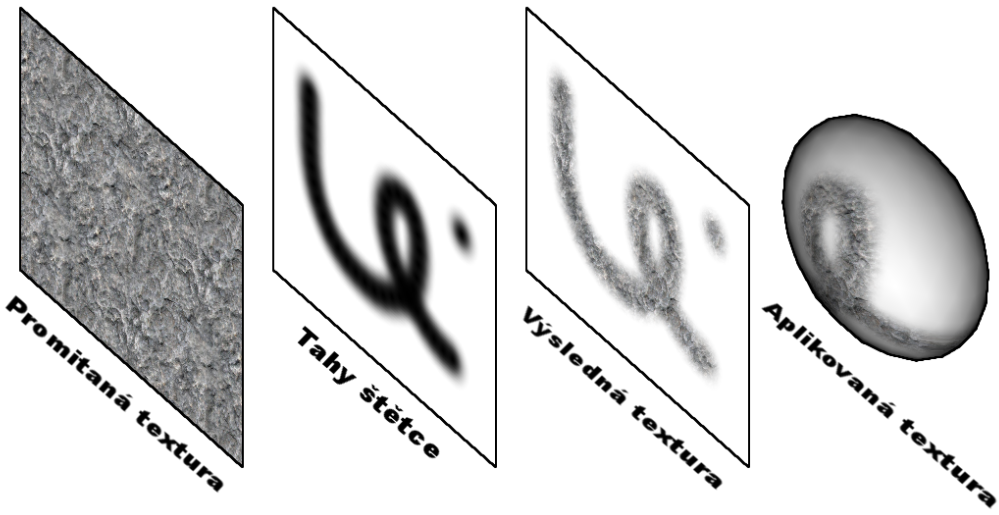
4.5.2 Typy štětců

V editoru existují tři typy štětců:

1. Obyčejný, barevný štětec kulatého tvaru, který na okrajích přechází do ztracena (průhlednost).

2. Texturový štětec, který na místo obyčejné barvy kreslí na model celou vybranou texturu.

3. Opakovatelný texturový štětec, který jako v předchozím případě kreslí na model celou vybranou texturu, ale textura je mapovaná z pohledu obrazovky a na okrajích štětce přechází do ztracena, takže výsledek je mnohem hladší a plynulejší. Princip kreslení opakovatelného štětce je vidět na obrázku 4.4.



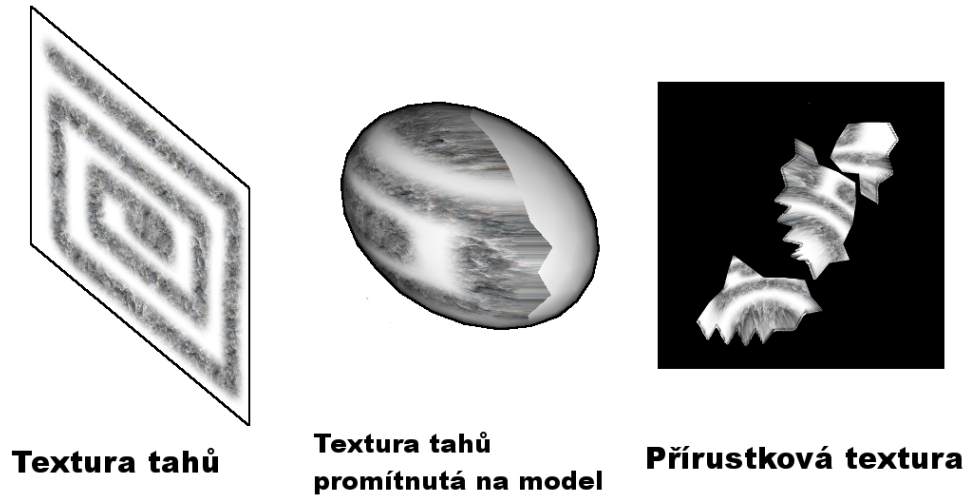
Obrázek 4.4: Princip kreslení opakovatelného štětce.

4.5.3 Tvorba přírustkové textury

Přírustková textura je vlastně obrázek právě nakresleného štětce, převedený do UV prostoru modelu. Vždy, po nakreslení nového tahu se tento přírustek přičte k finální textuře.

Přírustková textura vzikne tak, že se při vykreslování modelu použijí místo geometrických souřadnic vrcholů texturovací souřadnice. Model se tedy

promítne do UV mapy. Před tím, než se model převede do UV mapy, vzorkuje se textura tahů štětce z pohledu obrazovky (viz. Obr. 4.5).



Obrázek 4.5: Promítání textury tahů do přírustkové textury (černé plochy přírustkové textury zastupují průhlednost).

Důležité je zmínit, že do přírustkové textury se vykreslují jen pixely trojúhelníků, které jsou vidět na obrazovce, tzn. pixely odvrácených trojúhelníků nebo pixely zakrytých částí trojúhelníků jinými trojúhelníky se nevykreslují.

Převod textury tahů na přírustkovou texturu je jedna z nesložitějších a výpočetně nejnáročnějších operací použitých v navrhovaném editoru.

4.5.4 Korekce okrajů

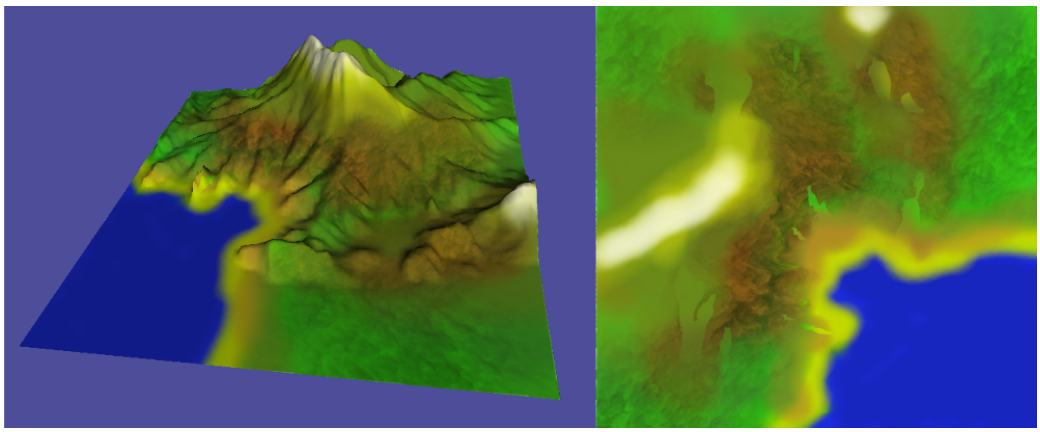
Korekce okrajů je velice důležitá součást vytváření nové textury. Kdybychom korekci neprováděli, tak na hranách trojúhelníků, které nemají sousedy, budou vznikat velice nepřirozené švy a budou kazit tak celý dojem z výsledné textury. Tyto švy se dají poměrně snadno odstranit tak, že se po okrajích trojúhelníkových skupin vytvoří okraj o tloušťce 1 pixelu, používající zprůměrované barvy ze svého okolí. Šev sice úplně nezmizí, ale člověk si ho už téměř nevšimne.

4.5.5 Sloučení textur

Poslední fáze spočívá ve sloučení finální textury a opravené přírustkové textury. Po sloučení je třeba ještě přidat data nové textury do historie změn,

aby bylo možné, při nepovedeném tahu štětce, vrátit původní stav editované textury.

Ukázka výsledku texturování je na obrázku 4.6.



Obrázek 4.6: Ukázka výsledku texturování.

4.6 Editace shaderů

Vyvíjený editor umožňuje programování jednoduchých GLSL shaderů. Díky shaderům můžeme ovlivňovat způsob vykreslování jak na úrovni vrcholů, tak i na úrovni pixelů.

Pro psání formátovaných zdrojových kódu jsme využili vizuální komponentu ScintillaNET[5]. Tato komponenta umí hned několik věcí. První a důležitou vlastností je obarvování zdrojových kódu (viz Obr. 4.7). Umí obarvovat komentáře, klíčová slova, řetězce i čísla. Druhou vlastností Scintilly je schopnost inteligentního odsazování kódu. Zvyšuje se díky tomu tak přehlednost a čitelnost zdrojových kódů. Poslední neméně důležitá schopnost Scintilly je doplňování kódu (viz. Obr. 4.8). V dnešní době je zvykem, aby uměl editor programátorovi radit, co může napsat.

4.6.1 Vstupní atributy shaderů

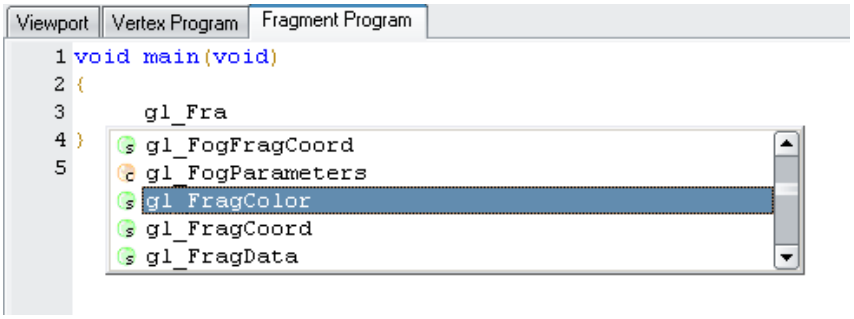
Jedním z požadavků na editor shaderů je, aby umožňoval nastavování vstupních (uniformních) atributů. Díky těmto atributům můžeme zvenčí ovlivňovat způsob vykreslování modelů.

[5]Více naleznete na [scinNET].

Viewport | Vertex Program | Fragment Program

```
1  uniform sampler2D tex;
2
3  varying vec3 normal;
4  varying vec3 lightDirection;
5
6  void main(void)
7  {
8       vec4 sample = texture2D(tex, gl_TexCoord[0].xy);
9       vec3 n = normalize(normal);
10
11      float light = 0.15 + clamp(dot(lightDirection, n), 0.0, 0.85);
12      gl_FragColor = vec4(sample.rgb*light, sample.a);
13 }
14
```

Obrázek 4.7: Ukázka obarveného zdrojového kódu fragment shaderu.



Obrázek 4.8: Ukázka našeptávání.

V prvé řadě musíme naparsovat vstupní atributy ze zdrojových kódů shaderu. Tyto atributy začínají klíčovým slovem `uniform`. Potom obvykle následuje datový typ a název proměnné (viz. Obr. 4.7). Uniformní atributy mohou obsahovat oba zdrojové kódy, jak vertex shader[6], tak i fragment shader[7]. Dokonce mohou stejný atribut oba sdílet. V rámci jednoho shaderu (vertex nebo fragment) může být atribut pojmenován stejným jménem pouze jednou. Tyto pravidla musela být brána v potaz při vytváření seznamu uniformních atributů.

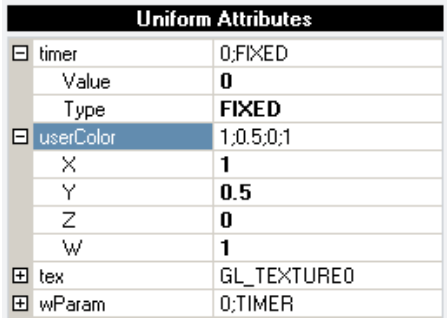
Parsování vstupních atributů probíhá ihned po spuštění kompilace shader programů. V případě, že došlo ke kompilační chybě, jsou atributy z předchozí kompilace zachovány a to pro případ, že by uživatel program opravil a chtěl pokračovat s nastavováním atributů. V prvních verzích programu nebylo toto

[6]Vertex shader je program, který se provede při zpracování každého vrcholu geometrie.

[7]Fragment (pixel) shader je program, který se provede při zpracování každého pixelu scény.

bráno v potaz a atributy při neúspěšné kompilaci zmizely. Po opravení chyb a opětovné kompilaci se atributy opět objevily, ale s implicitními hodnotami editoru a ne s těmi, které uživatel naposledy nastavil.

Pro zobrazování uniformních atributů v editoru byla použita komponenta PropertyGrid (viz. Obr. 4.9).



Obrázek 4.9: Ukázka uniformních atributů v PropertyGrid komponentě.

4.7 Export dat

Editor by postrádal smysl, kdyby neumožňoval uložit vytvořená díla do souboru. Tento program umožňuje exportovat jak texturu tak i model.

4.7.1 Export modelu

Pravděpodobně si říkáte, proč bychom měli mít možnost exportovat model, když ho stejně nemůžeme v editoru měnit. Opak je ale pravdou. Nesmíme zapomínat na automatické přidělování texturovacích souřadnic. Pokud bychom si model s nagenerovanými souřadnicemi neuložili, výsledná textura by nám na původní model pravděpodobně nepasovala.

Během exportu probíhá optimalizace modelu. Snažíme se zredukovat počet vrcholů, normál i texturovacích souřadnic a to tak, že odstraňujeme duplicity. Pokud provádíme takovou operaci, je třeba přepočítávat indexy, reprezentující trojúhelníky modelu.

Pro exportování modelu je opět využíván formát Wavefront OBJ.

4.7.2 Export textury

Export textury je asi po samotném kreslení to nejduležitější, co editor musí umět. Pokud bychom výslednou texturu nemohli uložit do souboru, byl by editor k ničemu.

Prvním důležitým krokem bylo zvolení vhodného obrazového formátu. Byl vybrán formát PNG, protože podporuje alfa kanál, umí obrázek komprimovat bezeztrátovou kompresí a navíc operace s tímto formátem jsou implementovány ve třídě `Bitmap`, obsažené v základních balíčcích jazyka C#.

Data textury, uložené v paměti grafické karty, nakopírujeme do běžné pracovní paměti. Jeden pixel zabírá 4B dat. Každý byte reprezentuje jednu barevnou složku s 256 stupni intenzity. Obrázky ve formátu PNG ukládají barevné složky v pořadí BGRA, data poskytnutá z paměti grafické karty jsou ale ve formátu RGBA. Před uložením bylo tedy třeba prohodit červený a modrý kanál.

4.8 Možná vylepšení

Přestože navrhovaný editor disponuje řadou užitečných funkcí, obsahuje určité nedostatky, které by bylo dobré v budoucnu odstranit. V následujícím textu projdeme hlavní nedostatky programu.

4.8.1 Ukládání rozpracovaného projektu

Při práci na nějakém projektu chceme mít obvykle možnost uložit si nedokončenou práci a pokračovat v ní později. Vlivem nedostatku času se na tuto důležitou funkci nedostalo. V editoru existuje sice možnost, jak uložit rozpracovanou texturu a aktuální model, ale program již neumožňuje uložit zdrojové programy shaderů, nastavení jejich vstupních atributů, což se postupem času může stát velkým problémem. V dalších verzích tohoto programu by měl být tento nedostatek dozajista odstraněn.

4.8.2 Podpora alfa kanálu

Nynější verze programu neumožňuje použití průhlednosti ve vytvářené textuře. Důvodem absence alfa kanálu byla nedostatečná analýza postupu pro kreslení textury a špatná znalost blendingu[8]. O přidání alfa kanálu se uvažovalo, ale po provedení několika testů se špatnými výsledky (viz. Obr. 4.10), byla tato možnost zamítnuta. Toto chování je způsobeno tím, že místa v bitmapě, která jsou průhledná, mají ve skutečnosti ještě nějakou barvu, ale ta není vidět. Když dochází k míchání výsledné textury s nakresleným tahem štětce, použije se jako stará barva pixelu právě ta barva, která není vidět. V našem

[8]Blending (nebo také míchání) je postup, kdy se kombinuje barva nového pixelu s barvou starého pixelu, čímž vzniká pixel nový.

programu je vždy barva pozadí bílá, proto se kolem tahu štětce vytvořil bílý okraj.

Obrázek 4.10: Srovnání požadovaného výsledku s dosaženým výsledkem při použití alfa kanálu (šachovnicový vzor představuje alfa kanál).

4.8.3 Mechanizmus výběru

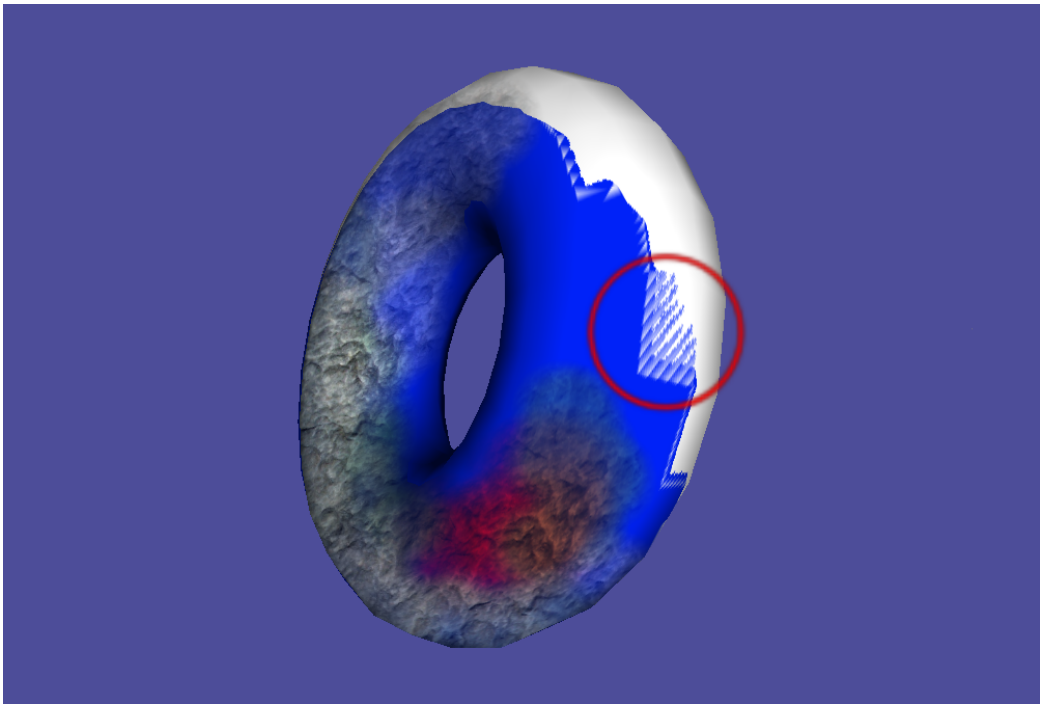
Někdy se může hodit možnost, vybrat si místa, kam lze a kam nelze kreslit. Určitě by nebylo špatné, kdyby existovala možnost vybrat jen ty trojúhelníky modelu, na které chceme kreslit. Také by bylo určitě dobré, mít možnost na obrazovce nakreslit pomocí jednoduchých čar polygon, uvnitř kterého by bylo možné na model kreslit. V dalších verzích editoru by tato funkce byla neocenitelným pomocníkem.

4.8.4 Editor texturových souřadnic

Aktuální verze editoru poskytuje uživateli jen jedinou možnost, jak změnit texturovací souřadnice a to pomocí automatického rozbalování modelu. V dalších verzích programu by se určitě hodilo mít zabudovaný editor UV souřadnic, kde by je mohl uživatel ručně editovat, popř. vybírat různé druhy automatických unwrapperů.

4.8.5 Korekce kreslení

Během kreslení na model, může docházet, při velkém sklonu trojúhelníků vůči pohledu kamery, k nežádoucím jevům, který kazí výslednou texturu (viz. Obr. 4.11). Cílem korekce by měla být schopnost detekovat tyto trojúhelníky a zabránit vykreslování tahu štětce. V aktuální verzi programu tato korekce částečně implementována je, ale bohužel funguje jen na některých typech modelů.



Obrázek 4.11: Nežádoucí jev při velkém sklonu trojúhelníků vůči kameře.

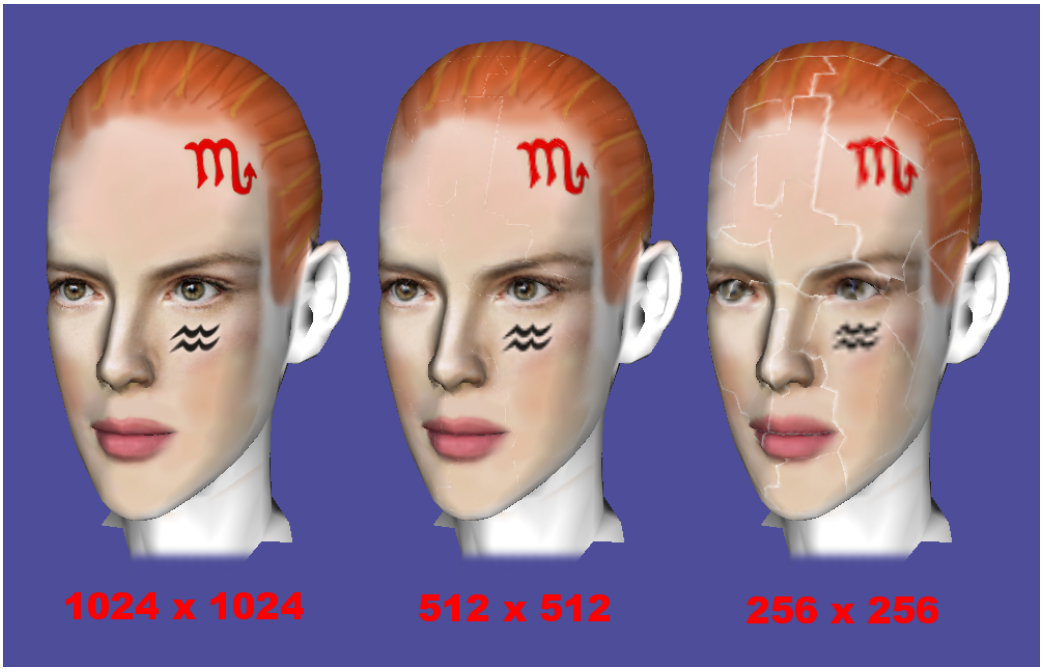
4.8.6 Korekce finální textury

Textury tvořené ve vytvářeném editoru jsou poměrně kvalitní a na modelech vypadají celkem hezky. To se bohužel změní, pokud je použitá technika zvaná mipmapping[9]. Vlivem interpolace, při zmenšování textury, začnou být vidět švy na hranách trojúhelníků, které nemají v UV mapě sousedy (viz. Obr. 4.12). Odstranění tohoto problému není vůbec jednoduché, ale existuje jistá možnost, jak texturu vylepšit, aby švy při použití mipmappingu zmizely. Je potřeba vytvořit kolem skupin trojúhelníků UV mapy okraj o tloušťce několika pixelů, kde se barva tohoto okraje bude měnit, v závislosti na okolních pixelech původní textury.

4.8.7 Vlastní automatický unwrapping

Do budoucna by se určitě vyplatilo navrhnout vlastní řešení automatického přidělování texturovacích souřadnic. Hlavní nevýhodou stávajícího algoritmu je vysoká časová náročnost. Rovněž by chtělo odstranit závislost na knihovně

[9]Mipmapping je technika, kdy se na jeden objekt aplikují zmenšené verze původní textury a to v závislosti na velikosti objektu na obrazovce. Technika urychluje vykreslování a snižuje ve výsledném obrazu zubatost a moaré efekt.



Obrázek 4.12: Zvětšující se šev při použití techniky mipmapping.

DirectX, aby mohl být editor použit i na jiných platformách, než jen na systému Windows.

5 Závěr

Cílem této práce bylo vytvořit editor schopný upravovat vizuální vlastnosti modelů na úrovni textury i shaderu. Vytvořená aplikace umožňuje kreslení přímo na model, čímž poskytuje uživatelům dostatečný komfort při vytváření nových textur, které by jinak jen stěží nakreslili. Rovněž si uživatelé mohou vyzkoušet naprogramovat jednoduché GLSL shadery, upravovat jejich vstupy a vidět okamžité výsledky.

Editor byl převážně inspirován programem MARI, který sice umožňuje editaci textury na mnohem lepší úrovni a rovněž poskytuje uživateli kreslení ve vrstvách, na které jsme zvyklí z programů Photoshop nebo Gimp, ale který je moc náročny na výkon počítače a také příliš drahý, aby si ho mohl běžný uživatel dovolit. Náš program je tedy zcela zdarma a pro spuštění nevyžaduje v počítači nejnovější hardware. Program běží svižně i na 5 let starém počítači.

Náš editor má oproti programu MARI zabudovaný editor shaderů, který zvýrazňuje syntaxi jazyka GLSL a dokonce je schopen jednoduchého doplňování zdrojového kódu, na které je hodně lidí zvyklých z pokročilejších programovacích editorů. Podle průzkumu neumí žádný editor shaderů doplňovat GLSL kód, čímž se naše aplikace stává v tomto ohledu vyjímečná.

I přes drobné nedostatky se s editorem dají dělat divy a proto mám v plánu na této aplikaci v budoucnu pokračovat a zlepšovat jeho možnosti a funkce.

Samotné zadání své bakalářské práce považuji za velmi zajímavé a jsem rád, že jsem mohl vytvářet program, který se bude hodit nejen mně, ale určitě i dalším lidem.

Literatura

[opengl2006] SHREINER, Dave. *OpenGL: průvodce programátora.* Vyd. 5. Brno: Computer Press, 2006, 679 s. ISBN 80-251-1275-6.

[slimdx] SlimDX. *Slimdx.org* [online]. 2009-2011 [cit. 8.5.2012]. Dostupné z: `http://slimdx.org/`

[scin] ScintillaNET. *Codeplex.com* [online]. 2006-2012 [cit. 8.5.2012]. Dostupné z: `http://scintillanet.codeplex.com/`

[uvatlas] UVAtlas. *Msdn.microsoft.com* [online]. 2012 [cit. 8.5.2012]. Dostupné z: `http://msdn.microsoft.com/en-us/library/windows/desktop/bb206321(v=vs.85).aspx`

[obj2] Wavefront Object File. *University of Western Australia* [online]. 2009 [cit. 8.5.2012]. Dostupné z: `http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/`

[obj1] Wikipedia: Wavefront OBJ. *Wikipedia.org* [online]. 2001-2012 [cit. 8.5.2012]. Dostupné z: `http://en.wikipedia.org/wiki/Wavefront_.obj_file`

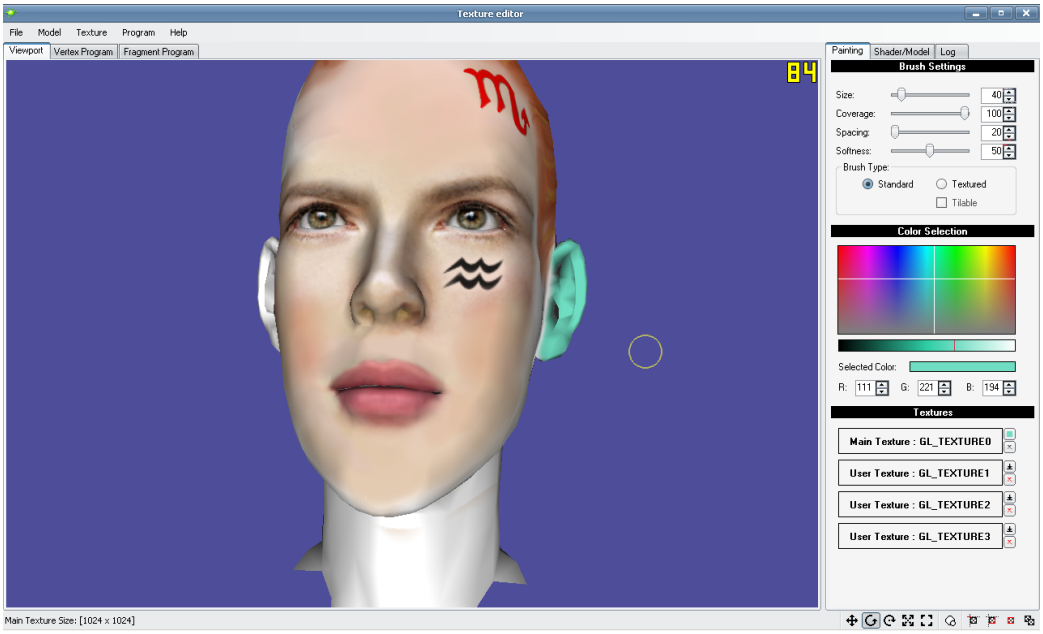
[lumina] Lumina. *Sourceforge.net* [online]. 2008 [cit. 8.5.2012]. Dostupné z: `http://lumina.sourceforge.net/`

[pmari] MARI: Review: The Foundry's Mari. *3Dworldmag.com* [online]. 2011 [cit. 8.6.2012]. Dostupné z: `http://www.3dworldmag.com/2011/01/18/review-the-foundrys-mari/`

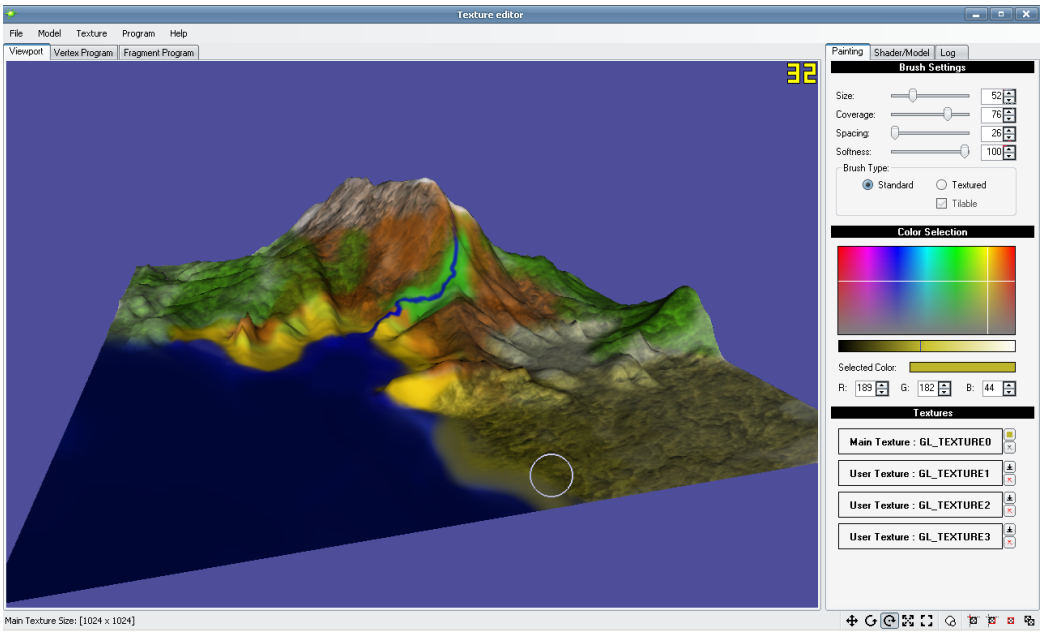
[3dsmax] Wikipedia: 3D Studio MAX. *Wikipedia.org* [online]. 2001-2012 [cit. 8.5.2012]. Dostupné z: `http://cs.wikipedia.org/wiki/3D_Studio_MAX`

[gl4net] OpenGL for .NET. *Heracles.zcu.cz* [online]. 2010-2012 [cit. 8.5.2012]. Dostupné z: `http://herakles.zcu.cz/~pvanecek/OpenGL4net/`

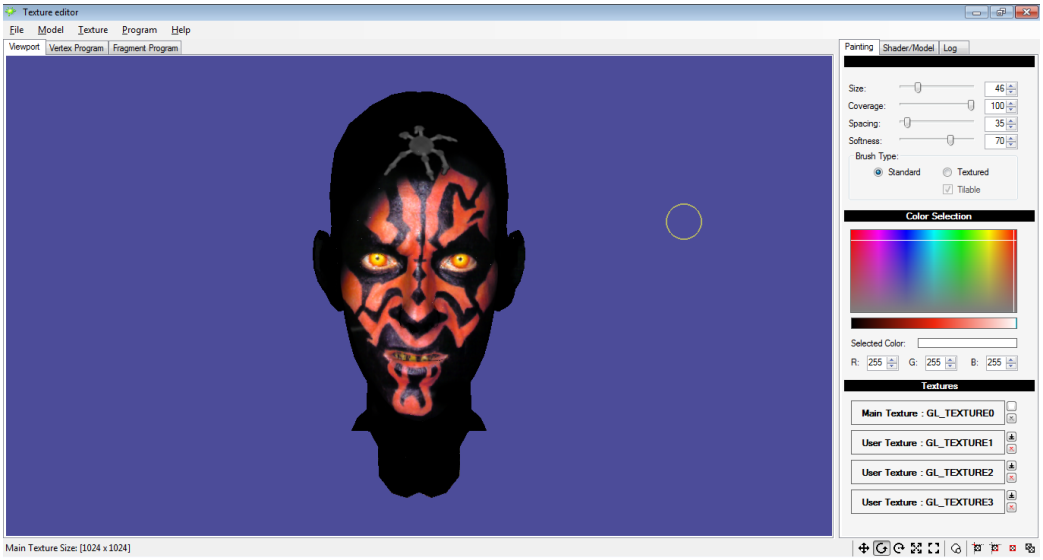
A Příloha: Grafické ukázky



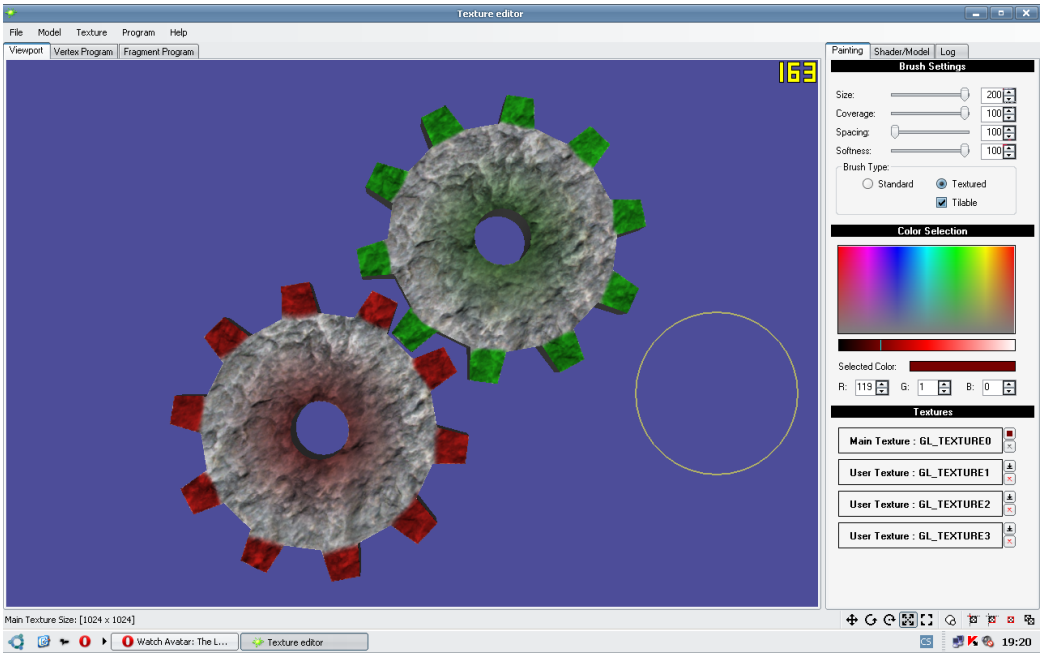
Obrázek A.1: Ukázka vytvořené textury ženského obličeje.



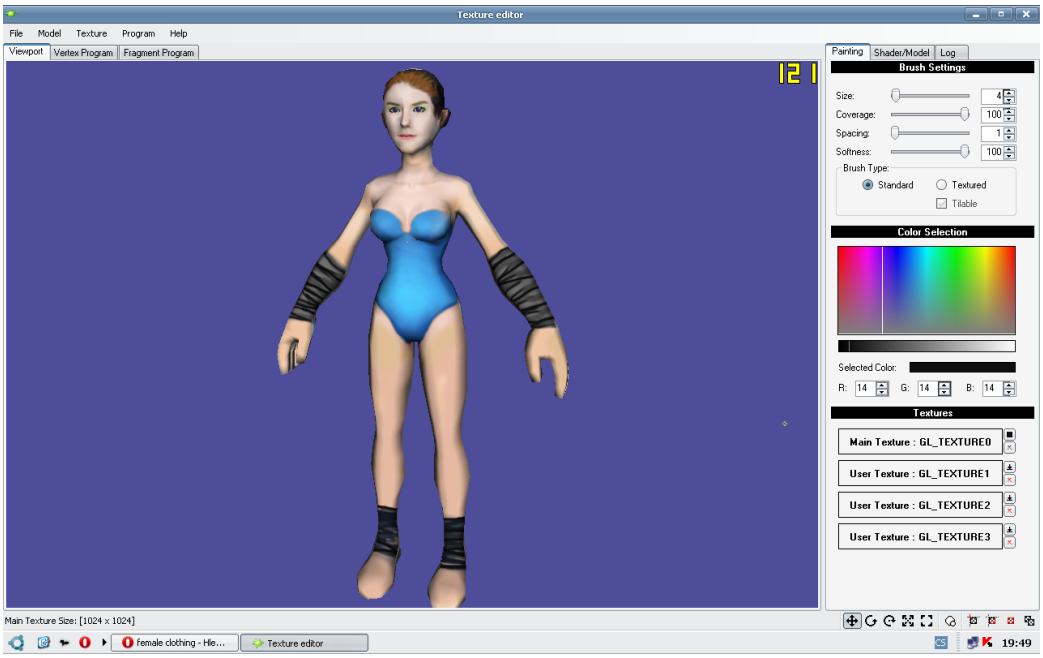
Obrázek A.2: Ukázka vytvořené textury terénu.



Obrázek A.3: Ukázka vytvořené textury Dartha Maula.



Obrázek A.4: Ukázka otexturovaných ozubených kol.



Obrázek A.5: Ukázka otexturovaného modelu ženy.

B Příloha: Obsah CD

Přiložené CD má následující strukturu:

\bin

- Spustitelný soubor programu.
- Soubory využívané editorem (obrázky, ikony, ...).

\doc

- Elektronická verze tohoto dokumentu.

\help

- Uživatelská příručka ve formátu HTML.

\images

- Obrázky uvedené v příloze A.

\install

- Instalační balíčky knihoven potřebných pro správný běh programu.
  - .NET Framework 4
  - SlimDX 32-bit

\obj-study

- Testovací modely ve formátu OBJ.

\src

- Zdrojové soubory programu.

\tex

- Zdroje použité pro vytvoření tohoto dokumentu.