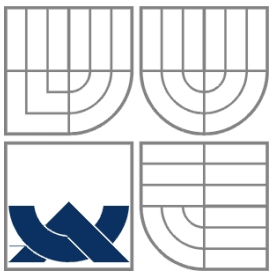
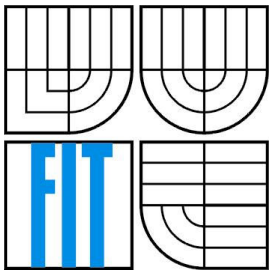


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MAPOVÁNÍ TEXTUR
TEXTURE MAPPING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Zdeněk Strach, Bc.

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. Pavel Zemčík

BRNO 2007

Abstrakt

Diplomový projekt se zabývá problematikou mapování textur v zobrazovací metodě „ray tracing“. V úvodu práce je stručně popsána metoda „ray tracing“ a metody mapování textur, nejprve procedurálních a později 2D. Následující kapitoly se zabývají implementací MIP mappingu a zhodnocením výsledků.

Klíčová slova

mapování textur, ray tracing, procedurální textura, 2D textura, Perlinův šum, turbulence, MIP mapping, antialiasing, bilineární a trilineární interpolace

Abstract

In this master’s thesis I’m engaged in problematic of texture mapping in ray tracing. Ray tracing is shortly described in the beginning. Texture mapping methods are described then, solid textures first and 2D textures follow. Implementation of MIP map method is deeply described in next chapters. The results are evaluated at the end.

Keywords

texture mapping, ray tracing, solid texture, 2D texture, Perlin noise, turbulence, MIP mapping, antialiasing, bilinear and trilinear interpolation

Citace

Strach Zdeněk: Mapování textur, diplomová práce, Brno, FIT VUT v Brně, 2007.

Mapování textur

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Dr. Ing. Pavla Zemčíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Zdeněk Strach
28.7.2007

Poděkování

Chtěl bych na tomto místě poděkovat doc. Dr. Ing. Pavlu Zemčíkovi za odborné vedení a konzultace, které mi poskytl při tvorbě na této diplomové práci.

Obsah

1 Úvod

V dnešní době velmi výkonných a dostupných počítačů se setkáváme stále častěji s použitím reálně vypadající počítačové grafiky. Ať už pro zábavu v počítačových hrách, na Internetu, tak i běžném životě při navrhování a simulování nových výrobků, nebo též v lékařství, kde pomáhají při vykonávání náročných lékařských zákroků.

Zobrazování 3D scén v počítačové grafice lze rozdělit podle realističnosti vykreslení na tři základní metody.

Přímé zobrazování – základem této metody jsou objekty scény, jsou zpracovávány sekvenčně, mezi objekty neexistuje interakce, tudíž ani stíny. Výhodou metody je hardwarová podpora v grafických kartách, vykreslování scén je velmi rychlé, proto se používá ve hrách, při náhledech v CAD systémech, ve virtuální realitě, všude kde je potřeba „real-time“ zobrazování scény a interakce. Nevýhodou metody je malá realističnost zobrazované scény.

Zobrazování po pixelech – pixely jsou zpracovávány sekvenčně, mezi objekty neexistuje globální interakce, stíny vrhané objekty jsou ostré. Vytvářené scény jsou dosti realistické, proto se používají k vytvoření realistických obrazů geometrických modelů, zobrazení medicínských dat, výrobků, atd. Zpracování scény po pixelech je výpočetně náročné, vykreslení je pomalé (desítky až stovky sekund). Využívá Phongův osvětlovací model. Typickou metodou zobrazující po pixelech je ray tracing (casting).

Zobrazování celé scény (globální) – zpracovává se celá scéna, kvůli složité realizaci existují pouze měkké stíny. Vytváří velmi realistické obrazy, pro ještě lepší výsledky se kombinuje s předchozí metodou (ray tracingem). Výpočet scény trvá hodiny. Pro výpočet osvětlení se používají složité fyzikálně správné modely, například Lambertův osvětlovací model. Mezi zástupce této metody patří radiozita a photon mapping.

Tato práce se věnuje druhé z výše zmíněných metod, konkrétně vylepšení vzhledu scén generovaných ray tracingem pomocí mapování textur. V následující kapitole je stručně popsán princip ray tracingu a dále jsou probrány způsoby mapování textur, které se používají v ray tracingu. Ve třetí kapitole se věnuji vlastní implementaci vybrané mapovací metody. Ve čtvrté kapitole jsou publikovány výsledky, jejich porovnání a hodnocení. Kapitola číslo pět uzavírá celou práci zhodnocením dosažených výsledků.

Uvedené informace jsou relevantní pro tuto diplomovou práci. Diplomová práce navazuje na semestrální projekt, v jehož rámci byly splněny první tři body z jejího zadání.

2 Obecné principy – ray tracing a mapování textur

2.1 Ray tracing

Ray tracing je technika generování 2D obrazu ze 3D scény. Ray tracingem v této práci se rozumí tzv. „back ray tracing“, což znamená, že primární paprsky jsou vrhány z kamery (oka) skrz průmětnu do prostoru scény, kde se vyhodnotí výsledná barva pixelu.

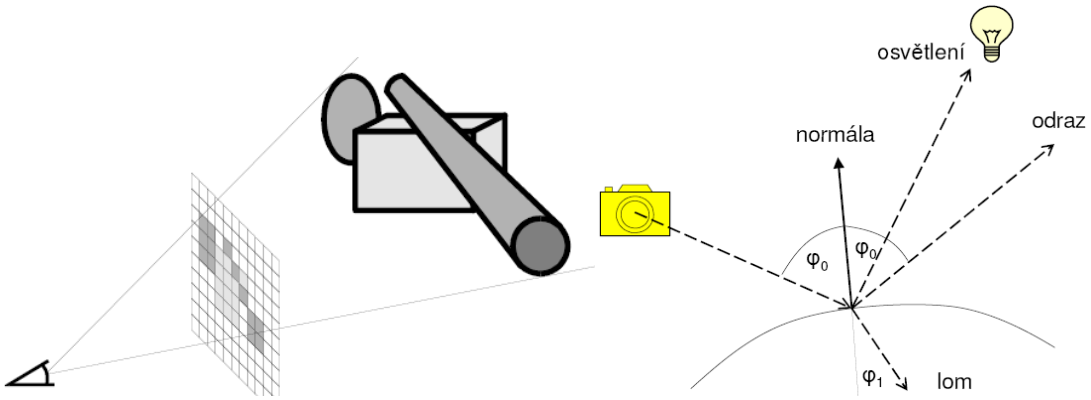
Předpoklady v ray tracingu:

- paprsky světla jsou zjednodušeny na přímky
- při zkřížení více paprsků nedochází ke vzájemné interferenci (nijak se neovlivňují)

Princip ray tracingu

Pro každý pixel

- poslat primární paprsek oko-pixel (v průmětně)-scéna
- zjistit průsečíky se všemi objekty ve scéně
- pro nejbližší průsečík (viditelný) vyhodnotit barvu pixelu ze sekundárních paprsků – stínového (shadow ray), reflexního (reflected ray) a paprsku lomu (refracted ray)
- je-li potřeba, pokračovat rekurzivně ve vyhodnocování sekundárních paprsků



Obrázek 2-1: Princip ray tracingu (vlevo), šíření sekundárních paprsků (vpravo). Převzato z [12].

2.2 Mapování textur

Mapování textur je činnost, kdy se přiřazuje detail povrchu objektu, aniž by se explicitně modeloval jako část geometrie povrchu. Lze též definovat jako proces mapování geometrického bodu na barvu v textuře. Typicky je povrch objektu parametrizován a přes parametry v určitém bodě se přistupuje do textury (2D nebo procedurální). Hodnoty vrácené z textury mohou určit nebo modifikovat kteroukoliv vlastnost povrchu – barvu, lesklost, průhlednost nebo směr normály.

Obrázek 2-2: Ukázka mapování textur v ray tracingu. Převzato z [12].

2.2.1 Procedurální textury

Aby vytvářené obrazy vypadaly reálně, musíme objektům ve scéně nastavit materiálové vlastnosti povrchu a definovat okolní osvětlení. Předpokládejme, že objekty jsou matné, nejsou lesklé. Abychom to udělali, nastavíme každému objektu barvu *R(**p**)*, která se může měnit podle pozice **p**. Například, barvy na obálce časopisu se mění podle pozice a mohou být popsány funkcí *R*. Ray-tracovací program by měl vrátit *R(**p**)*, pokud paprsek protne objekt v bodě **p**. Pro jednobarevné objekty platí, že *R(**p**)* je konstantní. Ale u texturovaných objektů budeme předpokládat, že *R(**p**)* se mění podle polohy **p** na povrchu objektu. Jedním ze způsobů jak vytvořit odpovídající texturu, je definovat barvu (např. RGB) v každém bodě 3D prostoru. Když paprsek protne povrch v bodě **p**, tato funkce je vyhodnocena a výsledkem je barva v daném bodě. Tento způsob se hodí pro povrchy objektů, které jsou vytvarovány jakoby z jednotného materiálu, např. model mramorové sochy.

Procedurální textury jsou tvořeny programem (funkcí nebo procedurou). Někdy jsou též nazývány „matematické textury“, protože bývají definovány matematickým zápisem.

Výhodou procedurálních textur je jejich malá velikost, zabírají řádově kB kódu, naproti tomu „obrázkové“ 2D textury zabírají řádově MB. Procedurální textury také nejsou limitovány diskrétním rozlišením, mohou pokrývat neomezený prostor, proto odpadá problém s navazováním textur a jejich opakováním. Mohou měnit vzhled podle zadaného parametru, program textury tedy generuje ne jednu texturu, ale třídu textur.

Nevýhodou procedurálních textur je jejich obtížné programování. Je těžké představit si výsledný vzhled. Také vyhodnocování procedurální textury za běhu programu je časově náročnější

než přístup do paměti k texelu (texture element). V neposlední řadě se mohou objevovat nepříjemné efekty způsobené aliasingem.

Textury z pruhů (Stripe textures)

Existuje řada možností jak vytvořit takovéto textury. Předpokládejme, že máme dvě barvy c_0 a c_1, které budou tvořit pruhy. Pro střídání barev potřebujeme nějakou oscilační funkci. Mezi jednoduché oscilační funkce patří kosinus:

```
RGB stripe(Vector3 p)
if sin(px) > 0 then
   return c0
else
   return c1
```

Pro kontrolu nad šířkou pruhů lze přidat parametr w:

```
RGB stripe(Vector3 p, float w)
if sin(π*px/w) > 0 then
   return c0
else
   return c1
```

Pro plynulé barevné přechody mezi pruhy poslouží parametr t:

```
RGB stripe(vector3 p, float w)
float t = (1 + sin(π *px/w))/2
return t*c0 + (1 – t)*c1
```

Pomocí výše popsaných funkcí jsou tvořeny textury s pravidelným vzorem. Pokud chceme textury simulující přírodní materiály (např. dřevo, mramor, kouř…), je potřeba vnést do vzoru textury nějaké nepravidelnosti, jelikož nepravidelnost je klíčem k reálnosti. K tomu je dobré použít náhodné číslo $n(\mathbf{p})$ pomalu se měnící v prostoru. U předchozích funkcí pro vzory pruhů stačí změnit $sin(p_x)$ za $sin(p_x + n(\mathbf{p}))$. Toto bude měnit funkci sinu podle změny **p**, jelikož $n(\mathbf{p})$ se mění. Pokud zajistíme, aby se n měnilo nepravidelně a spojitě, a šlo vyhodnotit deterministicky, dosáhneme požadovaného výsledku. V následujícím textu budou zmíněny nejpoužívanější metody.

Šum (Solid noise)

Šum je základním kamenem pro „tvorbu“ nepravidelností v procedurálních texturách. Pokud by se *n* získávalo pouze pomocí náhodných čísel pro každý bod prostoru, výsledkem by byl bílý šum. Ten je pro dané účely nevhodný, není závislý na vstupním parametru a není frekvenčně ohraničený, což způsobuje aliasing. Vhodný šum by měl být závislý na vstupu, měl by mít definovaný rozsah (-1, +1), musí být frekvenčně ohraničený (je známa maximální frekvence), nesmí být viditelně periodický a je nezávislý na posunutí a otočení (stacionární a isotropní).

Jednu z nejpoužívanějších funkcí ***n(p)*** vytvořil Ken Perlin, je známa jako Perlinův šum (Perlin noise). Perlin používá náhodně zvolené vektory ve vrcholech krychlové mřížky, pro vychýlení extrémů z mřížky souřadnic užívá skalární součiny. Pro zahlazení derivačních artefaktů používá interpolaci vyšších řádů (namísto trilineární interpolace). Dále využívá hashování, kvůli velikosti tabulky pro body souřadnic. Následuje základní vztah Perlinova šumu:

$$n(x,y,z)=\sum_{i=\lfloor x\rfloor}^{\lfloor x\rfloor+1}\sum_{i=\lfloor y\rfloor}^{\lfloor y\rfloor+1}\sum_{i=\lfloor z\rfloor}^{\lfloor z\rfloor+1}\Omega_{ijk}(x-i,y-j,z-k) \qquad \textbf{[1]}$$

kde (x, y, z) jsou kartézské souřadnice bodu **p**,

$$\Omega_{ijk}(u,v,w)=\omega(u)\omega(v)\omega(w)(\mathbf{g}_{ijk}\cdot(u,v,w)) \qquad \textbf{[2]}$$

a $\omega(t)$ je kubická váhová funkce:

$$\omega(t)=\begin{cases}-6|t|^6+15|t|^4-10|t|^3+1 & if\ |t|<1\\ 0 & otherwise\end{cases} \qquad \textbf{[3]}$$

Pro vektor $\mathbf{g}_{ijk}$ platí (x, y, z) = (i, j, k). K získání $\mathbf{g}_{ijk}$ se používá předpočítaná pseudonáhodná tabulka N vektorů:

$$\mathbf{g}_{ijk}=\mathbf{G}(\phi(i+\phi(j+\phi(k)))), \qquad \textbf{[4]}$$

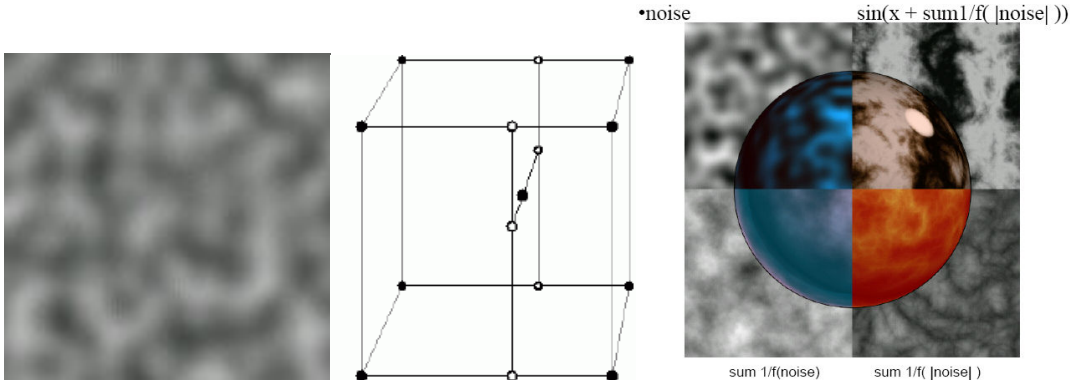
$$\phi(i)=P[i \bmod N], \qquad \textbf{[5]}$$

kde *P* je pole délky *N* obsahující permutace čísel *0* až *N-1*. V praxi se používá *N*=16 a následující vektory:

$$\begin{array}{cccc}(1,1,0) & (1,0,1) & (0,1,1) & (1,1,0)\\ (-1,1,0) & (-1,0,1) & (0,-1,1) & (-1,1,0)\\ (1,-1,0) & (1,0,-1) & (0,1,-1) & (0,-1,1)\\ (-1,-1,0) & (-1,0,-1) & (0,-1,-1) & (0,-1,-1)\end{array}. \qquad \textbf{[6]}$$

Prvních 12 vektorů (3 sloupce zleva) směřují z počátku do 12 hran krychle a poslední 4 vektory jsou jako doplněk kvůli efektivitě operace **mod**.

Perlinův šum je použitelný pro libovolnou dimenzi (např. 4D se používá pro animace), výhodou jsou nízké nároky na paměť. Velmi často je používán v počítačové grafice na efekty jako oheň, kouř nebo mraky.



Obrázek 2-3: Perlinův šum (vlevo), krychlová mřížka s pseudo-gradienty ve vrcholech (uprostřed), variace textur vycházejících z Perlinova šumu (vpravo). Převzato z [9].

Turbulence

Mnoho přírodních textur obsahuje charakteristiky o různých velikostech v jedné textuře. Ken Perlin přišel s pseudofraktální „turbulentní“ funkcí:

$$n_t(\mathbf{p}) = \sum_i^M \frac{n(2^i \mathbf{p})}{2^i}. \qquad [7]$$

Této funkce lze využít k deformaci pruhů, definovaných funkcí popsanou o dvě kapitoly výše:

```
RGB turbstripe(vector3 p, float w)
float t = (1 + sin(k1px + turbulence(k2p))/w)/2
return tc0 + (1 – t)c1
```

Parametry k_1 a k_2 ovlivňují charakter turbulence.

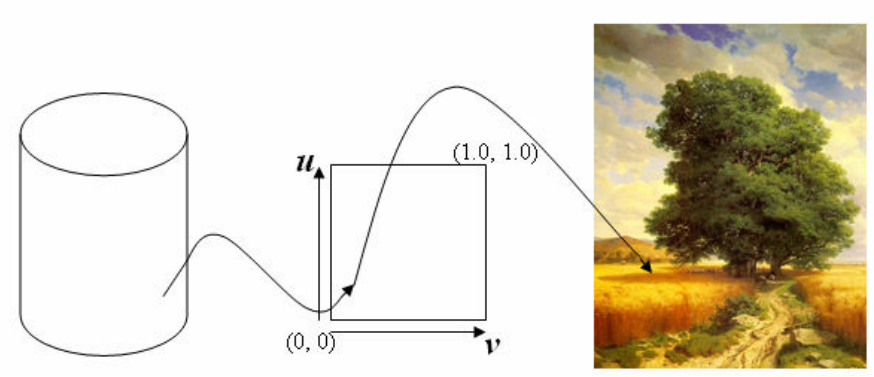
Obrázek 2-4: Příklad použití turbulence. Převzato z [12].

2.2.2 Mapování 2D textur (Image Texture Mapping)

V počítačové grafice je často potřeba namapovat na objekt ve scéně „obrázek“, 2D texturu, aby daný objekt působil co nejreálněji.

Mapování textury můžeme rozdělit do dvou kroků:

- namapujeme bod objektu na bod v abstraktním jednotkovém čtverci, tzv. *(u, v)* souřadném systému, který representuje konkrétní texturu (pixmapu)
- následuje mapování bodu z *(u, v)* souřadného systému na bod v konkrétní textuře



Obrázek 2-5: Postup mapování 2D textury na objekt. Převzato z [8].

Druhý krok je snadný, proto bude vysvětlen dříve. V tomto kroku se provádí transformace bodu z abstraktní spojité plochy do bodu v diskrétní mapě textury. V transformovaném bodě pak získáme požadovanou barvu. Známe-li *(u, v)* souřadnice, tak korespondující bod v textuře má souřadnice *(u*w, v*h)*, kde *w* a *h* jsou šířka (resp. výška) textury v pixelech. Obecně pro libovolný bod *(u, v)* z jednotkové plochy, existuje bod v prostoru textury *(u´, v´)*.

Nyní přejděme k prvnímu kroku mapování textur. Cílem je najít mapování (transformaci) z bodu na objektu *(x, y, z)* do jednotkové plochy *(u, v)*. Existují tři základní mapování:

- mapování na plochu
- mapování na kouli
- mapování na válec

Mapování textury na plochu

Při mapování textury na plochu se užívá desetinná část souřadnic (x, y, z), např. mapujeme-li texturu do roviny x-z (osu y zanedbáme), pak výsledné souřadnice do soustavy (u, v) se vypočtou podle:

```
u = x - floor(x),
v = z - floor(z),
```

kde funkce floor() vrací celočíselný základ parametru.

Mapování textury na kouli

Implicitní rovnice koule s počátkem v bodu (x_c, y_c, z_c) a poloměrem r je:

$$(x-x_c)^2+(y-y_c)^2+(z-z_c)^2=r^2. \quad \textbf{[8]}$$

Parametrizací do sférických souřadnic dostaneme:

$$\begin{aligned} x &= x_c + r\cdot\cos(\theta)\cdot\cos(\phi),\\ y &= y_c + r\cdot\cos(\theta)\cdot\sin(\phi),\\ z &= z_c + r\cdot\sin(\theta), \end{aligned} \quad \textbf{[9]}$$

kde $(\phi,\theta)\in[0,2\pi]\times[-\pi/2,\pi/2]$.

Odpovídající souřadnice jednotkové plochy (u, v) jsou definovány:

$$u=\frac{\phi}{2\pi},\quad v=\frac{(\theta+\pi/2)}{\pi}. \quad \textbf{[10]}$$

Výsledná transformace z (x, y, z) do (u, v) je následující:

$$\begin{aligned} u &= \frac{1}{2\pi}\cdot\arctan 2(y-y_c, x-x_c),\\ v &= \frac{1}{\pi}\cdot\left(\arcsin\frac{z-z_c}{r}\right). \end{aligned} \quad \textbf{[11]}$$

Mapování textur na válec

Válec o výšce H, jehož základna je v rovině x-y, je implicitně definován takto:

$$X^2+Y^2=r^2,\quad 0\le z\le H. \quad \textbf{[12]}$$

Parametrizací do válcových (cylindrických) souřadnic dostaneme:

$$\begin{aligned} X(\theta,h) &= r\cdot\cos\theta,\\ Y(\theta,h) &= r\cdot\sin\theta,\\ Z(\theta,h) &= h, \end{aligned} \quad \textbf{[13]}$$

kde $(\theta,h)\in[0,2\pi]\times[0,H]$.

Afinní transformací získáme libovolný válec:

$$[x(\theta,h),y(\theta,h),z(\theta,h),1]=[X(\theta,h),Y(\theta,h),Z(\theta,h),1]\cdot\begin{bmatrix}\mathbf{A}_{3\times3} & \begin{matrix}0\\0\\0\end{matrix}\\ \mathbf{p}^{\mathbf{T}} & 1\end{bmatrix}, \quad \textbf{[14]}$$

$\mathbf{A}_{3x3}$ musí být ortonormální matice, tzn., že řádkové vektory musí být jednotkové a navzájem kolmé. Pro (u, v) platí:

$$u = \frac{\theta}{2\pi}, \quad v = \frac{h}{H}. \qquad \textbf{[15]}$$

Výsledná transformace se vypočítá ze vztahu:

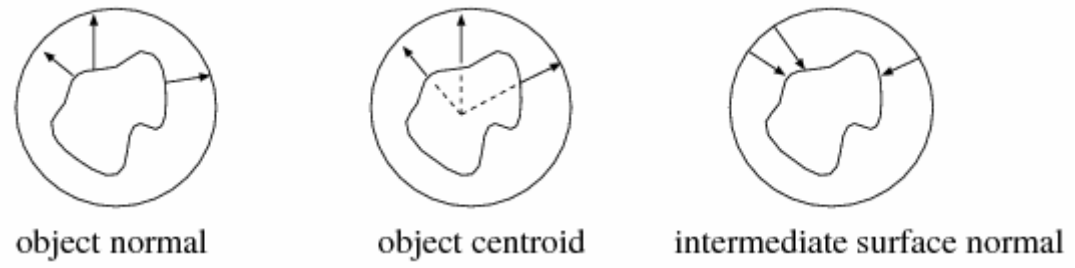
$$[\alpha, \beta, h, 1] = [x(u,v), y(u,v), z(u,v), 1] \cdot \begin{bmatrix} & & 0 \\ & \mathbf{A}_{3\times 3} & 0 \\ & & 0 \\ \mathbf{p}^{\mathrm{T}} & & 1 \end{bmatrix}^{=1},$$

$$u(x,y,z) = u(\alpha,\beta) = \frac{1}{2\pi} \cdot \arctan 2(\beta,\alpha), \quad v(x,y,z) = \frac{h}{H}. \qquad \textbf{[16]}$$

Mapování textury na obecný povrch

Toto mapování se provádí ve dvou fázích. Využívá se k tomu pomocného objektu, který zajišťuje mapování mezi jeho povrchem a texturou. Nejprve se namapuje textura na pomocný objekt, a až potom pomocí nějaké projekce dojde k mapování na cílový objekt. Pomocný objekt musí být lehce parametrizovatelný, mezi takovéto objekty se řadí

- rovinný polygon
- koule
- válec
- roviny krychle

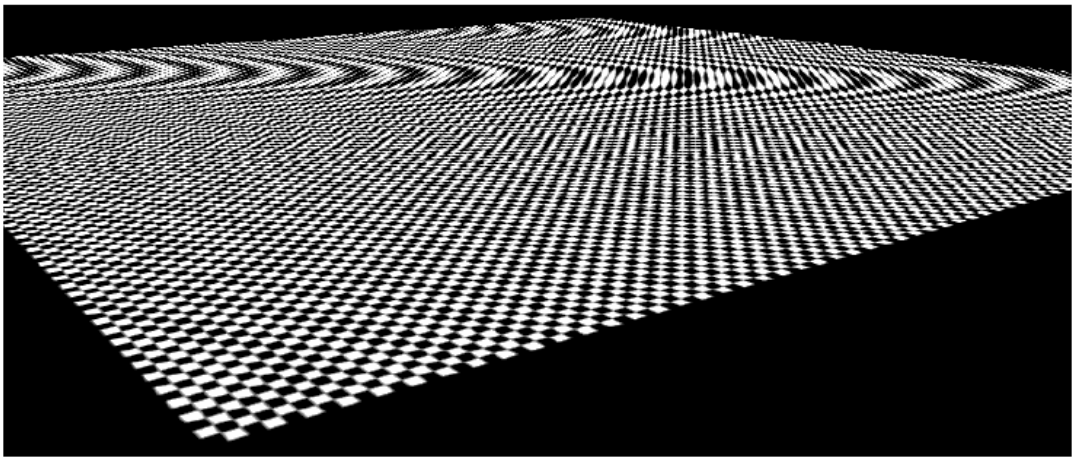


Obrázek 2-6: Příklad použití pomocné koule při mapování textury na obecný povrch. Převzato z [7]

2.3 Filtrování textur

Mapování textur stanovuje vztah mezi prostorem textury a obrazovky. Mapování může způsobovat zvětšování nebo zmenšování regionů v prostoru textury, tzn. nerovnoměrné vzorkování v prostoru textury, tím dochází k aliasingu. Aby nedocházelo k aliasingu, musel by se jeden pixel (vzorek) mapovat na jeden texel – buď musíme zvýšit vzorkovací frekvenci pixelu, nebo snížit maximální frekvenci v textuře.

K omezení aliasingu se aplikují především filtrační techniky. Od jednodušších – lineární, bilineární, bikubická interpolace, až po výpočetně náročnější, ale účinnější metody, tam patří např. MIP mapping, EWA (Elliptical weighted average) nebo Summed-area table.



Obrázek 2-7: Příklad aliasingu. Převzato z [10].

2.3.1 MIP mapping

MIP znamená „multum in parvo“, což se dá přeložit jako „mnoho v malém“. To vystihuje princip této techniky. Předem se připraví struktura podvzorkovaných verzí originální textury, zprůměrováním barev čtyř texelů jedné úrovně se vytvoří jeden texel následující menší úrovně. Výhodou je poměrně kompaktní rozměr struktury, ta je větší oproti originální textuře pouze o jednu třetinu.

Obrázek 2-8: Ukázka MIP mapy (vlevo) a možné uložení v paměti (vpravo). Převzato z [10].

K textuře uložené ve struktuře se přistupuje pomocí tří indexů – souřadnice *(u, v)* a levelu *D* (tzv. MIP level), který určuje požadované rozlišení textury. Nalezení barvy texelu v dvou-dimenzionálním mip-map poli **MxM** (dále jen **MM**) se provádí následovně:

$$\begin{aligned} R(u,v,D) &= MM\left[\left(1-2^{-D}\right)\cdot M + u\cdot 2^{-D}, \left(1-2^{-D}\right)\cdot M + v\cdot 2^{-D}\right], \\ G(u,v,D) &= MM\left[\left(1-2^{-(D+1)}\right)\cdot M + u\cdot 2^{-D}, \left(1-2^{-D}\right)\cdot M + v\cdot 2^{-D}\right], \\ B(u,v,D) &= MM\left[\left(1-2^{-D}\right)\cdot M + u\cdot 2^{-D}, \left(1-2^{-(D+1)}\right)\cdot M + v\cdot 2^{-D}\right]. \end{aligned} \quad \textbf{[17]}$$

Díky mip-map struktuře je možné počítat výslednou barvu v konstantním čase. Parametr *D* se musí získat z určitého rozmezí:

$$\begin{aligned} d &= \max\left\{\left|\left[u(x+1),v(x+1)\right]-\left[u(x),v(x)\right]\right|,\left|\left[u(y+1),v(y+1)\right]-\left[u(y),v(y)\right]\right|\right\} \\ &\approx \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial v}{\partial x}\right)^2},\sqrt{\left(\frac{\partial u}{\partial y}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2}\right\}. \end{aligned} \quad \textbf{[18]}$$

Požadovaný MIP level je:

$$D = \log_2\left(\max\left\{d,1\right\}\right). \quad \textbf{[19]}$$

Pro vylepšení se používá trilineární interpolace. Použijí se dvě nejbližší mipmapy, z nich za pomoci souřadnic *u* a *v* získáme bilineární interpolací dvě hodnoty, které se pak lineárně interpolují.

3 Vlastní implementace

Jedním z cílů této diplomové práce bylo prostudovat a implementovat vhodnou mapovací metodu, jež zohledňuje „hustotu vzorkování“. Po dohodě s vedoucím mé diplomové práce doc. Dr. Ing. Pavlem Zemčíkem byla jako vhodná metoda vybrána metoda MIP mapping. Pro implementaci mapovací metody jsem měl k dispozici již existující program pro „raytracing“ – Pray, jehož autorem je doc. Dr. Ing. Pavel Zemčík.

3.1 Program Pray

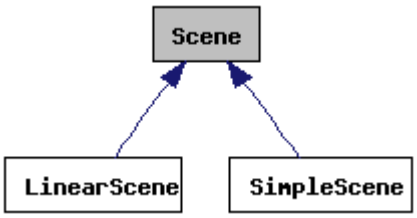
Program Pray je konzolová aplikace, implementovaná v jazyce C++, která vytváří metodou „raytracing“ obraz předdefinované scény. Vstupem programu je textový soubor s popisem scény. Výstupem je pak soubor s obrazem scény v grafickém formátu TIFF (Tagged Image File Format). Ve stejném grafickém formátu musí být i textury. Jelikož má grafický formát TIFF více variant a program Pray pracuje pouze s několika z nich, tak tuto problematiku popisuji v textové příloze A.

3.1.1 Mapování textur v programu Pray

Pro pochopení mapování textur v programu Pray stručně popíši v následující části nejdůležitější třídy, které s touto činností úzce souvisí.

třída Scene

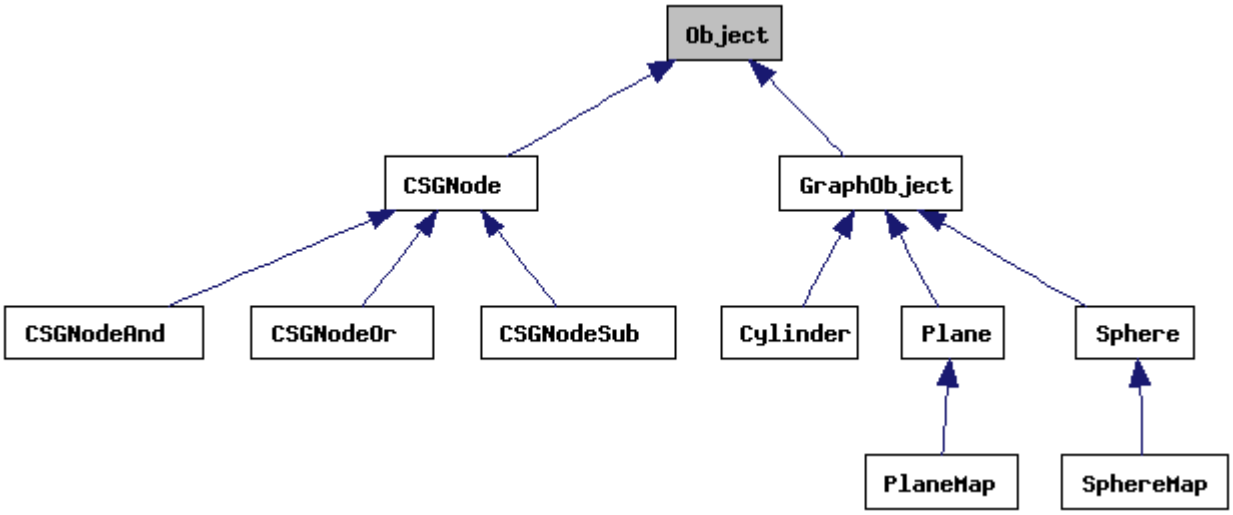
- zastupuje model scény, obsahuje informace o scéně, zjišťuje průsečíky s objekty, atd.



Obrázek 3-1: Diagram dědičnosti třídy Scene. Vygenerováno v Doxygen.

třída Object

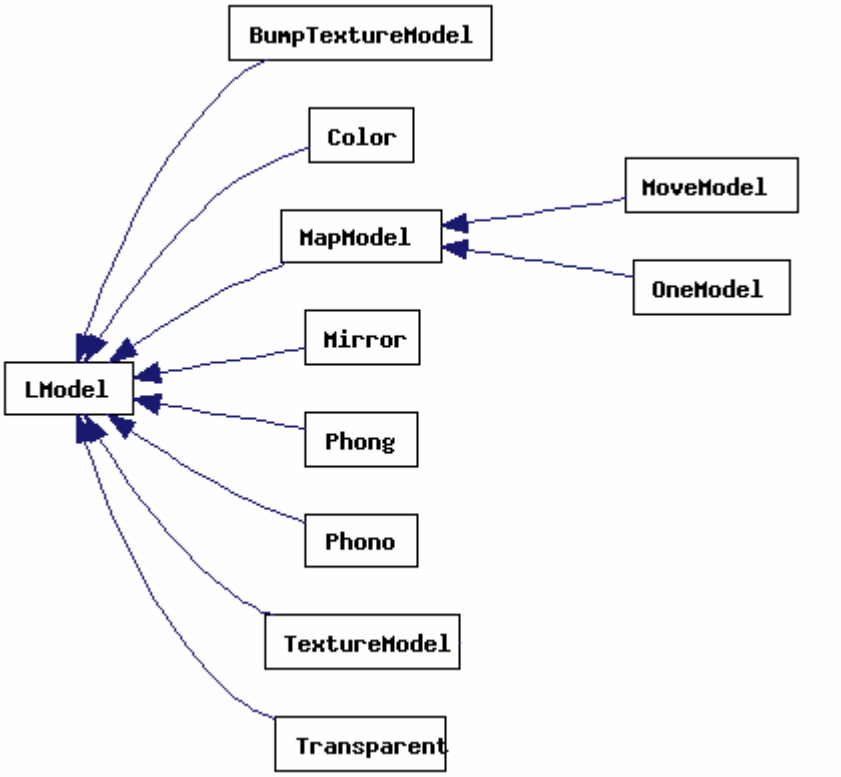
- je základní třídou objektů scény, z této třídy dědí, například, třída Plane, zastupující plochu, nebo třída Sphere, zastupující kouli, atd.
- textury lze mapovat pouze na objekty třídy PlaneMap a SphereMap
- atributem třídy GraphObject je osvětlovací model třídy LModel



Obrázek 3-2: Diagram dědičnosti třídy Object. Vygenerováno v Doxygen.

třída TextureModel

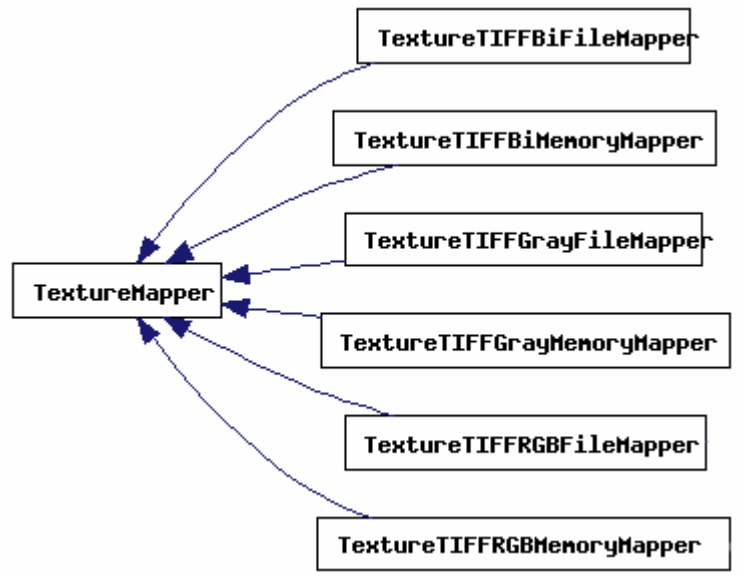
- dědí z třídy LModel, zastupuje model textury
- jeho atributy jsou další dva osvětlovací modely třídy LModel, které se podílí na výsledné barvě objektu, a TextureMapper



Obrázek 3-3: Diagram dědičnosti třídy LModel. Vygenerováno v Doxygen.

třída TextureMapper

- získává barvu z textury, která může být uložena v datové struktuře v operační paměti nebo v souboru



Obrázek 3-4: Diagram dědičnosti třídy TextureModel. Vygenerováno v Doxygen.

Třída Scene inicializuje požadavek na získání výsledné barvy ze scény. Protne-li „vržený“ paprsek nějaký objekt ve scéně, pak se pro osvětlovací model (TextureModel) daného objektu zjišťuje barva v místě průniku s paprskem. Nejdříve se transformují souřadnice bodu průniku z prostoru scény do prostoru textury a to v závislosti na typu objektu. Získané souřadnice se předloží některému potomkovi třídy TextureMapper, který s jejich pomocí získá z datové struktury nebo přímo ze souboru konkrétní barvu textury. Výsledná barva je dána smícháním barvy textury s dalšími dvěmi barvami od osvětlovacích modelů.

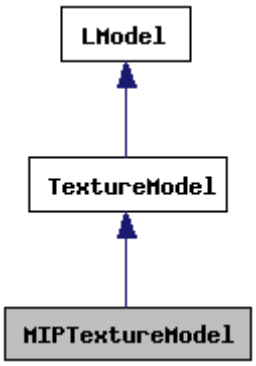
Podrobná dokumentace k programu Pray vygenerovaná pomocí aplikace Doxygen (http://www.stack.nl/~dimitri/doxygen/) se nachází na přiloženém CD. Tamtéž se nachází i detailní informace k jazyku pro popis scén, konkrétně v souboru *pray.txt*. Více o obsahu a struktuře CD přílohy je v textové příloze B.

3.2 Implementace mapování MIP textur

Pro implementaci pokročilého mapování textur jsem využil stávající strukturu tříd programu Pray, kterou jsem rozšířil o tři nové třídy.

3.2.1 MIPTextureModel

V MIPTextureModel jsou re-implementovány metody z rodičovské třídy TextureModel.

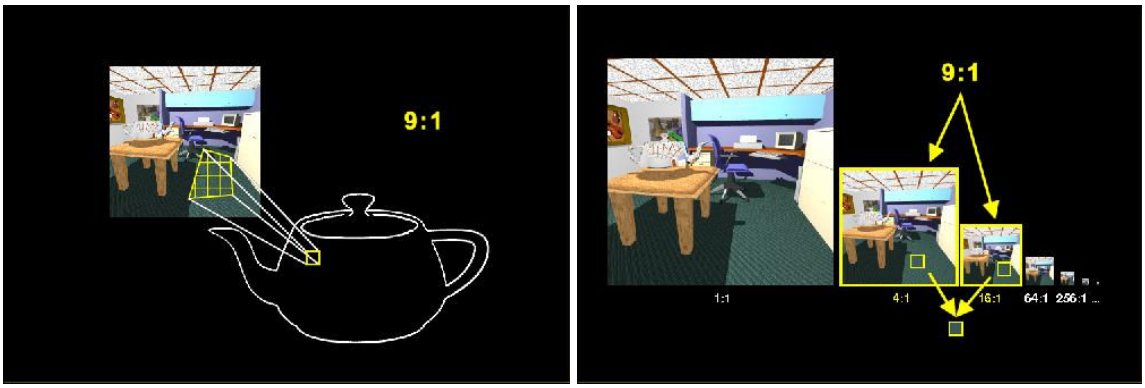


Obrázek 3-5: Diagram dědičnosti třídy MIPTextureModel. Vygenerováno v Doxygen.

Třída zajišťuje výpočet výsledné barvy v průsečíku paprsku a objektu. Aby bylo možné určit potřebnou úroveň v MIP textuře, jsou vyslány do scény další dva paprsky, které jsou odkloněny od původního paprsku o vzdálenosti *dx*, resp. *dy*. Nově získané průsečíky s objektem, jsou-li nějaké, jsou vzdáleny od původního průsečíku o *dx'*, resp. *dy'*. Ke zjištění MIP úrovně používám větší z poměrů:

$$ratio = MAX\{dx'/dx, dy'/dy\}. \qquad \textbf{[20]}$$

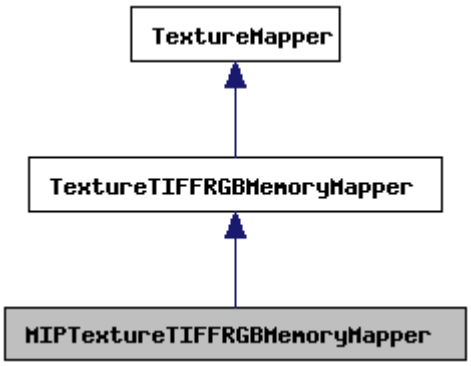
Výsledný poměr přibližně udává, na kolik texelů v textuře se mapuje jeden pixel z obrazovky (viz. Obrázek 3-6).



Obrázek 3-6: Poměr mapování prostoru textury ku prostoru scény (vlevo), výběr MIP úrovní podle poměru (vpravo). Převzato z [15].

3.2.2 MIPTextureTIFFRGBMemoryMapper

Tato třída je potomkem třídy TextureTIFFRGBMemoryMapper. Obstarává mapování do specifické struktury MIP textury uložené v operační paměti podle [u, v] souřadnic a úrovně MIP.



Obrázek 3-7: Diagram dědičnosti třídy MIPTextureTIFFRGBMemoryMapper. Vygenerováno v Doxygen.

MIP úrovně se odvodí z poměru vypočteného v metodě třídy MIPTextureModel (viz. Obrázek 3-6). MIP mapa je v paměti uložena ve dvourozměrném poli. Přístup k hodnotám dané MIP úrovně se provádí následovně:

$$c(u,v,D) = NM\left[\left(1-2^{-D}\right)\cdot N + v\cdot 2^{-D}, \left(1-2^{-D}\right)\cdot M + u\cdot 2^{-D}\right], \qquad \textbf{[21]}$$

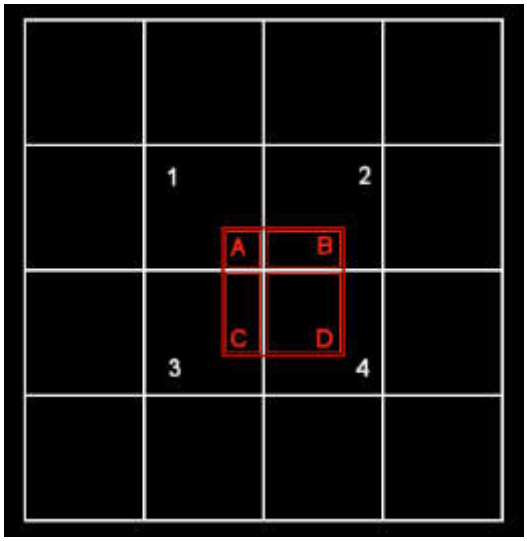
kde D je MIP úroveň, N je výška MIP textury a M její šířka.

Kvůli vylepšení výsledku pomocí tzv. trilineární interpolace se provádí výpočet nad dvěmi sousedními MIP úrovněmi. Trilineární interpolace je lineární interpolace hodnot získaných bilineárních interpolací ze dvou nejbližších MIP úrovní. Bilineární interpolace v každé úrovni se provádí podle následujícího kódu:

```
// fetch a bilinearly filtered texel
u' = u*(M/2) + 0.5;
v' = v*N + 0.5;

u1 = int(u')%(M/2);
v1 = int(v')%N;
u2 = (u1 + 1)%(M/2);
v2 = (v1 + 1)%N;

// calculate fractional parts of u and v
fracu = u' - u1;
fracv = v' - v1;
```

```
// calculate weight factors
w1 = (1 - fracu) * (1 - fracv);
w2 = fracu * (1 - fracv);
w3 = (1 - fracu) * fracv;
w4 = fracu *  fracv;

c = c1*w1 + c2*w2 + c3*w3 + c4*w4;
```

Pomocí souřadnic *u1*, *v1*, *u2* a *v2* se naleznou 4 nejbližší texely od souřadnice [*u'*, *v'*] (viz. Obrázek 3-8). Váhové faktory *w1* až *w4* určují, jakou měrou se budou dané 4 texely podílet na výsledné barvě *c*. *c1* až *c4* jsou barvy z vybraných texelů.



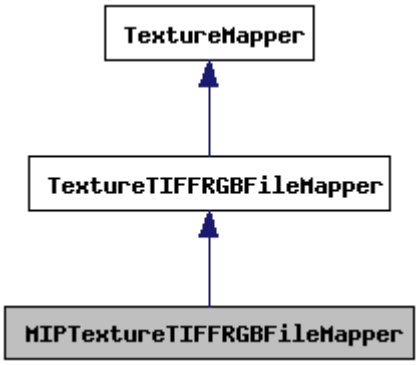
Obrázek 3-8: Červený čtverec znázorňuje polohu namapovaného texelu, očíslované texely se podílí na bilineární interpolaci, váhové faktory jsou dány obsahy čtverců A, B, C, D. Převzato z [14].

Nakonec je nad dvěmi získanými barvami provedena interpolace a navíc gamma korekce pro lepší podání barev:

$$c = \left(0.5 \cdot \left(c_1^{2.5} + c_2^{2.5}\right)\right)^{0.4}. \qquad \textbf{[22]}$$

3.2.3 MIPTextureTIFFRGBFileMapper

Tato třída se liší od třídy MIPTextureTIFFRGBMemoryMapper pouze přístupem k datům MIP textury. Textura není načtena v paměti, ale hodnoty barev jsou čteny přímo ze souboru. Tento přístup zmenší velikost běžícího programu, naopak se ale podstatně prodlouží doba běhu programu, jak bude ukázáno v následující kapitole.



Obrázek 3-9: Diagram dědičnosti třídy MIPTextureTIFFRGBFileMapper. Vygenerováno v Doxygen.

3.2.4 Mapování textury na kouli

Dále jsem přepsal metodu třídy SphereMap, která převádí souřadnice z prostoru scény na souřadnice v prostoru textury, tzn., převádí kartézské souřadnice na sférické pomocí následujících rovnic:

$$\phi = \arccos(-S_n \cdot S_p)$$

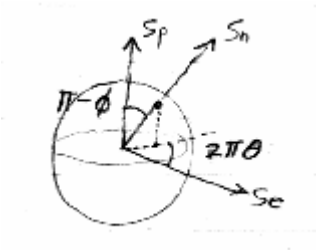
$$v = \phi / \pi\,, \qquad \textbf{[23]}$$

$$\theta = \frac{\arccos((S_e \cdot S_n)/\sin(\phi))}{2\pi}, 0 \le \theta \le 1$$

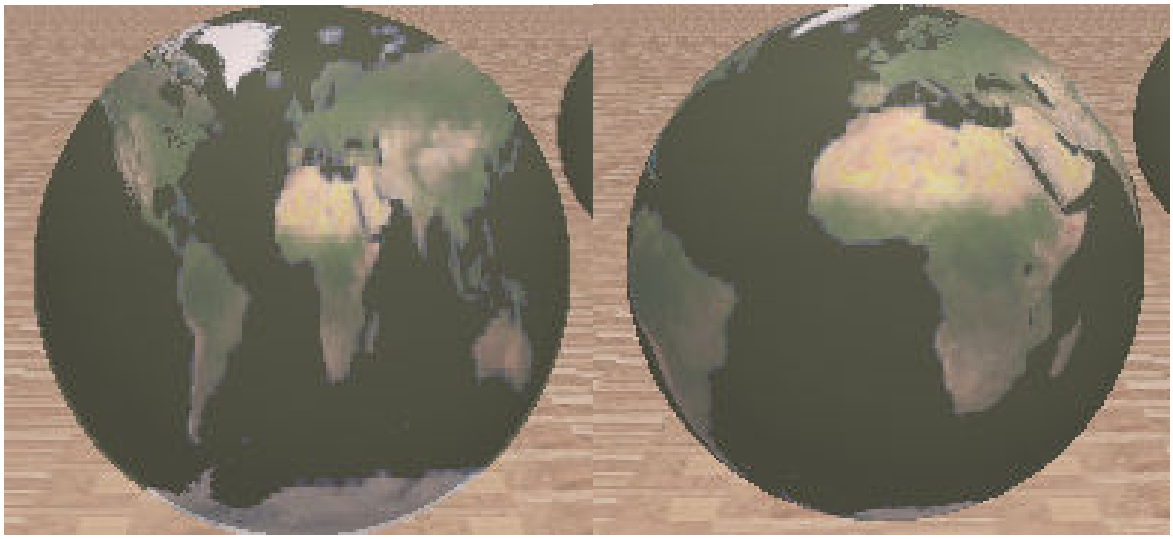
$$if\left(\left(S_p \times S_e\right) \cdot S_n\right) > 0$$

$$then \quad u = \theta \qquad \textbf{[24]}$$

$$else \quad u = 1 - \theta$$



Obrázek 3-10: Parametrizace koule. Převzato z [4].



Obrázek 3-11: Textura mapovaná na kouli před úpravou (vlevo) a po úpravě (vpravo).

4 Výsledky

V předchozích kapitolách byly popsány metody mapování textur v „ray tracingu“ a implementace. V této kapitole budou uvedeny výsledky poskytované programem Pray s implementovaným pokročilým mapováním textur. Porovnávány budou vybrané scény renderované

- s jednoduchým mapováním textur
- s mapováním pomocí MIP map, které jsem implementoval (v výsledkových tabulkách budou tyto scény označeny příponou MIP)
- s pomocí antialiasingu, který je řešen supersamplingem (ve výsledkových tabulkách budou tyto scény označeny příponou AA)

Scény budou porovnány podle časové náročnosti, dále podle počtu vržených paprsků a podle vzhledu. Ke konci ještě porovnám renderování scén užívající načítání textur z paměti (třída MIPTextureTIFFRGBMemoryMapper) proti načítání textur ze souborů (třída MIPTextureTIFFRGBFileMapper).

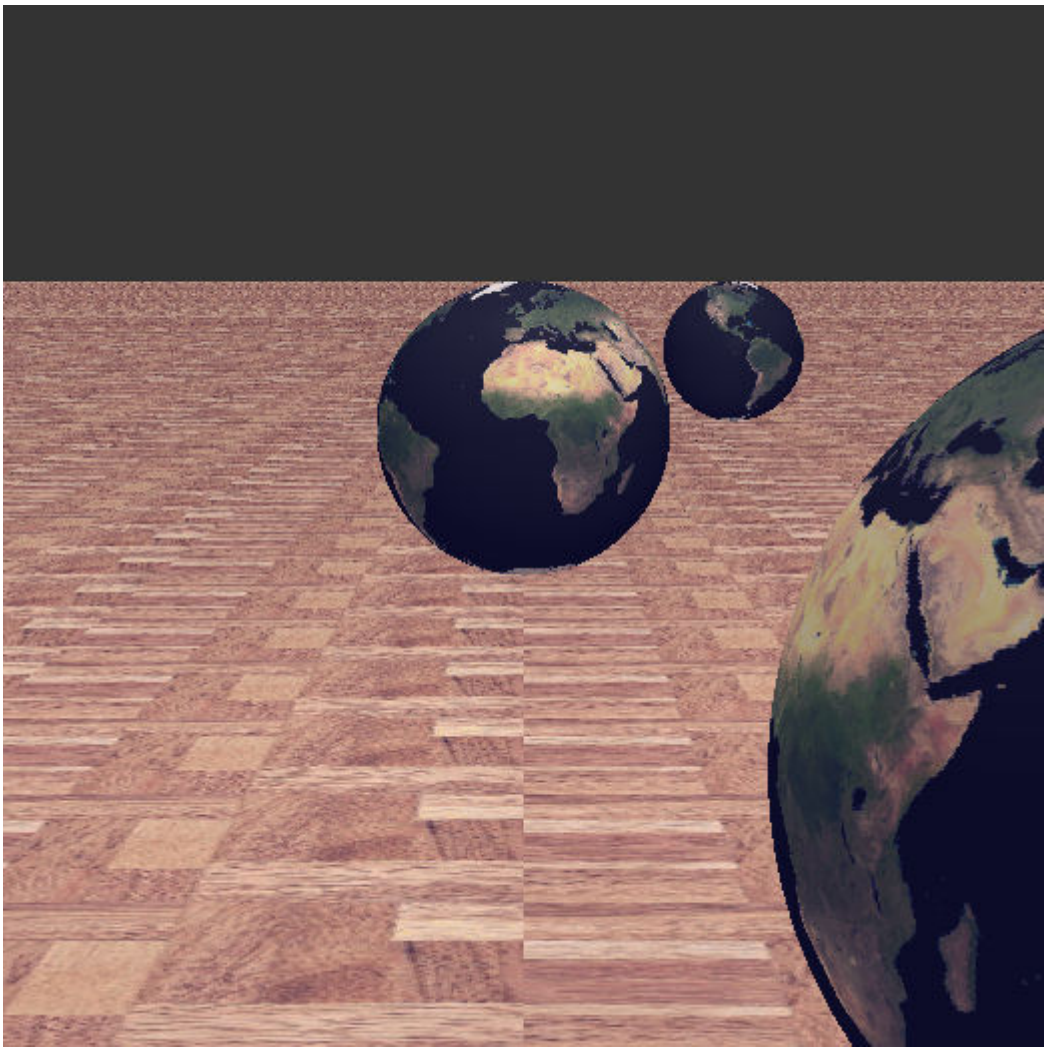
Všechny níže prezentované výsledky byly vytvořeny na notebooku ACER TravelMate 242LCe, který měl takovouto konfiguraci:

čipová sada	Intel® 852/855
procesor	Intel® Celeron 2.4GHz
operační paměť	512MB DDR SDRAM
pevný disk	30GB Ultra ATA
grafická karta	Intel® 852/855 GM
operační systém	MS WindowsXP Professional SP2

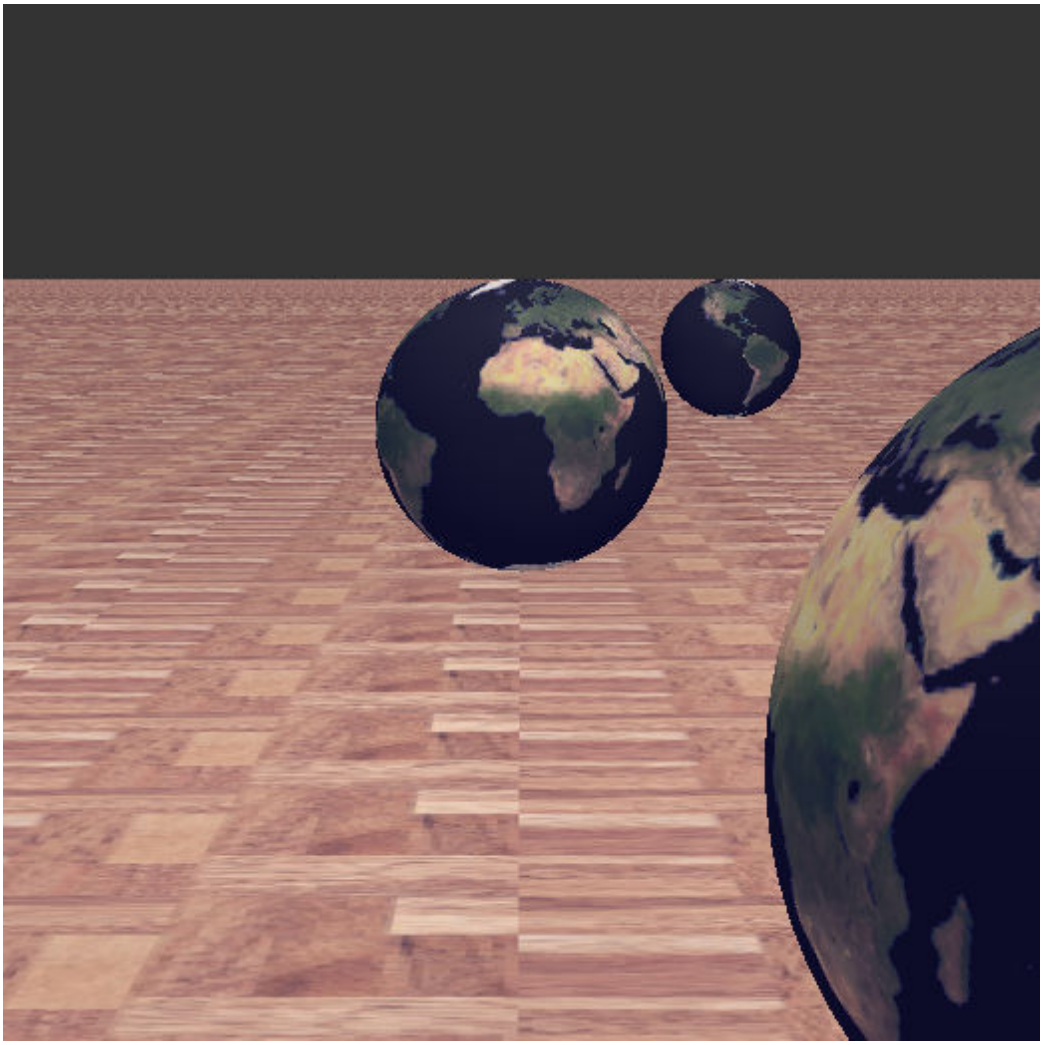
4.1 Naměřené hodnoty

Dále bude porovnáno pět scén, u prvních tří scén budou zobrazeny všechny výsledné obrazy v plném rozlišení, u dalších třech bude zobrazen náhled a obrazy v plném rozlišení budou ke shlédnutí na přiloženém CD. Za obrazy bude následovat tabulka s naměřenými hodnotami a slovní zhodnocení. Další testovací scény se nacházejí na přiloženém CD.

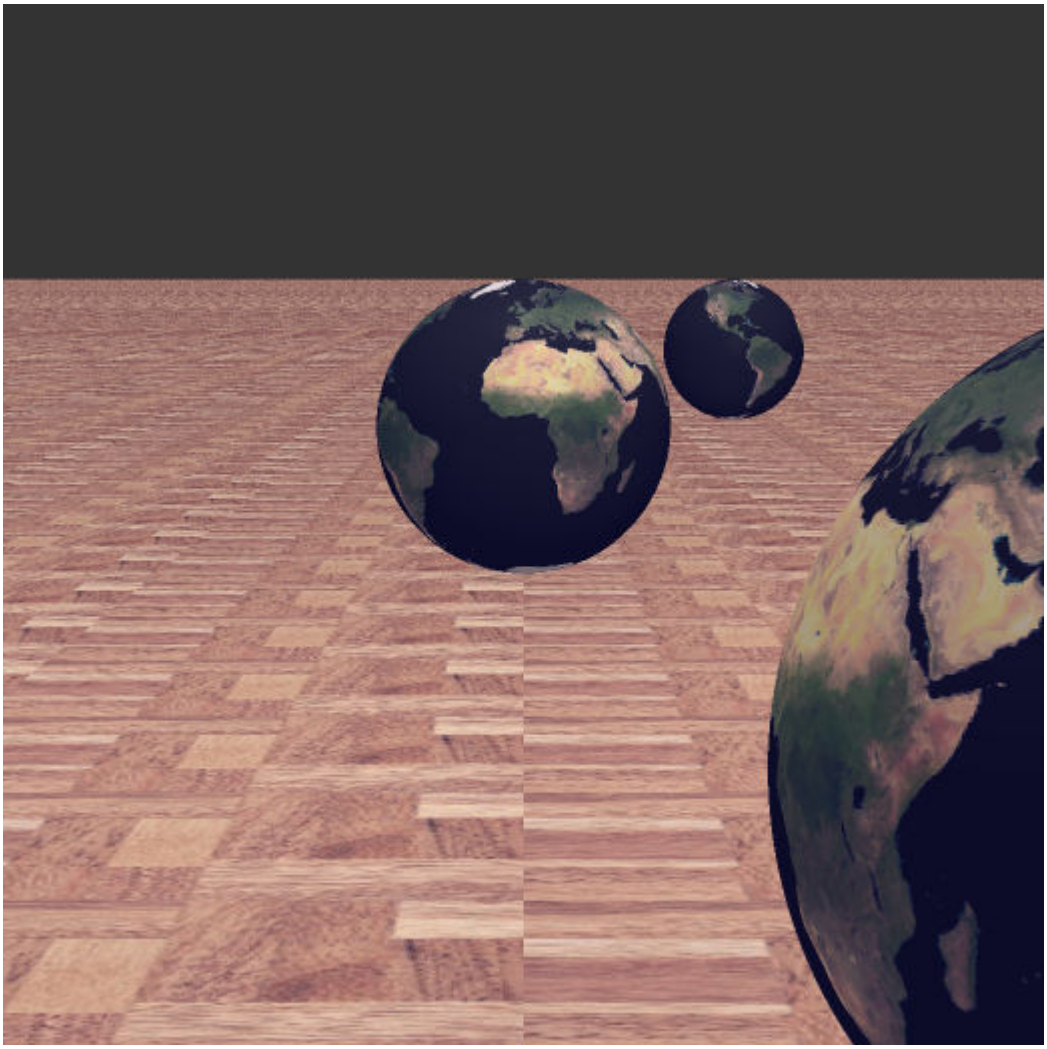
scene1



Obrázek 4-1: Jednoduché texturování.



Obrázek 4-2: MIP mapping.



Obrázek 4-3: Antialiasing – supersampling.

Rozlišení originálu: 520x520

	Počet vystřelených paprsků	*čas [s]*
scene1	888 978	3
scene1 MIP	1 287 414	5
scene1 AA	3 553 817	8

pc

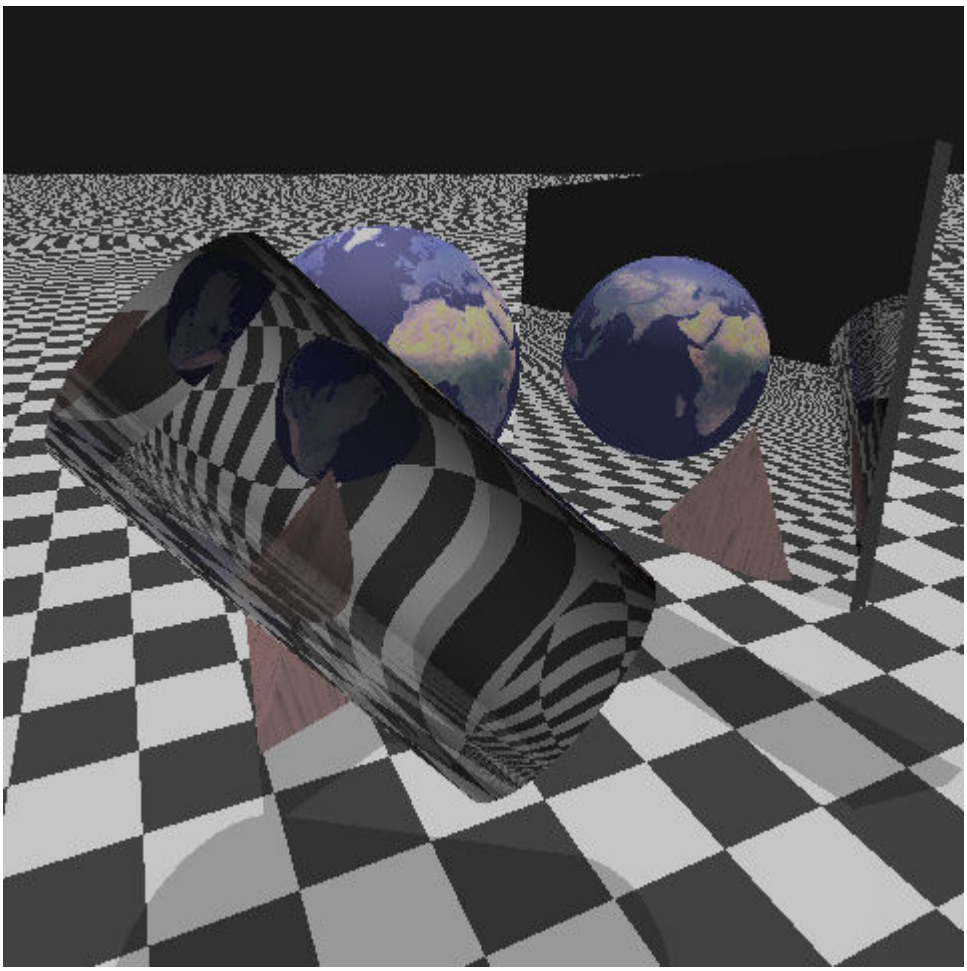
Obrázek 4-4: Jednoduché texturování

Obrázek 4-5: MIP mapping

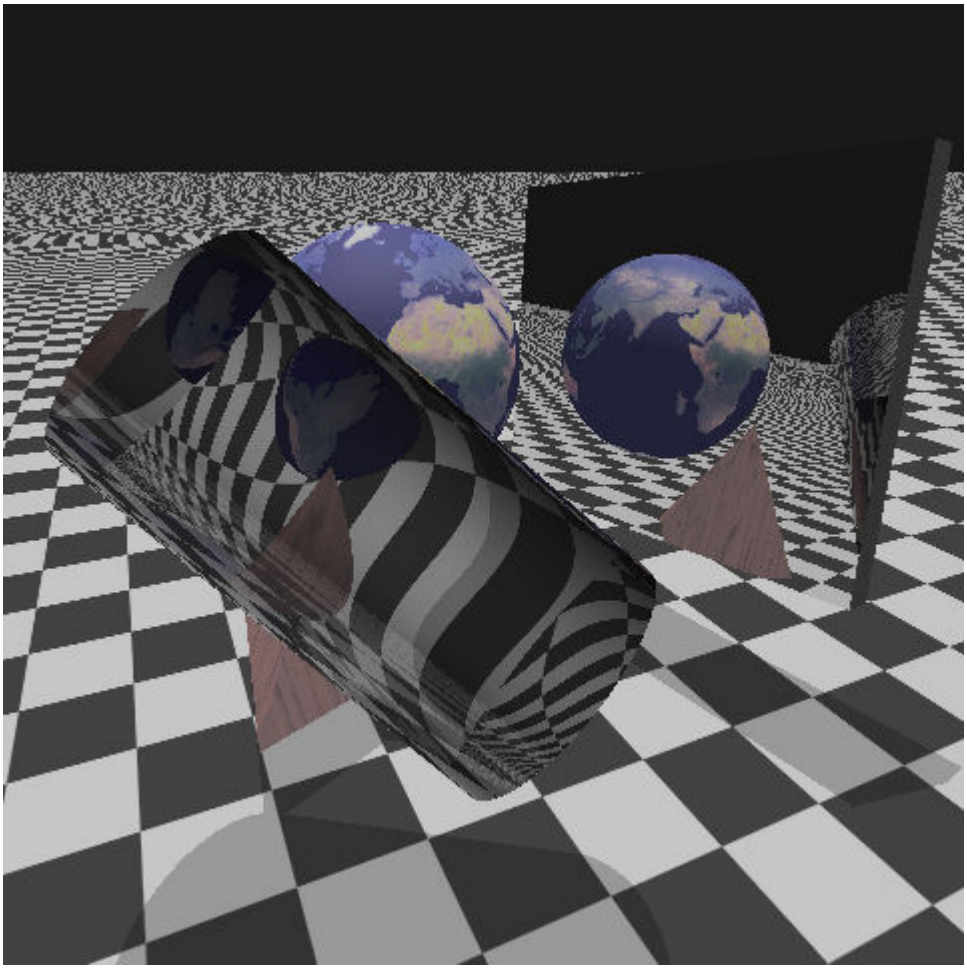
Obrázek 4-6: Antialiasing – supersampling

Rozlišení originálu: 520x520

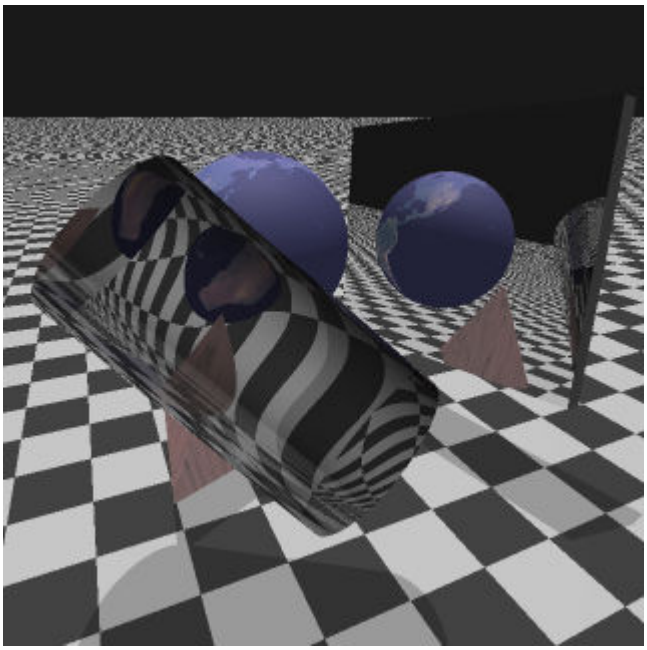
	Počet vystřelených paprsků	*čas [s]*
pc	838 814	5
pc MIP	938 570	6
pc AA	3 321 003	15



Obrázek 4-7: Jednoduché texturování



Obrázek 4-8: MIP mapping



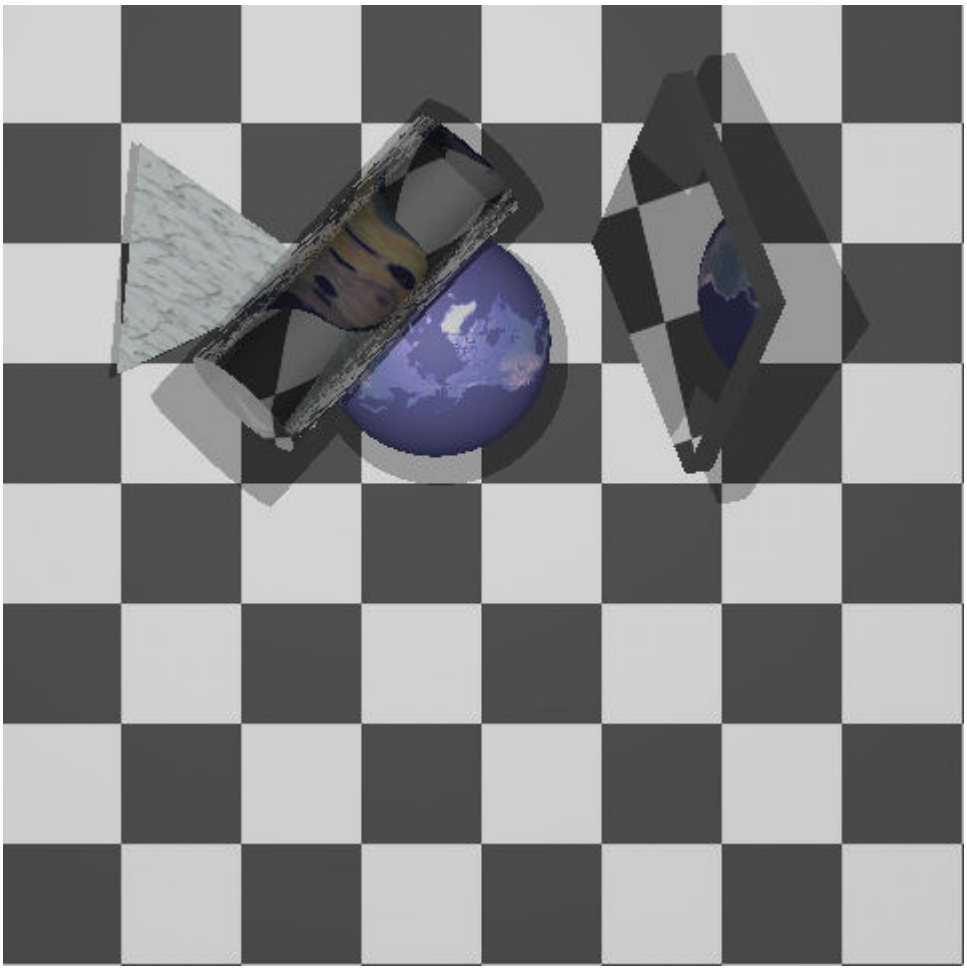
Obrázek 4-9: Antialiasing – supersampling

Rozlišení originálu: 480x480

	Počet vystřelených paprsků	*čas [s]*
izus2	605 058	4
izus2 MIP	997 517	7
izus2 AA (320x320)	1 058 430	6

Scéna renderovaná se zapnutým antialiasingem je v menším rozlišení, protože při vyšším rozlišení je nestabilní.

izus



Obrázek 4-10: MIP mapping a paralelní promítání

Rozlišení originálu: 480x480

	Počet vystřelených paprsků	*čas [s]*
izus	526 725	4
izus MIP	990 424	8
izus AA	2 106 710	12

izus1

Obrázek 4-11: MIP mapping a paralelní promítání

Rozlišení originálu: 480x480

	Počet vystřelených paprsků	*čas [s]*
izus1	731 227	5
izus1 MIP	1 182 695	9
izus1 AA	2 924 913	15

Vyrenderované scény potvrdily to, co se od každé očekávalo. Scény používající jednoduché texturování byly vytvořené nejrychleji a s nejmenším počtem použitých paprsků, ale jak je vidět především na scénách *scene1* a *izus2*, výsledek je nejslabší, hlavně na vzdálených nakloněných plochách. U MIP mappingu je viditelné výrazné zlepšení v kvalitě texturování proti jednoduchému texturování. Trilineární interpolace způsobuje, že je nanesená textura lehce rozmazaná, tím se ztrácejí drobné detaily. Z ukázek také vyplývá, že MIP mapping hodně závisí na použitých texturách. Jak je vidět na *scene1*, tam je textura parket namapovaná poměrně dobře i ve vzdálenějších částech scény.

Oproti tomu textura šachovnice použitá ve scéně *izus2* způsobuje aliasing na velké části plochy. Renderovací časy u MIP mappingu jsou podle očekávání horší než u jednoduchého texturování. Časy přibližně třikrát delší oproti jednoduchému texturování potřebuje program pro rendering scén se zapnutým antialiasingem. Výsledné obrazy produkuje nejlepší, proti MIP mappingu si textury zachovávají detaily, nedochází k rozmazávání. U některých scén je však rozdíl zanedbatelný, vezmeme-li v úvahu dobu výpočtu.

4.2 MIPTextureTIFFRGBMemoryMapper vs. MIPTextureTIFFRGBFileMapper

Pro porovnání funkčnosti těchto tříd jsem vybral scénu *pc* a *izus2*.

Scéna *pc* používá tři textury, z toho jedna není MIP:

frakt.tif 301kB

pc_boxMIP.tif 630kB

pc_keybdMIP.tif 546kB

..... 1 477kB

Scéna *izus2* používá tři MIP textury:

earth1024MIP 3072kB

tileMIP 384kB

marbleMIP 384kB

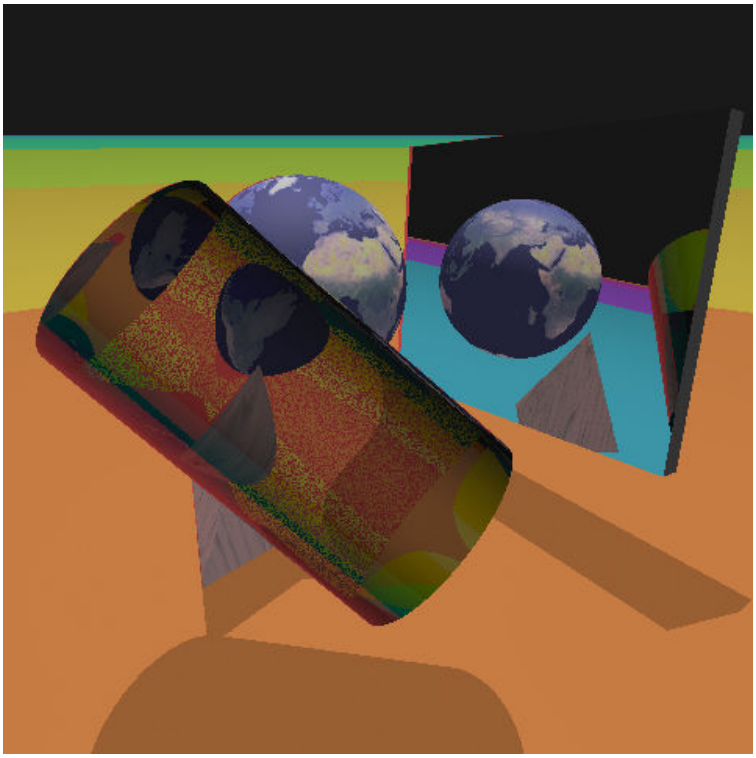
..... 3840kB

	MIPTextureTIFFRGBMemoryMapper		*MIPTextureTIFFRGBFileMapper*	
	op. paměť [kB]	čas [s]	op. paměť [kB]	čas [s]
pc	2524	6	1040	9
izus2	4888	8	1044	15

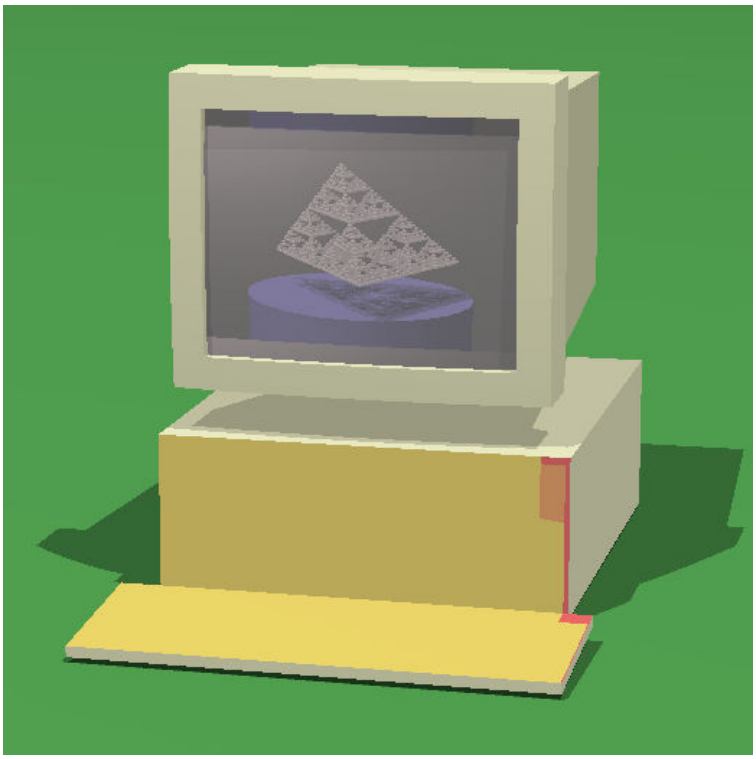
Z tabulky jsou dobře patrné paměťové nároky programu Pray, i vliv obou způsobů přístupu k texturám.

4.3 Zastoupení MIP úrovní ve scéně

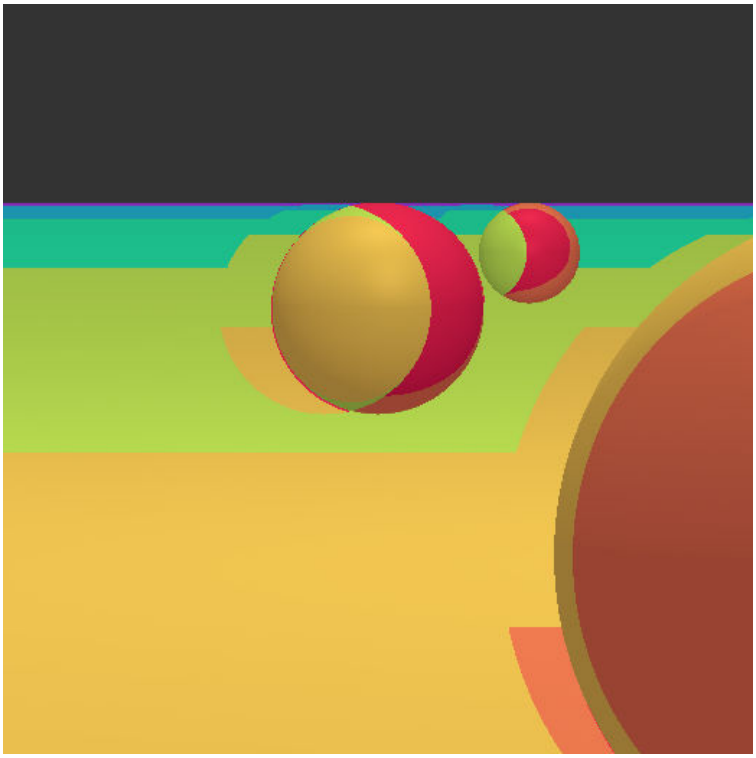
V této kapitole jsou vloženy na ukázku obrazy scén, kde je použita MIP textura, která má každou úroveň odlišenou různou barvou, aby bylo možné posoudit zastoupení MIP úrovní ve scéně.



Obrázek 4-12: Scéna izus2.



Obrázek 4-13: Scéna pc.



Obrázek 4-14: Scéna scene1color.

5 Závěr

Cílem diplomové práce bylo nastudovat metody mapování textur, které se používají v „ray tracingu“ a implementovat jednu, která zohledňuje „hustotu vzorkování“. Cíle bylo dosaženo.

V diplomové práci byly charakterizovány základní metody mapování textur využívané v „ray tracingu“. Navržena a implementována byla metoda „MIP mapping“. Tato metoda používá pro zmírnění projevu aliasingu při mapování již připravené, přefiltrované textury o několika úrovních. Tímto se ušetří čas při výpočtu, kdy se již nemusí filtrovat, tak velké oblasti textury, aby se získala výsledná barva pro jeden pixel. Pro lepší výsledky se u této metody používá trilineární interpolace.

Vybraná metoda byla implementována do existujícího programu Pray. V tomto programu již existuje jednoduchá metoda pro mapování textur, která je rychlá, ale výsledky nejsou moc pěkné. V programu je navíc možno použít antialiasingovou metodu, tzv. supersampling, kdy je každý pixel vzorkován několikrát, tím velmi vzroste kvalita výstupu, ale za cenu velkého výpočetního nárůstu. Implementovaná metoda je jen o málo pomalejší než jednoduchá metoda a výsledné obrazy se kvalitou přibližují náročnému „supersamplingu“. U implementované metody velmi záleží na kvalitě připravených textur.

I když se dnes „MIP mapping“ s trilineární interpolací stále používá, existují kvalitnější, mnohdy i rychlejší metody.

Literatura

[1] Shirley P., Morley R.K.: Realistic ray tracing – 2nd edition. A K Peters, Ltd., 2003, ISBN 1-56881-198-5.

[2] Glassner A.S.: An Introduction to Ray Tracing. Morgan Kaufmann Publishers, Inc., 2002, ISBN 0-12-286160-4.

[3] Levoy M.: Basics of ray tracing. Dokument dostupný na URL http://graphics.stanford.edu/courses/cs348b-96/basics/ (prosinec 2006).

[4] Levoy M.: Texture mapping (part 1). Dokument dostupný na URL http://graphics.stanford.edu/courses/cs348b-96/texturing/texturing1.html (prosinec 2006).

[5] Levoy M.: Texture mapping (part 2). Dokument dostupný na URL http://graphics.stanford.edu/courses/cs348b-96/texturing/texturing2.html (prosinec 2006).

[6] Perlin noise. Dokument dostupný na URL http://en.wikipedia.org/wiki/Perlin_noise (prosinec 2006).

[7] Texture mapping. Dokument dostupný na URL http://www.fsz.bme.hu/~szirmay/texture.ps.gz (prosinec 2006).

[8] Van Dam A.: Polygon shading and Texture Mapping, Brown University, Providence, listopad 2006. Dokument dostupný na URL http://www.cs.brown.edu/courses/cs123/lectures/Polygon.ppt (prosinec 2006).

[9] Durand: Shaders and Transformation, MIT CSAIL, Cambridge. Dokument dostupný na URL http://courses.csail.mit.edu/6.837/lectures/15_shading_trsfm.pdf (prosinec 2006).

[10] Durand: Texture Mapping and Other Fun Stuf, MIT CSAIL, Cambridge. Dokument dostupný na URL http://courses.csail.mit.edu/6.837/lectures/21_texture_mapping.pdf (prosinec 2006).

[11] Wolfe R.: Teaching Texture Mapping, DePaul University, 1997. Dokument dostupný na URL http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe_mapping.pdf (prosinec 2006).

[12] Přednášky z předmětu PGR, FIT VUT Brno, 2005.

[13] Chalupecký V.: Počítačová grafika Textury, FJFI ČVUT Praha, 2006. Dokument dostupný na URL http://bimbo.fjfi.cvut.cz/~chalupec/pogr/system/files/pogr-17-textury.pdf (červen 2007).

[14] Bikker Jacco: Raytracing: Tudory & Implementation Part 6, Textures, Cameras and Speed, 2005. Dokument dostupný na URL http://www.devmaster.net/articles/raytracing_series/part6.php (červen 2007).

[15] MIP-mapping : chose the best interpolation. Dokument dostupný na URL http://escience.anu.edu.au/lecture/cg/Texture/MIPmapping2.en.html (červen 2007)

Seznam příloh

Příloha 1. Příprava MIP textur

Program Pray načítá textury z grafických souborů typu TIFF (*.tiff, *.tif). U těchto souborů ale existuje velké množství variant, různé bitové hloubky barev, komprimované i nekomprimované, atd. Pro MIP mapy do programu Pray je potřeba, aby byl obsah nekomprimovaný a ve 24bitové barevné hloubce, tzn. 8bitů na R, G i B kanál.

Když jsem vytvářel první MIP mapy, narazil jsem na problém. Protože, standardní program pro kreslení v MS Windows automaticky TIFF komprimuje a ukládá i alfa kanál. Další programy co jsem zkoušel (Paint.NET, GIMP) sice nabízely uložení nekomprimovaného TIFFu, ale pouze v 32bitové barevné hloubce. Nakonec se mi osvědčilo převést TIFF na BMP formát, například v „kreslení“, čímž se odstraní alfa kanál a poté soubor převést z BMP do TIFF v grafickém programu GIMP.

Příloha 2. CD

Součástí diplomové práce je CD, které je strukturováno následovně:

/examples – vyrenderované obrazy

/pray

/src – zdrojové soubory programu, makefile

/bin – spustitelný program

/documentation – dokumentace vygenerovaná programem Doxygen

pray.txt – návod pro práci s Pray

/scenes – definice scén v souborech s příponou „ray“

/text – obsahuje zdrojový text této zprávy, plus ve formátu pdf

/textures – textury ve formátu TIFF

README – informace o obsahu CD