Vysoké učení technické v Brně

Fakulta informačních technologií

Ročníkový projekt

Pavel Černý

2004/2005

Zadání

Název:

Rendrování krajiny pomocí ROAM algoritmu

Vedoucí: Pečiva Jan, Ing., UPGM FIT VUT

Zadání:

1. Prostudujte si tématiku rendrování krajiny a otevřených virtuálních scén.
2. Nastudujte a implementujte ROAM algoritmus.
3. Proveďte měření vlastností výkonových a vizuálních vlastností tohoto algoritmu.
4. Proveďte diskuzi nad dosaženými výsledky.
5. Práci publikujte na internetu.

Kategorie: Počítačová grafika

Implementační jazyk: C++

Volně šířený software: Coin

Literatura:

- Sánchez-Crespo, Core Techniques and Algorithms in Game Programming, 2003
- Trent Polack, Focus On 3D Terrain Programming, Muska & Lipman/Premier-Trade, 2002, ISBN: 1592000282
- Open Inventor tutoriál na ROOT.CZ
- Josie Wernecke, The Inventor Mentor, Addison-Wesley Professional, 1994, ISBN: 0201624958

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Podpis:

Poděkování

Velmi děkuji ing. Janu Pečivovi za pomoc při vytváření projektu. Hlavně za ochotu při konzultacích a za podmětné nápady k tématu tvorby krajiny.

Abstrakt

Projekt se zabývá tématikou rendrování krajiny, tzv. outdoors algoritmů. Přesněji algoritmy z oblasti urychlení rendrování krajiny. Jedná se o algoritmus ROAM, který snižuje počet renderovaných trojúhelníků v závislosti na vzdálenosti od daného bodu (např. hráče ve hře) a velikosti převýšení dané krajiny. Použití tohoto algoritmu umožňuje vykreslení větší mapy při stejných nárocích na grafický procesor. Použitím tohoto algoritmu však dochází ke změně přesného tvaru krajiny, má tudíž omezené oblasti použití, jako například hry (zde jevelmi populární), nikoliv však již průlet nad krajinou měsíce, neboť by došlo ke změně zobrazení přesných dat. Krajina vzniklá pomocí těchto algoritmů pozbývá jakýchkoliv objektů, jako jsou například domy, stromy, atd. Krajina má dokonce i určitá omezení týkající se sklonu, nelze zde například vytvořit strž o úhlu větším či rovno devadesáti stupňů. Pokud chceme takovouto krajinu vytvářet musíme použít jiný algoritmus, případně algoritmus jehož vstup je výstup našeho ROAM algoritmu.

Klíčová slova

Graphics, algoritmus ROAM, outdoors, terrain programming, hightmaps, midpoint displacement, binary triangle tree, BTT, Real-time, Mark Duchaineau, 3D krajina, Open Inventor, coin, SoTriangleStripSet, SoIndexTriangleStripSet, render, FPS.

Obsah

1. Úvod

1.1Co znamená ROAM?

Přesné znění zkratky je Real-time Optimally Adapting Meshes, což lze přeložit jako optimalizace rozložení trojúhelníků v mapě prováděná v reálném čase. Jedná se o nejmocnější outdoors algoritmus – neboli algoritmus pro generování 3D krajiny, který je velmi populární v počítačových hrách, či simulacích povrchu. Byl navržen týmem pracujícím v Lawrence Livermore National Laboratory, který vedl Mark Duchaineau.

1.2 Proč používat ROAM algoritmus?

Důvod je jednoduchý: máme-li jakýmkoliv způsobem vytvořenou výškovou mapu o vyšším rozlišení (např. 512 x 512 bodů), z níž jednoduše vytvoříme síť trojúhelníků, ve které se musíme ještě pohybovat, velmi nám to zatíží grafickou kartu. ROAM algoritmus slouží k ulehčení grafické kartě od této zátěže. A to tak, že zredukuje počet trojúhelníků, které by bylo nutno grafické kartě posílat. Toto snížení odlehčí vykreslování grafiky, ale na druhou stranu zvýší zatížení procesoru, který je nucen neustále přepočítávat rozložení trojúhelníků. Existují i možnosti optimalizace, rozebírané v kapitole šesté.

2. ROAM algoritmus dnes

Tento algoritmus je dnes hojně využíván v herním průmyslu, kde jsou stále vyvíjeny náročnější scény, na jejichž vykreslení nestačí ani nejmodernější grafické akcelerátory. Musí se tedy přistoupit ke kompromisu, zachování nejvyšší kvality, ale snížení nároků na akcelerátor. Zde je to pravé místo pro ROAM algoritmus.

Popis algoritmu v literatuře je velmi obecný a popisuje pouze základní myšlenku. Konkrétní implementace je plně v rukou vývojářů neboť velmi závislá na požadovaném cíly a konkrétní technologii. Je například nutné nastavit optimálně kompromis mezi rychlostí a paměťovou náročností či mírou detailu krajiny.

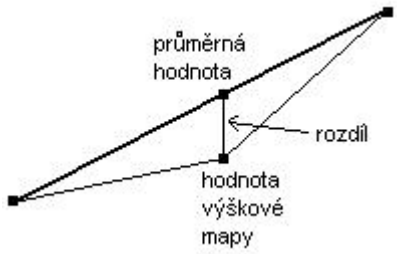
Ovšem i tento algoritmus má své meze. Jeho obor působnosti je omezen na práci s trojúhelníkovou mapou. Použitím jiných technologií pro rendrování krajiny, jako posun bodů povrchu podle textury (displacement mapping), změna povrchu beze změny jeho geometrie (bump mapping), či jiné, je tento algoritmus automaticky vyřazen z možnosti použití.

3. Obecný popis algoritmu

Tato kapitola je inspirována knihou: Focus on 3D Terrein Programming (viz. odst.Literatura)

Algoritmus ROAM se skládá ze dvou částí. **Části statické**, která se provádí ihned po získání vstupních dat (výškové mapy). Hodnoty vzniklé v této části se vypočtou pouze jednou a poté se již nemění. A **části dynamické**, která se provádí při běhu programu neustále dokola. Hodnoty této části se ve smyčce neustále mění.

V části statické jsou vytvářeny a plněny datové struktury jako Pole rozdílů, BTT , Pole souřadnic bodů, Pole indexů bodů a další. Většina těchto struktur je popsána níže a pro základní myšlenku algoritmu nejsou tak důležité, až na Pole rozdílů a BTT. V **Poli rozdílů** je pro každý bod mapy (mimo krajních bodů – nemohou zaniknout) uložena hodnota chyby, která by vznikla při jeho zániku. Což je velmi důležitá hodnota, která je často využívána v části dynamické pro rozhodnutí zda daný bod zobrazit či nikoliv. Postup výpočtu je pro názornost znázorněn na obrázku:



rozdíl = | průěrná .hodn. – reálná hodn.výškové mapy |

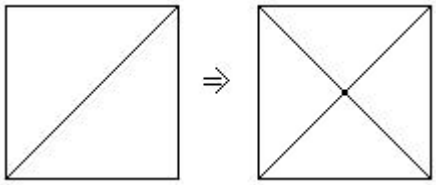
Obr.1. Výpočet hodnot v Poli rozdílů

V části dynamické, která musí nutně proběhnout co nejrychleji, je již pouze využíváno datových struktur z části statické. Jsou zde donastavovány hodnoty jednotlivých trojúhelníků, značící zda daný trojúhelník zobrazit či nikoliv. Toto rozhodnutí je klíčem celého algoritmu a je nazýváno **Rozhodovacím poměrem**. Tento poměr je složen z konstantní hodnoty Prahu (určené uživatelem), se kterou je porovnáván podíl vzdálenosti a chyby daného bodu od kamery. Je-li tento podíl menší než Prahová hodnota, je tento bod zrušen a přiléhající čtyři trojúhelníky nahrazeny pouze dvěma. Nahrazení dvěma trojúhelníky je trochu zjednodušeně řečeno. Pro nahrazování jsou definovány tři tzv**.**

Rozdělovací pravidla:

1. Je-li bod středem kosočtverce (part of a diamond). Může být provedeno dělení naprosto jednoduše (každý trojúhelník rozdělen napolovic přes přeponu i se svým

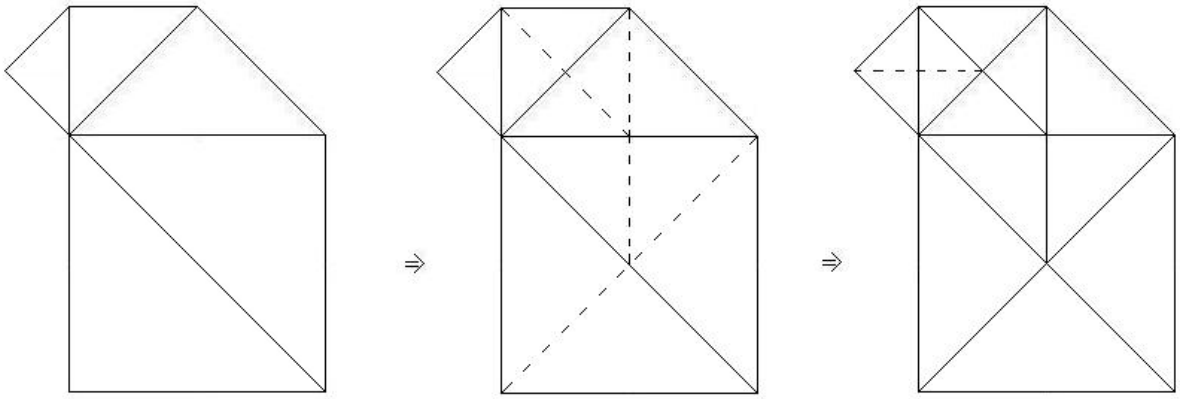
sousedem):



Obr.2. Part of a diamond

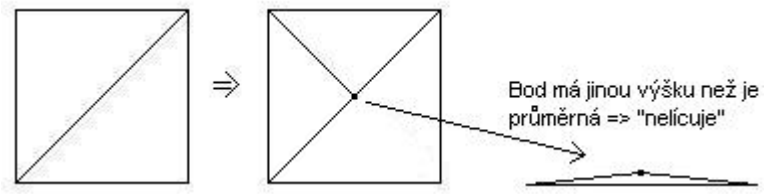
2. Je-li bod na okraji mapy (at the edge of a mesh). Je dělení analogicky s bodem 1., jak je možno (nemusím dělit souseda, když tam žádný není).

3. Není-li bod středem kosočtverce, provedu silové dělení (force-split). Nejdříve musím provést dělení souseda, abych došel opět k bodu 1. :



Obr.3. force-split

Tyto pravidla je nutné dodržovat. Zabraňují totiž vzniku některých chyb v zobrazení jako je například, že po sloučení dvou trojúhelníků vznikne mezera, kterou je vidět pod zem:



Obr.4. Ukázka chyby zobrazení

Pomocí Rozhodovacího poměru a za dodržení těchto pravidel se provede ohodnocení všech trojúhelníků celé mapy a jejich následné vykreslení. Toto ohodnocení je třeba provádět pokaždé se změnou pozice kamery, neboť dojde ke změně hodnot Rozhodovacího poměru.

4. Obecný popis implementace

4.1 Statická část

4.1.1 Výšková mapa

Získat výškovou mapu lze mnoha způsoby, například pouhým načtením obrázku v odstínech šedi (s čtvercovým rozlišením o velikosti 2^n). Tento způsob by ovšem algoritmu ROAM mnoho nevyzkoušel. Proto je v mé implementaci použit algoritmus Midpoint, který při každém spuštění vygeneruje jinou výškovou mapu, z čehož plyne vždy jiná výchozí data pro ROAM algoritmus. Tato výšková mapa je uložena do dvojrozměrného pole hmap.

4.1.2 Rozdílová mapa

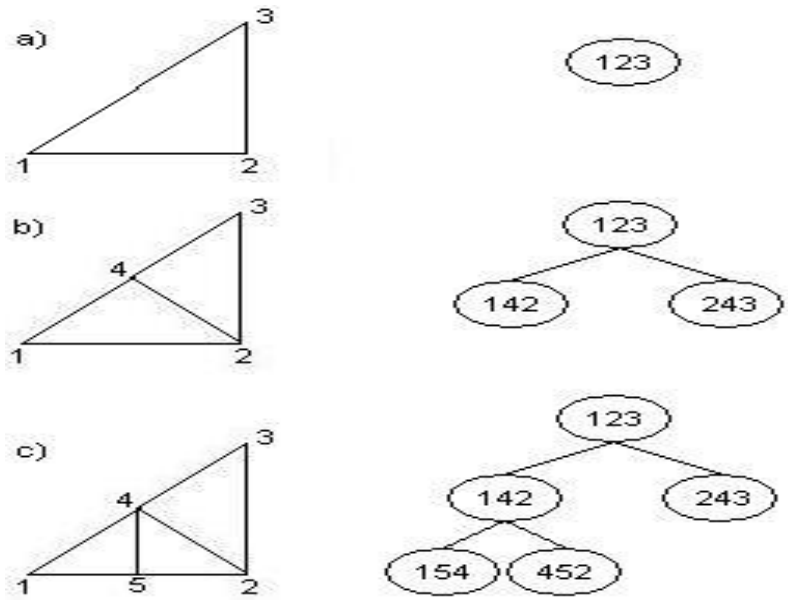
Následuje funkce, jejímž úkolem je vytvoření rozdílové mapy. Tato mapa je využívána při rozhodování zda daný trojúhelník vykreslit či ne (viz. dále). Tvorba této mapy je velmi jednoduchá a již byla popsána v kapitole třetí. Jelikož je nutné při tvorbě procházet všechny body mapy jakožto i v následující funkci Vytvor_novy_strom není táto funkce implementována samostatně, ale je její součástí.

4.1.3 Tvorba BTT

Tedy po vytvoření výškové mapy následuje funkce Vytvor_novy_strom, která provede vytvoření nejdůležitější datové struktury v celém algoritmu. Tato struktura, odborně zvaná **Binary Triangle Trees** (BTT), je jednoduchý binární strom, kde každý jeho uzel obsahuje veškeré informace o jednom trojúhelníku mapy (indexy na souřadnice všech tří bodů, odkaz na syny a na otce, informace o tom zda trojúhelník vykreslit či nikoliv, informace zda již byl trojúhelník zpracován). Počátkem celého stromu není jeden kořenový uzel, ale dva, které vznikly rozdělením čtvercové mapy na dva trojúhelníky. Tyto dva trojúhelníky mají jednoho fiktivního otce (ID = 0), který však nikde skutečně neexistuje a má pouze informační hodnotu, že jde o tzv. Root trojúhelník. Podobně je tomu i na opačné straně stromu u listů. Listy již nemají žádné syny, ale jejich ukazatele na syny mají ID = -1, což je indikátor toho, že se jedná o list.

Při vytváření této datové struktury je nutno mít informaci o tom kolik bude obsahovat celkově uzlů. Tuto informaci získáme pomocí jednoduchého vzorce: $4^{n+1} - 2$, kde n je mocnina 2 velikosti mapy. Pro názornost uvedu příklad: Mapa 32x32 má n = 5 ($2^5 = 32$) tedy velikost BTT bude 4092.

Vytváření stromu se děje ve statické části algoritmu, tedy hned po vytvoření výškové mapy. Důležité je, že při tvorbě dochází k nastavení pouze některých parametrů uzlu (těch které jsou statické). Jsou to odkazy na otce a syny, ale hlavně indexy třech bodů trojúhelníka. Zároveň s nimi se plní **Pole souřadnic bodů**, které je v odpovídajícím pořadí k daným indexům. Nenápadným, ovšem velmi důležitým prvkem vytváření stromu je dělení trojúhelníků na syny. Názorně to ukazuje následující obrázek:



Nový uzel vznikne vždy přidáním nového indexu bodu mezi první dva indexy otce (pro levý uzel) a mezi druhé dva indexy otce(pro pravý uzel)

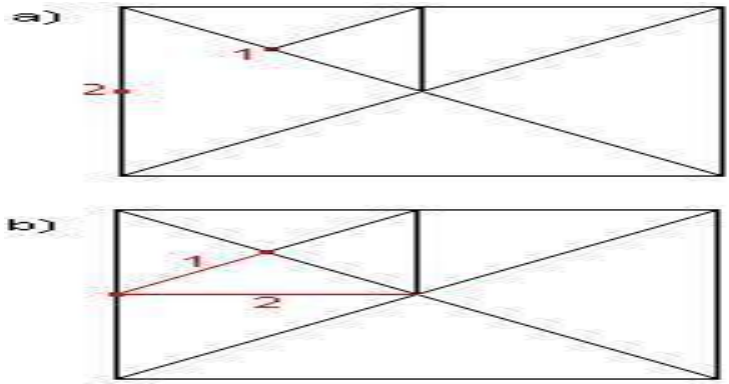
Obr.5. Ukázka dělení trojúhelníku a vytváření stromu

V bodě a) je strom tvořen jediným trojúhelníkem zadaným body 1,2,3. Ovšem pozor, důležité je pořadí bodů, které je neustále udržováno. Body jdou v pořadí, kdy vždy mezi prvním a třetím leží přepona. Rozdělíme-li trojúhelník dle bodu b) nový bod leží mezi dvěmi předchozími a tedy opět platí, že přepona je mezi prvním a třetím. Obdobně v bodu c).

Tímto je tvorba stromu myslím jasná. Připojím pouze poslední poznámku: při tvorbě stromu se postupuje predixově od Root trojúhelníků až k těm nejmenším (tedy pro připomenutí - nejprve pohled na strom z leva, poté jakoby ze spodu a nakonec z prava).

4.1.4 Korekce

Jedná se o velmi jednoduché pole do kterého se ukládají pro každý bod výškové mapy hodnoty indexů trojuhelníků, pro které je ten konkrétní bod prostředním bodem (tedy bod u pravého úhlu) a hodnoty příznaků zobrazení pro tyto trojůhelníky. Tato informace je využívána ke korekci force-split. Neboť jak bude vysvětleno v odstavci 4.2 při ohodnocování stromu je prováděno pouze dělení Part-of-diamond. Pokud by tedy nebyla provedena kontrola pomocí korekce zda není potřeba dělit také force-split docházelo by k poruchám ve zobrazení krajiny.



a) Výsledek stromu po funkci Ohodnocení BTT

b) Provedeme korekci – v bodě 1 zjistíme nesrovnalost, napravíme přímkou 1, a jelikož postupujeme rekurzivně vzhůru napravíme také bod 2 přímkou 2.

Obr.6. Ukázka funkce korekce

4.2 Dynamická část

Nyní přistupujeme k části, která probíhá dynamicky, tedy při vykreslování obrazu. Zde již velmi závisí na detailech zpracování, aby byly co možná nejjednodušší a tedy nejrychlejší. Tato část začíná velmi jednoduchou funkcí, která má za úkol nastavit všem uzlům stromu příznak zobrazení na nezobrazen a stejně tak příznak korekce jednotlivým trojúhelníkům.

4.2.1 Ohodnocení BTT

Tato část je mírně složitější. Jsou zde u jednotlivých trojúhelníků (tedy uzlů stromu) nastavovány příznaky zobrazení. Platí zde několik základních pravidel, která bych nyní jednotlivě probral:

- Pro ujasnění pojmů – Otec = nadřazený uzel(aktuální uzel vznikl z něho),
  Syn = podřazený uzel(vznikl dělením aktuálního uzlu),
  Příznak zobrazení = N – znamená trojuh. nezobrazovat,
  Příznak zobrazení = A – znamená trojuh. zobrazovat.

1. Má-li aktuální uzel příznak zobrazení = N, tak i jeho bratr musí mít příznak zobrazení = N. Toto pravidlo plyne z jednoduché logické úvahy, že buď je trojúhelník rozdělen nebo není – není nic mezi tím.
2. Má-li aktuální uzel příznak zobrazení = A, tak i jeho bratr musí mít příznak zobrazení = A. Analogicky plyne z pravidla 1.
3. Má-li aktuální uzel příznak zobrazení = A, tak i jeho otec musí mít příznak zobrazení = A. Plyne z potřeb následující funkce (Plnění pole indexu), která bude popsána níže. Toto pravidlo musí být provedeno v cyklu, neboť změnou příznaku zobrazení otce na A musím zkontrolovat příznak zobrazení otce, otce aktuálního uzlu, atd. Z čehož plyne i následující pravidlo.
4. Je-li příznak zobrazení syna = N, tak mohu zaručit, že celý tento podstrom má příznaky zobrazení = N. Neboť první změna příznaku zobrazení v této větvi na A se projeví změnou otců až k tomuto synovy. Plyne z pravidla 3.

Pomocí těchto pravidel a rozhodovacího poměru (již byl popsán v odstavci Obecný popis algoritmu) jsou tedy postupně ohodnoceny všechny uzly stromu. Tyto pravidla mají však za následek dělení pouze part-of-diamond. Dělení force-split je v BTT stromu prováděno jen velmi náročně neboť jednotlivé trojúhelníky pro toto dělení jsou od sebe velmi vzdáleny. Pro toto dělení bylo tedy zavedeno Pole korekce. Je tedy nutné doplňovat při ohodnocování stromu také jeho hodnoty (velmi jednoduché).

4.2.2 Korekce BTT

Tato funkce je velmi jednoduchá. Projdu Pole korekce pro všechny body mapy a vždy zkontroluji, zda jsou buď všechny trojúhelníky zobrazeny či nikoliv. Pokud dojde k nesouladu, je nutné zobrazit všechny trojúhelníky daného bodu. Tato změna je nutná provést v BTT i pro otce daného trojúhelníka. Jak ukázaly zkušenosti je průchod Polem korekce velmi rychlý, neboť jen velmi málo trojúhelníků potřebuje dělit force-split a tedy provádět změny v BTT.

4.2.3 Naplnění pole indexů

A následuje poslední funkce této dynamické části, která naplní pole indexů. Pole indexů je složeno z 4 * N hodnot. Kde 4 znamená tři body daného trojúhelníku a ukončovač (-1). N je počet trojúhelníků, který nyní bohužel ještě neznáme a musíme mít tedy pole indexů dynamické (vector). Plnění probíhá postupným procházením stromu, kde si všímáme hlavně listů (syn = -1). Do pole indexů dáváme všechny listy s příznakem zobrazení = A. U těch kde je příznak zobrazení = N je nutné postupovat trochu komplikovaněji: do pole indexů dáme prvního otce (postupným procházením otců aktuálního listu), který má příznak zobrazení = A (maximálně můžeme dojít k Root uzlu, který má A vždy). A poté jelikož dle pravidel víme, že jeho syni mají celé podstromy = N, postoupíme je-li to možné ještě o otce výš. Pak vejdeme do jeho pravého syna (plyne z pravidel Predixe) a dále co nejvíce vlevo až k listu, kde pokračujeme dalším vyhodnocováním. Tak projdeme opět celý strom a je hotovo. Pole indexů je naplněno ukazateli na body jednotlivých trojúhelníků.

4.3 Poznámky

Postup ohodnocování a plnění indexů tak jak je popsáno v odstavci 4.2 odpovídá první rozvaze a implementaci ROAM algoritmu. V kapitole 5 jsou popsány změny metod provádění ohodnocování. Ovšem pro uvedení do problému a objasnění základních principů je tato první rozvaha velmi vhodná.

Použití algoritmu Midpoint displacement není nijak závazné a je to pouze má volba, klidně jej lze nahradit Fault-formation algoritmem, particle deposition, či libovolným jiným. Jen pro úplnost dokládám jednoduché vysvětlení funkčnosti algoritmu Midpoint. Pracuje na principu postupného zvyšování detailů výškové mapy. Na počátku máme pouze čtyři krajní body mapy. Klasicky zjistíme průměrnou výšku prostředního bodu (např. výšky krajních děleno 4), a přičtením náhodné hodnoty (±) tuto výšku pozměníme. Obdobně změníme prostřední body hran čtverce. Rekurzivním opakováním tohoto algoritmu (každá rekurze = zdvojnásobení detailu) dostaneme výškovou mapu. Důležitá hodnota je konstanta drsnosti, určující míru zvlnění krajiny.

Při tvorbě algoritmu jsem měl na výběr dvě cesty zobrazování, které Open Inventor podporuje: První – SoTriangleStripSet posílá grafické kartě vrcholy tak jak leží v paměti za sebou. Tento přístup je při implementaci trochu jednodušší. Výsledek však zabírá více místa v paměti a na některých kartách je i pomalejší. Druhá cesta, kterou jsem si vybral já, je pomocí – SoIndexTriangleStripSet, kde každý vertex (bod) definujeme pouze jednou a poté jej již jen indexujeme. Pro porovnání obou metod bych odkázal na server root.cz a jeho tutoriál o knihovně Open Inventor, konkrétně díl šestý.

5. Průběh implementace, měření

V průběhu implementace jsem dospěl k mnoha rozličným metodám řešení daného problému. Jak již bylo uvedeno výše, je popis algoritmu v literatuře velmi obecný a zůstává tedy mnoho prostoru pro individuální návrh implementace. Uvedu zde několik možných cest implementace, v tom pořadí v jakém byly mnou implementovány, přičemž vycházím z Obecného popisu implementace, tak jak je uveden výše.

Měření FPS jsou pouze orientační. Jde o průměrnou hodnotu pokusů, vždy se stejnými parametry. Pokusy jsou velmi závislé na vzdálenosti kamery od mapy, konkrétní mapě, rychlosti pohybu s mapou, atd.

Zkratky parametrů uváděných při měření:

M: NxN kde N udává TERRAIN_DIMENSION tedy velikost mapy

PZ: M značí PRAH_ZOBRAZENI, tedy práh v Rozhodovacím poměru

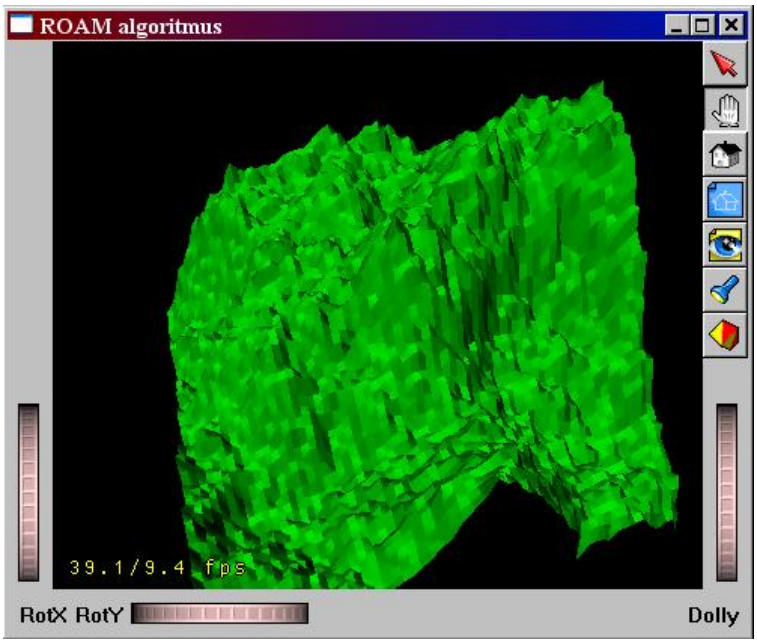
PPK: L udává PRAH_POSUNU_KAMERY

THQ: I je TERRAIN_HEIGHT_Q tedy maximální výška a hloubka terénu

Parametry vykreslování nastavitelné v programu jsou popsány v poslední kapitole Přílohy. Průběh mření a jeho hardwarová podpora jsou uvedeny na konci kapitoly (5.5).

Pro porovnání uvádím hodnoty FPS pro zobrazování bez ROAM algoritmu:

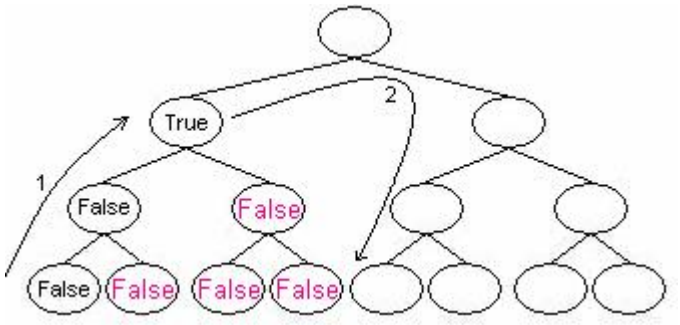
- **M:** 64x64, **THQ:** 120 => **FPS** = 28
- **M:** 128x128, **THQ:** 120 => **FPS** = 10,5
- **M:** 257x257, **THQ:** 120 => **FPS** = 2



Obr.7. Zobrazení krajiny bez ROAM algoritmu

5.1 Procházení stromu od listů

Tato metoda procházení stromu by se mohla zdát jako velmi pomalá, neboť procházíme uzly, z nichž žádný nemusí být zobrazován (pokud je kamera dostatečně daleko). Pravdou je, že je velice jednoduše implementovatelná. Jednoduše začnu procházení na nejlevějším listu a pokud má být zobrazen, tak jej zobrazím a pokud ne, tak postoupím k jeho otci atd. Jakmile takto vystoupám do určité úrovně, až k uzlu, který se má zobrazit, tak jej zobrazím a zanořím se v pravém podstromu do nejlevějšího listu, čímž přeskočím všechny listy, které jsou pod právě zobrazeným uzlem (jsou určitě nezobrazeny).



Šipka 1 ukazuje postupné ohodnocování uzlů, až dospěji k uzlu s hodnotou Zobraz=True, zde je již jasné, že uzly pod tímto mají hodnotu Zobraz=False (uzly zobrazeny červeně), následuje posun ohodnocování na další list – zobrazený šipkou 2.

Obr. 8. Ukázka výběru následujícího uzlu

Jako další urychlení procházení jsem využil znalosti toho, že musí být zobrazeni zároveň oba bratři. Tedy vždy tisknu při jednom vyhodnocení oba bratry a posunu se následně až za ně.

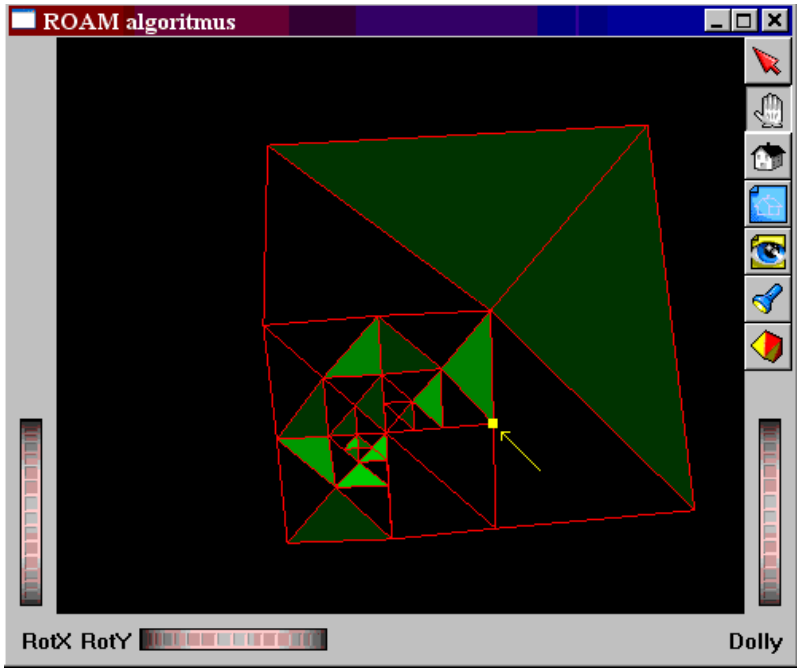
Měření FPS:

- **M:** 64x64, **PZ:** 100, **THQ:** 40, **PPK** neimplementován => **FPS** = 33
- **M:** 128x128, **PZ:** 100, **THQ:** 40, **PPK** neimplementován => **FPS** = 14
- **M:** 64x64, **PZ:** 200, **THQ:** 40, **PPK** neimplementován => **FPS** = 15
- **M:** 128x128, **PZ:** 200, **THQ:** 40, **PPK** neimplementován => **FPS** = 3,5

Zvýšení PZ při stejné pozici kamery, tak jak je popsána v odstavci (5.5), má za následek to, že je krajina celkově vykreslována ve větším detailu, což má za následek pokles FPS.

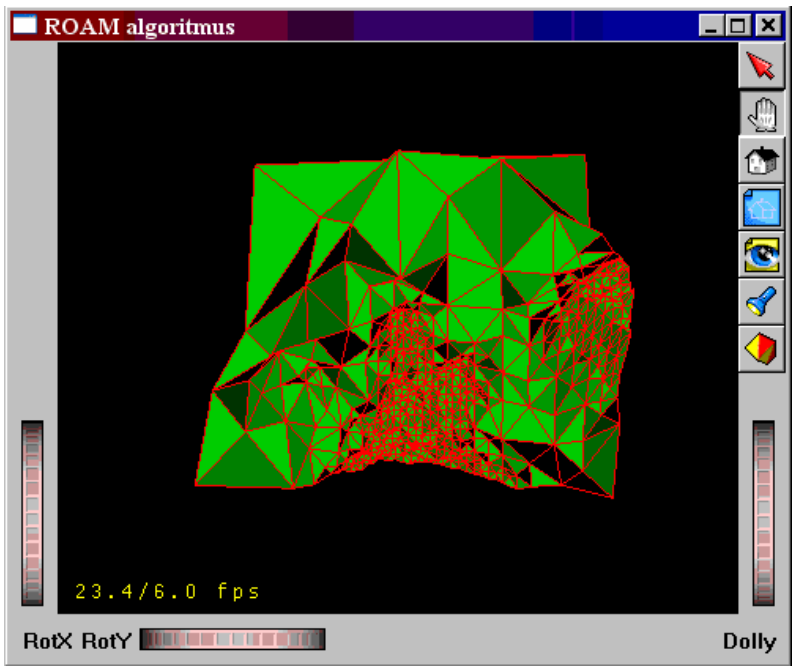
Jak je ovšem patrno při porovnání hodnot FPS bez ROAM a s ROAM procházeným od listu, je jasně vidět, že tato metoda nepřináší dostatečné urychlení vykreslování a je tedy potřeba nějaké další optimalizační metody.

Ukázky výsledků:



Šipka ukazuje na naprosto jasnou chybu zobrazení způsobenou neimplementováním force-split dělení

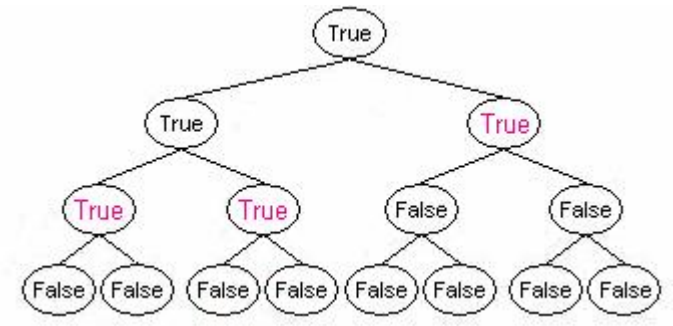
Obr. 9. Bez Pole korekce může docházet k chybám zobrazení



Obr. 10. Zde již je implementováno force-split dělení, nedochází tedy k chybám zobrazení

5.2 Procházení stromu od kořene

Implementace procházení stromu od kořene teoreticky i prakticky mnoho urychlení nepřinese, nicméně velmi zprůhlední ohodnocování stromu a také přinese možné vylepšení do budoucna, jako je dogenerovávání stromu při běhu programu, což je při prohledávání od listů obtížnější. Při této metodě procházíme strom do hloubky (max. k listu), až do doby kdy se změní příznak zobrazení z true na false. Vznikne tedy taková čepička stromu, ve které jsou všechny uzly pro zobrazení. Zobrazují se uzly na hraně této čepičky, tedy ty jejichž synové mají příznak zobrazení false.



Uzly tvořící hranici čepičky, tedy ty které je třeba zobrazit jsou zobrazeny červeně

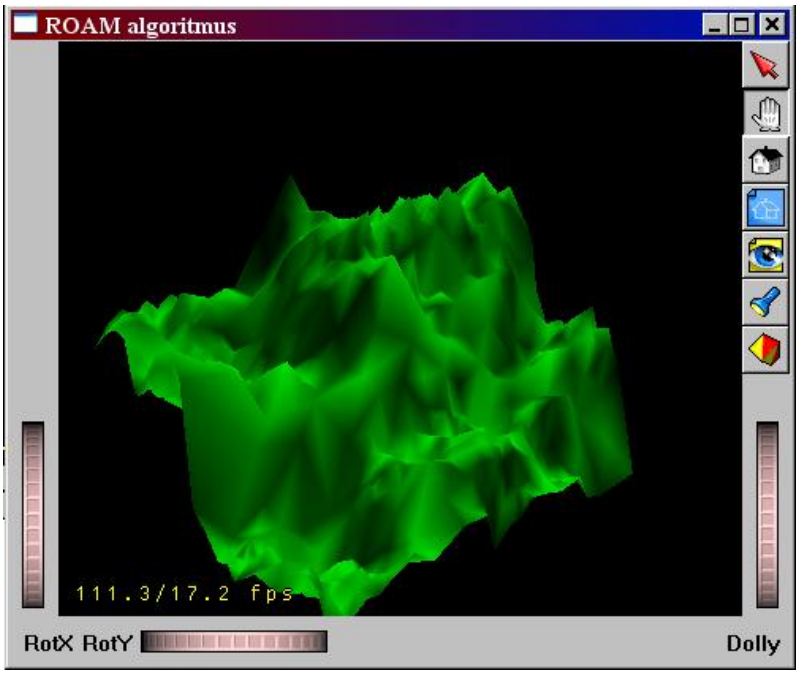
Obr. 11. Strom ohodnocen od kořene

Měření FPS:

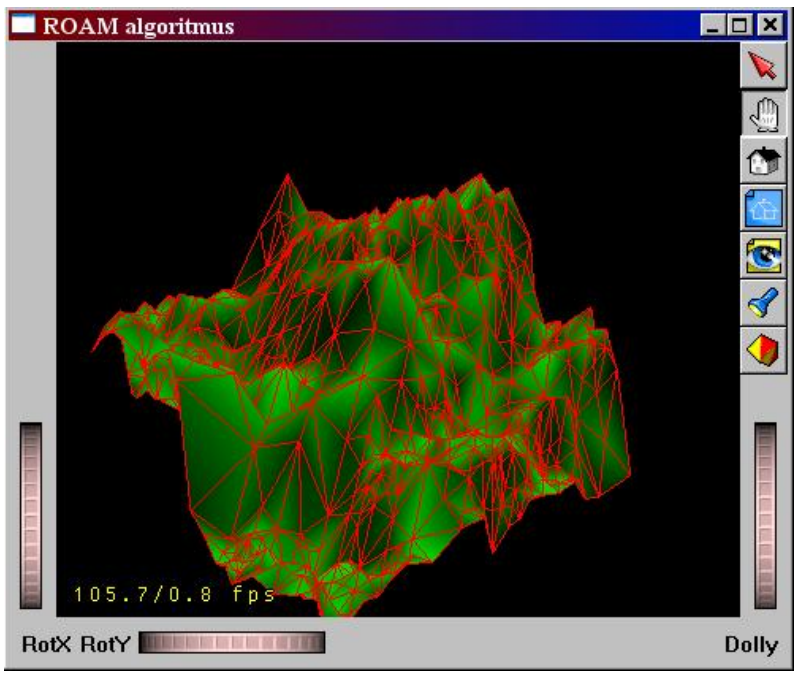
- **M:** 128x128, **PZ:** 100, **THQ:** 40, **PPK** neimplementován => **FPS** = 42
- **M:** 256x256, **PZ:** 100, **THQ:** 40, **PPK** neimplementován => **FPS** = 32
- **M:** 128x128, **PZ:** 200, **THQ:** 40, **PPK** neimplementován => **FPS** = 22
- **M:** 256x256, **PZ:** 200, **THQ:** 40, **PPK** neimplementován => **FPS** = 29
- **M:** 128x128, **PZ:** 200, **THQ:** 140, **PPK** neimplementován => **FPS** = 10
- **M:** 256x256, **PZ:** 200, **THQ:** 140, **PPK** neimplementován => **FPS** = 10

Z výsledků měření plyne již dříve očekávaný závěr. Jelikož Rozhodovací poměr ovlivňuje jak vzdálenost daného bud od kamery tak také hodnoty z rozdílové mapy, je jasný výsledek, že po zvýšení PZ poklesne FPS. Také zvýšením THQ, které má za následek změnu hodnot Rozdílové mapy, provede pokles FLP.

Ukázky výsledků:



Obr. 12. Pro zkrášlení krajiny jsem provedl dopočet normál, čímž odstraním ostré přechody mezi trojúhelníky (dříve vypočítával normály OpenInventor sám).



Obr. 13. Stejný obrázek jako předchozí, pouze jsem zobrazil rozložení trojúhelníků

5.3 Změna vytváření stromu

Zde jsem pouze pozměnil myšlenku, že nový bod, který je ukládán do SoCoordninate není vypočítáván postupně při vytváření stromu, kde se muselo testovat zda již byl uložen dříve, ale je uložen před vytvářením stromu a při tvorbě stromu je již pouze pomocí rovnice určen index kolikátý bod to je (z důvodů indexování). Tato jednoduchá změna měla za následek ohromné urychlení vytváření stromu.

Pro zajímavost uvádím časy pro mapu velikosti 128 x 128 bodů:

- ukládání ve stromu = 6.84 s
- ukládání před tvorbou stromu = 0.03 s

5.4 Ohodnocení stromu jen při větším posunu kamery

Jde o metodu, která provede urychlení vykreslování pouze v případě, že se kamera pohybuje pomalu. Využije se jedno ohodnocení stromu což má za následek zvýšení FPS. Z tesů vyplývá, že se projeví velmi příznivě, zvláště v situacích, kdy je kamera prakticky na místě a pouze se otáčí (hráč se rozhlíží po krajině).

Měření FPS:

- **M:** 128x128, **PZ:** 200, **THQ:** 100, **PPK** 5 => **FPS** = 22 (pomalá rotace)
- **M:** 128x128, **PZ:** 200, **THQ:** 100, **PPK** 5 => **FPS** = 15 (rychlá rotace)
- **M:** 128x128, **PZ:** 200, **THQ:** 100, **PPK** 50 => **FPS** = 25 (pomalá rotace)
- **M:** 128x128, **PZ:** 200, **THQ:** 100, **PPK** 50 => **FPS** = 20 (rychlá rotace)
- **M:** 256x256, **PZ:** 200, **THQ:** 100, **PPK** 5 => **FPS** = 42 (pomalá rotace)
- **M:** 256x256, **PZ:** 200, **THQ:** 100, **PPK** 5 => **FPS** = 22 (rychlá rotace)
- **M:** 256x256, **PZ:** 200, **THQ:** 100, **PPK** 50 => **FPS** = 55 (pomalá rotace)
- **M:** 256x256, **PZ:** 200, **THQ:** 100, **PPK** 50 => **FPS** = 44 (rychlá rotace)

Při měření FPS u této Metody platí všechna předešlá omezení na přesnost plus také velký vliv na rychlost otáčení krajiny při měření. Jelikož nemám žádný nástroj pro měření rychlosti otáčení krajiny, musel jsem se omezit na označení rychlá a pomalá rotace.

Při porovnání těchto hodnot FPS s hodnotami FPS bez ROAM algoritmu jsou již jasně viditelné výhody tohoto algoritmu. Např. M: 256x256 bez ROAM je rovno 2, zatímco s ROAM je v rozmezí 22 až 55 FPS.

5.5 Poznámky

Měření jsou prováděna na notebooku UMAX 575t. Sestava obsahuje Celeron 2 GHz, RAM DDR 256 MB sdílených s garfickou kartou Sis 650 která má 30MB.

VZDALENOST_BODU je ve všech měřeních nastavena na hodnotu 1.
Měření provádím pro pozici kamery umístněnou tak, aby byly její okraje umístněny na hranici zobrazení. Poté krajinu pomalu roztočím a pozoruji změny FPS. Jako měřenou hodnotu beru její průměr.

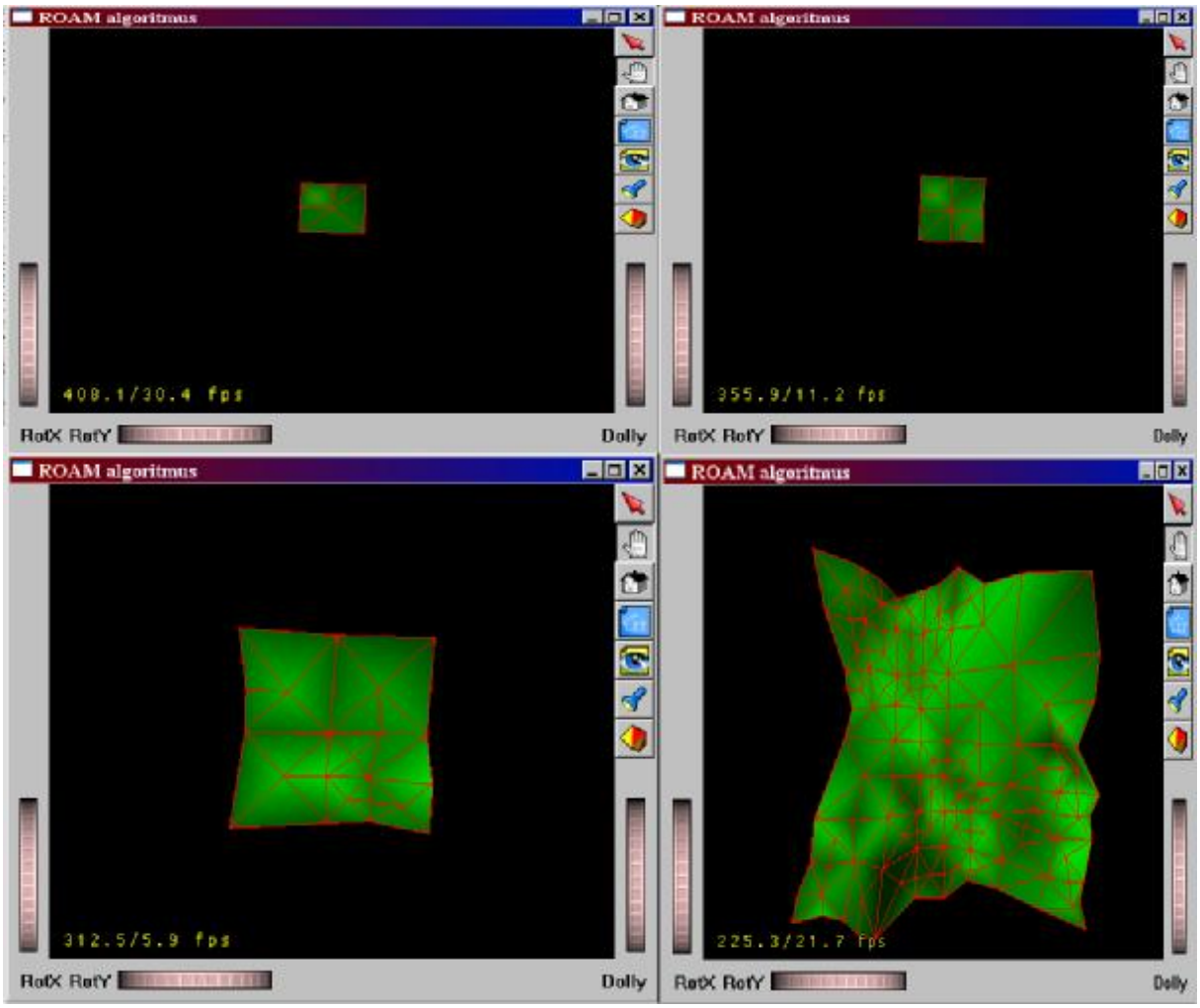
Velmi zajímavé jsou také výpočty **obsazení paměti** při využívání ROAM. Neboť to je nejspíš nejvíce omezující prvek celého algoritmu. Všechna použitá pole a struktury mají mají své přesné místo a nelze je nijak odstranit či omezit. Uvádím tedy ilustrující příklad obsazení paměti pro M: 256x256, uvedené hodnoty jsou orientační, neboť neznám přesné zarovnání obsahu struktur a jiných:

BTT strom = 25,2MB
Pole_korekce = 9,6 MB
Výšková mapa = 0,3MB
Rozdílová mapa = 0,3MB
Pole souřadnic bodů = 0,9MB
Pole indexů bodů = 0,9MB

Vypočteno celkem = 37,2MB
+ lokální proměnné ve funkcích
+ struktury knihovny Coin (OpenInventor)

Naměřeno = 39 MB

40MB je stále ještě únosná metoda, ovšem jak ukazují měření pro M: 1024x1024 je to již 250MB obsazené paměti, což je již příliš mnoho.



Obr. 14. Ukázka změny krajiny při přibližování ke krajině

6. Metody optimalizace

Umožňují urychlení provádění algoritmu, což následně umožňuje renderovat větší plochu nebo detailnější kousky terénu ještě rychleji. Mezi tyto metody patří také podkapitoly 5.1 až 5.3. V této kapitole bych však chtěl uvést ty metody optimalizace, které jsem doposud neověřil konkrétní implementací:

6.1 Ohodnocení jen části stromu

Takovéto ohodnocení by nejspíš způsobilo snížení zatížení systému, je však otázka jak by se projevilo vizuálně, zda by nedocházelo k nežádoucímu rušení. Je však jisté, že by muselo dojít k nápravě i v jiných částech stromu, z důvodů force-split dělení.

6.2 Ohodnocení stromu jen v té části, kde byla minule hranice čepičky

(viz.kapitola 5.2)

Tato metoda by měla nejspíše větší efekt než předchozí, neboť by došlo k úpravě celého stromu, jen by se použila nová metoda procházení stromu. Musela by však nejspíše přibýt nová datová struktura, která by si pamatovala kde byla minule ona hranice a je tedy otázka zda by se tato komplikace vůbec vyplatila.

6.3 Neprovádět ohodnocování stromu pro každý snímek

Touto metodou by se mohlo dosáhnout zvýšení FPS bez větších zásahů do kódu. Jednoduše by se zavolala metoda provádějící ohodnocení stromu jen po každých např. 10 voláních senzoru. To by ovšem vedlo nejspíše k viditelnému skoku ve změně krajiny.

6.4 Rozdíl ve způsobu specifikování vertexů

Použitím indexované verze (SoIndexTriangleStripSet), která obsahuje dvě tabulky: 1. specifikuje souřadnice bodu, 2. obsahuje indexy jednotlivých bodů, ušetříme místo v paměti (viz. přiklad v 6. kapitole tutoriálu o Open Inventoru) a dokonce není tato úspora ani poznat na výkonu neboť je již hardwarově podporována (už od karty GeForce256).

6.5 Rozdělení terénu do sektorů s možným dogenerováváním stromu

Rozdělení terénu do sektorů neboli matice je vhodné pro velmi veliké mapy. Lze to provést tak, že mapu rozdělíme podle 2D matice na mnoho BTT. Což má za následek méně hluboké stromy. Kde vždy vzdálené mapy zobrazím pouze jako dva trojúhelníky a teprve až se kamera přiblíží za nějakou danou hranici, začnu vytvářet strom. Touto metodou dosáhneme toho, že se prochází mnohem menší strom a je tudíž paměťově i výpočetně velmi výhodná. Jediná věc která se musí pamatovat u všech sektorů je výšková mapa. Ovšem lze provést také to, že je výšková mapa generována vždy nová. Tato metoda je tedy asi jediný reálný způsob jak pomocí ROAM algoritmu zobrazit nekonečnou krajinu.

Vystupuje nám zde však řada nových problémů jako je návaznost jednotlivých sektorů map, jak na úrovni výškové mapy tak na úrovni zobrazování ROAM algoritmem. Mohl by se zde například objevit podhled pod krajinu, či zobrazení sešití krajiny způsobený výškovou mapou.

7. Závěr

Vypracováním projektu bylo dosaženo získání teoretických znalostí a implementace algoritmů pro rendrování krajiny, zvláště pak algoritmu ROAM. Použitím ROAM algoritmu při vykreslování výškové mapy, získané algoritmem MidPoint Displacement, byl následným měřením a pokusy zjištěn nárůst výkonu grafického systému. Došlo tedy k potvrzení jeho kvalit.
Možnosti vylepšení dané implementace jsou například:

1. Na každý rendrovaný trojúhelník je možné nanést texturu. Textury lze měnit například podle výšky mapy, náhodně, či jiného důmyslného klíče.
2. Možnost provádět ohodnocování stromu pouze lokálně, či jinou sofistikovanou metodou.
3. Provést implementaci ROAM pro nekonečnou mapu.
4. Provést implementaci ROAM pro nekonečné zanoření stromu (neustále zvyšovat detail krajiny).
5. Spolupráce ROAM algoritmu s jinými algoritmy pro urychlení zobrazování krajiny.
6. Možnosti paralelizace výpočtů.

Jako nejvíce omezující prvek celého algoritmu považuji jeho paměťovou náročnost, kdy je nutné si pro každý bod mapy pamatovat velmi mnoho informací a tedy se zvětšující se mapou velmi roste velikost obsazené paměti, jak je ukázáno v odstavci 5.5.

Program je funkční pouze pod operačním systémem Windows, kde byl také vyvíjen a testován. Pro jiné operační systémy by nejspíše bylo potřeba provést jemné úpravy.

8. Literatura

- Tutoriál Jana Pečivy o knihovně Open Inventor na serveru www.root.cz
- Sánchez-Crespo, Core Techniques and Algorithms in Game Programming, 2003
- Trent Polack, Focus On 3D Terrain Programming, Muska & Lipman/Premier-Trade, 2002, ISBN: 1592000282
- Josie Wernecke, The Inventor Mentor, Addison-Wesley Professional, 1994, ISBN: 0201624958
- Články ze serveru lighthouse.com/opengl
- Článek ze serveru llnl.gov/graphics/ROAM/roam.pdf

9. Přílohy

Licence:

ROAM.cpp, Main-Windows.cpp, Main-Linux.cpp, Makefile

Authors: Pavel Černý (xcerny19 _AT stud.fit.vutbr.cz)
PCJohn (peciva _AT fit.vutbr.cz)

Contributors:

Nastavování parametrů krajiny v kódu programu se provádí na začátku souboru ROAM.cpp (zhruba na 30 – 40 řádku):

#define TERRAIN_DIMENSION 257

Udává rozměr krajiny. Krajina 32x32 čtverců se konstruuje ze 33x33 bodů. Povolené hodnoty jsou 2^N +1 kde N je přirozené číslo vyjma 0 (např. 3, 5 ,9 , 17, 33, 65, 129, 257,…).

#define POSLEDNI_ID 262142

Udává počet uzlů stromu. Vypočte se 4^{N+1} - 2 , kde N je N z mocniny 2 z TERRAIN_DIMMENSION (např. pro krajinu 3x3 bodů je 14, 5x5 je 62, 9x9 je 254,…).

#define PRAH_ZOBRAZENI 250

Udává hranici Rozhodovacího poměru při ohodnocování stromu. Určuje při jaké vzdálenosti a jakém převýšení zobrazovat jaký detail trojúhelníků v mapě. Jeho nastavení tedy záleží na konkrétních požadavcích tvůrce. Obecně platí pro malé mapy (3x3 – 17x17) má hodnotu kolem 20, pro větší mapy (129x129) řádově několika stovek.

#define PRAH_POSUNU_KAMERY 40

Udává max. změnu polohy kamery, kdy se ještě nevolá ohodnocení stromu. Uvedená hodnota značí velikost vektoru – umístnění kamery při posledním přepočtu ku aktuální pozici kamery. Tedy pro urychlení vykreslování lépe větší hodnoty.

#define TERRAIN_HEIGHT_Q 120.f

Maximální výška a hloubka krajiny při vytváření výškové mapy pomocí Midpoin-dispacement algoritmu.

#define VZDALENOST_BODU 1

Koeficient určující jak jsou od sebe vzdáleny body v krajině. Hodnota 1 značí vzdálenost rovnu jednomu pixelu.

#define POUZIJ_ROAM

Příznak zda používat ROAM algoritmus pro vykresleni mapy. Zapoznámkováním tohoto řádku se krajina bude vykreslovat bez ROAM, tedy v plném detailu. Slouží pouze pro porovnání, zda dochází k urychlování.