Semestrální práce KIV/ZPG

Pohoda na Sahaře

Tomáš Maršálek
marsalet@students.zcu.cz
14. prosince 2012

Souhlasím s vystavením této semestrální práce na stránkách katedry informatiky a výpočetní techniky a jejímu využití pro prezentaci pracoviště.

Tomáš Maršálek

1 Úvod

Pro semestrální práci jsem po osobní dohodě se cvičícím zvolil jazyk Java, poté podle vlastního výběru jeden z nejpoužívanějších wrapperů OpenGL pro Javu - **Lightweight Java Game Library (LWJGL)**.

2 Pohyb po terénu

2.1 Rychlostní vektor

Požadavkem je, aby rychlost pohybu hráče po terénu byla konstantní ve všech směrech. Postup, který je zde použit nejprve vytvoří rychlostní vektor v rovině xz, které zjistí ze stisknutých kláves WSAD. Současná pozice s tímto vektorem udává novou pozici. Nad ní je doplněna výšková souřadnice a rychlostní vektor je normalizován. Problém, který nastane, je ten, že po normalizaci vznikla chyba ve výškové složce rychlostního vektoru. Proto je souřadnice y vypočítána znovu a vektor opět normalizován. Tento postup je iterován, dokud není dosaženo přijatelné chyby. Tato metoda obecně nekonverguje, proto je počet iterací omezen konečnou hodnotou.

2.2 Fyzika

Jsou implementována dvě rozšíření nad standardním pohybem. Jedním z nich je implementace setrvačnosti po zvednutí pohybových kláves. V každém snímku je pouze rychlostní vektor přenásoben útlumovou konstantou $c, 0 < c < 1$.

Druhým rozšířením je skok do výšky. Vektor rychlosti je ve finále počítán jako součet vektoru rychlosti vypočteném podle stisknutých pohybových kláves, vektoru skoku a vektoru představujícího gravitaci.

$$\mathbf{v} = \mathbf{velocity} + \mathbf{jump} - gravityConstant \cdot t$$

Po získání vektoru rychlosti získáme pozici hráče v novém snímku ze vztahu $\mathbf{s} = \mathbf{v} \cdot t$ Čas t je zde časový rozdíl mezi snímky.

2.3 Detekce kolizí

Kolize s terénem jsou vypočteny jednoduše jako

$$y := max(y, getY(x, z))$$

3 Terén

Terén je načten z raw souboru, který reprezentuje výškovou mapu pro každý z bodů roviny xz. Tyto řídící body jsou uloženy do dvourozměrného pole bodů h. Terén je pak v programu reprezentován funkcí $getY(x, z) = y$, která používá body výškové mapy a interpoluje ty body, které leží mezi řídícími.V

3.1 getY

Pro funkci getY můžeme použít nespojitou funkci „schody“. Tj. $getY(x, z) = h(\lfloor x \rfloor, \lfloor z \rfloor)$.

3.1.1 Bilineární interpolace

Lepší řešení poskytuje použití spojité interpolace. Protože se jedná o interpolaci ve dvou rozměrech, uvažujeme interpolaci bilineární.

Její princip spočívá ve dvojité lineární interpolaci mezi čtyřmi body A, B, C, D, když máme zadány interpolační koeficienty u a v pro každý ze dvou rozměrů. Nejprve najdeme pomocné body, které leží na přímkách mezi body A a B, respektive C a D.

$$E = linearInterpolate(A, B, u)$$
$$F = linearInterpolate(C, D, u)$$

Nakonec proložíme přímku mezi těmito body, čímž podruhé použijeme lineární interpolaci - odtud název metody.

$$G = linearInterpolate(E, F, v)$$

3.1.2 Bilinearly blended patch

Pokud chceme nejen spojitou funkci, ale i spojitou derivaci, tzn. aby se na povrchu netvořily špičaté vrcholy, musíme použít funkci se spojitou první derivací. Pro dosažení vyhlazeného povrchu se nejčastěji používá bikubická nebo bilineární interpolace mezi kubickými křivkami mezi řídícími vrcholy. Zde byla zvolena bilineárně namíchaná interpolace (bilinearly blended patch)[4][9] mezi Catmull-Rom spline křivkami. Použití bikubické interpolace namísto bilineární nepřináší znatelně více hladkosti, více záleží na kontrolních křivkách.

Catmull-Rom spline je hladká interpolační křivka mezi n body, která je často používána pro svou jednoduchost. Pro výpočet používá Hermitovu formu kubické spline křivky, kde parametry derivace jsou zvoleny pro sousední body shodné, aby došlo právě ke kýžené hladkosti.

Lepším řešením by bylo použití b-spline křivky, která dosahuje spojité i druhé derivace, za cenu složitějšího výpočtu.

Bilinearly blended patch je metoda pro získání bodu, který se nachází na povrchu ohraničeném čtyřmi libovolnými křivkami. Nejprve jsou spočteny body na kontrolních křivkách dle parametrů u a v.

$$E = spline(A, B, u)$$
$$F = spline(C, D, u)$$
$$G = spline(A, C, v)$$
$$H = spline(B, D, v)$$

Tyto jsou lineárně interpolovány

$$I = linearInterpolate(E, F, v)$$
$$J = linearInterpolate(G, H, u)$$

Výsledek je součet těchto dvou lineárních interpolací, od kterých je odečtena jedna bilineární interpolace:

$$K = I + J - bilinearInterpolate(A, B, C, D, u, v)$$

3.2 Vyhlazování

Pro vykreslování je terén uložen do vertex bufferu na grafické kartě. Na základě úrovně rozdělení je terén opakovaně rozčtvrcen a nově vzniklé body určené podle zvolené interpolující funkce.

3.3 Adaptivní vyhlazování

Terén, který je zadán jako výšková mapa velikosti 128×128, je po zjemnění úrovně 3 velký 1024×1024. Vykreslování takto detailního terénu je výpočetně zbytečně složité, proto se úroveň vyhlazování (LOD - Level of Detail) určujeme na základě vzdálenosti od pozorovatele. Čím blíže, tím očekáváme detailnější terén a naopak.

3.4 Quadtree

Datová struktura, ve které je uložená reprezentace vyhlazeného povrchu je Quadtree. Její největší přínos je snadná teselace terénu, zejména tam, kde na sebe navazují rozdílné úrovně vyhlazení.

Quadtree je stromová struktura, jejíž kořenový uzel představuje celý terén jako jeden čtyřúhelník. Potomci tohoto a každého dalšího uzlu jsou čtyři čtvrtiny tohoto čtyřúhelníka, respektive čtyřúhelníka, který daný uzel představuje.

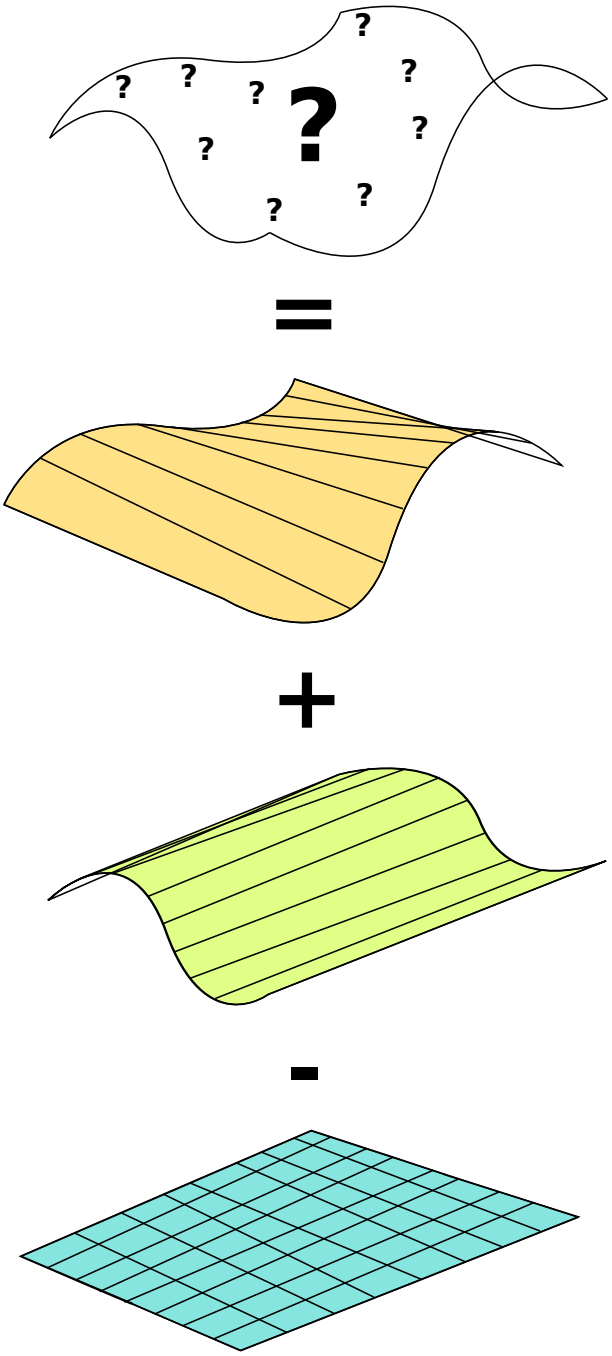
Při vykreslování každého snímku je tento strom rozložen do požadované hloubky a na konci opět složen zpátky. Obě operace jsou rekurzivní, jedna úroveň vytvoření potomků vypadá takto:

Složení je pak pouze opačná operace.

Popis a implementace Quadtree pro adaptivní vyhlazování je delailněji a lépe popsán v několika publikacích a článcích. [8]

3.5 Výpočet normály

Při ukládání vrcholů do vertex bufferu jsou zároveň spočteny normály pro každý bod. Metoda spočívá ve výpočtu gradientu takové funkce, že terén, tedy $getY(x, z)$, je jednou z jejích vrstevnic. Pro vektor gradientu platí, že je vždy kolmý na vrstevnici své křivky, což je přesně to, co potřebujeme pro normálu.

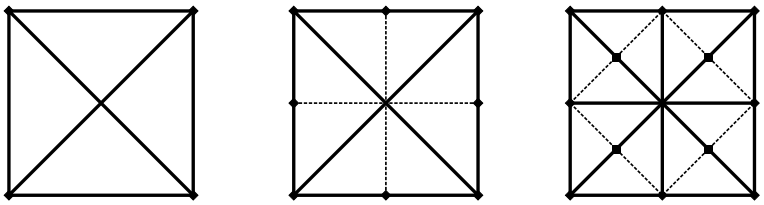


Obrázek 1: Bilinearly blended patch mezi čtyřmi kontrolními křivkami.

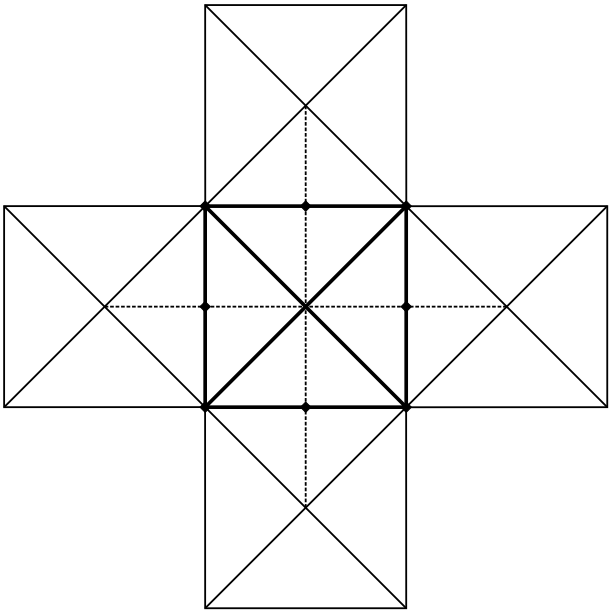
Funkce, kterou hledáme je $f(x, y, z) = getY(x, z) - y$, protože $c = getY(x, z) - y$ je obecný předpis pro vrstevnici f a pro $c = 0$ dostáváme náš terén. Gradient této funkce je jednoduše

$$\nabla f = \left[\frac{\partial getY}{\partial x}, -1, \frac{\partial getY}{\partial z}\right]^T$$

Derivaci samozřejmě spočteme numericky, například pomocí obousměrné diference.



Obrázek 2: Vytvoření potomků uzlu.



Obrázek 3: Rozložení uzlu a korekce u sousedních uzlů.

Gradient normalizujeme a máme jednotkový vektor normály pro daný bod.

4 Pohledové ořezávání

Počet trojúhelníků, které budeme chtít vykreslit nebo vyhladit pomocí quadtree, můžeme výrazně omezit tak, že vynecháme všechny uzly, které se ve výsledném snímku vůbec nezobrazí.

Provedeme vlastně clipping, tedy oříznutí elementů mimo snímek, stejně jako ho za nás provede OpenGL ve výsledném snímku. Abychom mohli tuto proceduru aplikovat, musíme získat projekční a modelview matici, se kterými v daném snímku grafická knihovna pracuje.

Pro převedení souřadnic bodu $\mathbf{b}$ z modelových do clipping souřadnic použijeme modelview $\mathbf{M}$ a projekční matici $\mathbf{P}$:

$$\begin{aligned} \mathbf{b_{world}} &= \mathbf{Mb} \\ \mathbf{b_{clip}} &= \mathbf{Pb_{world}} \\ \mathbf{b_{clip}} &= \mathbf{PMb} \end{aligned}$$

Ve skutečnosti nás zajímá matice pro převod z modelových do clipping souřadnic, tedy součin modelview a projekční matice. Její řádky se poté použijí pro konstrukci stěn komolého jehlanu (view frustum), který představuje viditelnou oblast.

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \mathbf{PMb} = \begin{pmatrix} pm_1^T \\ pm_2^T \\ pm_3^T \\ pm_4^T \end{pmatrix} \mathbf{b}$$

V clipping souřadnicích je snadné určit, zda se bod $\mathbf{b_{clip}} = [x, y, z, w]^T$ nachází uvnitř pohledové pyramidy, či nikoliv. Bod uvnitř splňuje šest nerovnic, pro každou z šesti stran komolé pyramidy.

$$\begin{aligned} -w \leq & x \leq w \\ -w \leq & y \leq w \\ -w \leq & z \leq w \end{aligned}$$

Protože ale w je reprezentován posledním řádkem $pm_4^T\mathbf{b}$ a x je $pm_1^T\mathbf{b}$, přepíšeme první nerovnici:

$$\begin{aligned} -pm_4^T\mathbf{b} &\leq pm_1^T\mathbf{b} \\ 0 &\leq pm_1^T\mathbf{b} + pm_4^T\mathbf{b} \\ 0 &\leq (pm_1^T + pm_4^T)\mathbf{b} \end{aligned}$$

Poslední rovnice představuje rovnici první roviny pohledové pyramidy. Stejným způsobem zjistíme zbylých pět.

5 Osvětlení

5.1 Plynulá intenzita světla

Máme-li den, který trvá od času 0 (00:00) do 1 (24:00), očekáváme v čase 0.5 nejvyšší intenzitu světla. Použitá funkce určující intenzitu slunce je Gaussova křivka centrovaná do bodu 0.5.

$$Intenzita \ = \ e^{-(C(t-\frac{1}{2}))^2}$$

kde C je magická konstanta, která byla nalezena po chvíli testování tak, aby byl výsledek co nejbližší skutečnosti.

Získaná intenzita se používá přímo jako intenzita hlavního zdroje světla - slunce a jako barva atmosféry.

5.2 Ambientní osvětlení

Scénu navíc osvětluje globální ambientní iluminace a dvě světla umístěná nad dvěma protilehlými rohy terénu, která vrhají realistický stín i v "noci".

6 Přehled implementovaných variací

- grafický vzhled a efekty
  - pohledové ořezávání pyramidou
  - hladký terén
  - adaptivní zjemňování terénu
- avatar, uživatelské rozhranní a pomoc hráči
  - dynamika avatara (hráče)

7 Přeložení a spuštění

Aplikace byla vyvíjena pod GNU/Linux. Použitá verze Javy je 1.6.

V kořenovém adresáři jsou dodatečně přiloženy spustitelné soubory vytvořené pomocí utility **jarsplice**, které obsahují všechny potřebné knihovny a nativní soubory.

- **ZPG.exe** pro Windows
- **ZPG.sh** pro Linux
- **ZPG.jar** pro Windows i Linux, spuštění přes příkaz java -jar

7.1 Z příkazové řádky

spuštění přiloženého archivu:

```
bin$  java -jar ZPG_A10B0632P.jar
```

7.2 Použitím Ant

Pokud je nainstalovaný Apache Ant, pak pro přeložení a spuštění stačí použít příkaz z adresáře obsahující skript *build.xml*.

```
$  ant
```

nebo

```
$  ant run
```

7.3 V Eclipse

Získání a integrace **LWJGL** pod eclipse je popsána ve videu [2], ale není třeba. Práce je odevzdána přímo jako projekt Eclipse, tedy stačí v Eclipse importovat existující projekt. Při importu se však smaže adresář bin, je třeba ho nakopírovat zpátky, jinak chybí soubor s terénem v bin/data.

8 Ovládání hry

Kamerou pohneme standardně klávesami WSAD nebo šipkami a otočíme pohybem myši. Skok je pomocí mezerníku, sprint pomocí levého shiftu. Invertování vertikálního pohledu provedeme pomocí klávesy U a přepnutí mezi drátěným a plným modelem pomocí klávesy C. Hru ukončíme stiskem klávesy ESC.

9 Závěr

Výsledkem je spíše jednoduchý základ pro hru, než hra samotná. Zaměřil jsem se především na výpočetně orientované variace úlohy. Nejzajímavější část této práce je pohled v drátěném modelu, kdy jsou vidět přechody mezi jednotlivými úrovněmi detailu.

I přesto, že se jedná o pouhý terén a pohyb po něm, jedním jednoduchým vylepšením scény by bylo otexturování terénu na základě například výšky nebo sklonu v daném bodě. Dále jsem chtěl implementovat vyhlazování pomocí kubické b-spline křivky a porovnat s hotovou Catmull-Rom křivkou.

Reference

[1] *Bill Jacobs*
OpenGL tutorial 2007 - 2012
`http://www.videotutorialsrock.com/`

[2] *TheCodingUniverse*
#1 LWJGL Workspace - LWJGL Tutorials
2011
`http://youtu.be/0v56I5UWrYY`

[3] *Wikipedia contributors*
"Triangle strip," Wikipedia, The Free Encyclopedia
(accessed October 27, 2012)
`http://en.wikipedia.org/wiki/Triangle_strip`

[4] *Wikipedia contributors*
"Coons surface," Wikipedia, The Free Encyclopedia
(accessed 14 December 2012)
`http://en.wikipedia.org/w/index.php?title=Coons_surface&oldid=486532739`

[5] *The ryg blog*
Frustum planes from the projection matrix
`http://fgiesen.wordpress.com/2012/08/31/frustum-planes-from-the-projection-matrix/`

[6] *Dion Picco - flipcode*
Frustum Culling
`http://www.flipcode.com/archives/Frustum_Culling.shtml`

[7] *skytiger*
XNA Large Terrain
http://skytiger.wordpress.com/2010/11/28/xna-large-terrain/

[8] *Dustin Horne*
XNA Terrain Tutorial
http://www.dustinhorne.com/page/XNA-Terrain-Tutorial-Table-of-Contents.aspx

[9] *David L. Finn*
MA 323 Geometric Modelling
Course Notes: Day 31
Blended and Ruled Surfaces Coons Patches
http://www.rose-hulman.edu/~finn/CCLI/Notes/day31.pdf