Vysoké učení technické v Brně

Fakulta informačních technologií

Diplomová práce

Pavel Černý

2006

Zadání

Název:

Zobrazování krajiny a jeho optimalizace pomocí ROAM algoritmu

Vedoucí: Pečiva Jan, Ing., UPGM FIT VUT

Zadání:

1. Prostudujte tématiku zobrazování krajiny a různých relevantních optimalizačních technik.
2. Navrhněte a implementujte robustní řešení pro zobrazování krajiny ROAM algoritmem. Celé řešení detailně popište v diplomové práci.
3. Pokuste se o rozšíření ROAM algoritmu na nekonečnou krajinu. Případně doplňte jiná vlastní vylepšení.
4. Proveďte detailní měření výkonových a paměťových nároků vaší implementace ROAM algoritmu.
5. Proveďte analýzu výsledků a navrhněte směry pro budoucí vylepšení.
6. Diskutujte dosažené výsledky.

Část požadovaná pro obhajobu SP:

Splnění prvních dvou bodů zadání.

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 diplomový 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 podnětné nápady k tématu tvorby krajiny.

Abstrakt

This project comes under the area of computer graphic. It is dealing with the thematic of landscape rendering, so-called outdoors algorithms, more precisely with the algorithms of the landscape rendering acceleration. The project is concerned about the algorithm ROAM, which reduces the number of rendered triangles in dependence on the distance from a given point (e.g . player in a game) and on the landscape camber. Using this algorithm makes possible the depiction of a bigger map with same requirement on the graphical processor.

Assigned project is: „ROAM algorithm for infinite landscape“. It is concerned about the finding the best method solution for infinity landscape using ROAM algorithm. A theoretical analysis of individual methods is performed which result in the implementation of the best method and to the detailed estimation of the parameter.

Key words

Graphics, algoritmus ROAM, Chunked LOD, outdoors, terrain programming, hightmaps, Midpoint displacement, binary triangle tree, BTT, Real-time, Mark Duchaineau, Open Inventor, Coin, SoTriangleStripSet, SoIndexTriangleStripSet, render, FPS, Terrain Engine, infinite map, virtual reality, landscape.

Abstrakt

Projekt spadá do oblasti počítačové grafiky. Zabývá se tématikou rendrování krajiny, tzv. outdoors algoritmů. Přesněji algoritmy z oblasti urychlení rendrování krajiny. Jedná se o algoritmus ROAM, který snižuje počet rendrovaných 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.

Zadání projektu zní: „ROAM algoritmus pro nekonečnou krajinu“. Jde tedy o nalezení nejlepší metody pro řešení nekonečnosti krajiny s využitím ROAM. Je proveden teoretický rozbor jednotlivých metod, praktická implementace nejlepší metody a detailní proměření parametrů.

Klíčová slova

Grafika, algoritmus ROAM, Chunked LOD, krajina, programování terénů, výšková mapa, Midpoint displacement, binary triangle tree, BTT, reálný čas, Mark Duchaineau, Open Inventor, coin, SoTriangleStripSet, SoIndexTriangleStripSet, rendrování, FPS, Terrain Engine, nekonečná mapa, virtuální realita.

Obsah

1 Úvod ..... - 8 -
1.1 Co znamená ROAM ..... - 8 -
1.2 Proč používat ROAM algoritmus ..... - 8 -
1.3 Řešení nekonečné krajiny pomocí ROAM ..... - 9 -
1.4 Další metody optimalizace rendrování krajiny ..... - 10 -
1.5 Návaznost diplomového projektu na ročníkový a semestrální projekt ..... - 10 -
2 ROAM pro jeden čtverec mapy ..... - 11 -
2.1 ROAM algoritmus dnes ..... - 11 -
2.2 Obecný popis algoritmu ..... - 11 -
2.3 Obecný popis implementace pro jeden čtverec mapy ..... - 13 -
3 Nekonečná krajina ..... - 19 -
3.1 Obecné rozdělení metod tvorby ..... - 19 -
3.2 Metoda Čtyř čtverců ..... - 19 -
3.3 Metoda Snižování levelu krajiny ..... - 22 -
3.4 Pásková metoda ..... - 25 -
3.5 Modifikovaná metoda Čtyř čtverců ..... - 26 -
3.6 Kombinovaná metoda Chunked-LOD a ROAM ..... - 27 -
3.7 Výběr nejlepší metody ..... - 29 -
4 Implementace ..... - 30 -
4.1 ROAM algoritmus pro jeden čtverec mapy ..... - 30 -
4.2 Nekonečná krajina pomocí metody Čtyř čtverců ..... - 34 -
4.3 Další části implementace ..... - 36 -
5 Měření ..... - 41 -
5.1 Počítače na kterých bylo prováděno měření ..... - 42 -
5.2 Druhy testů ..... - 42 -
5.3 Výsledky měření ..... - 44 -
6 Závěr ..... - 54 -
7 Literatura ..... - 55 -
8 Seznamy ..... - 56 -
8.1 Seznam obrázků ..... - 56 -

8.2 Seznam tabulek ..... - 57 -
8.3 Seznam grafů..... - 57 -
Přílohy ..... - 58 -

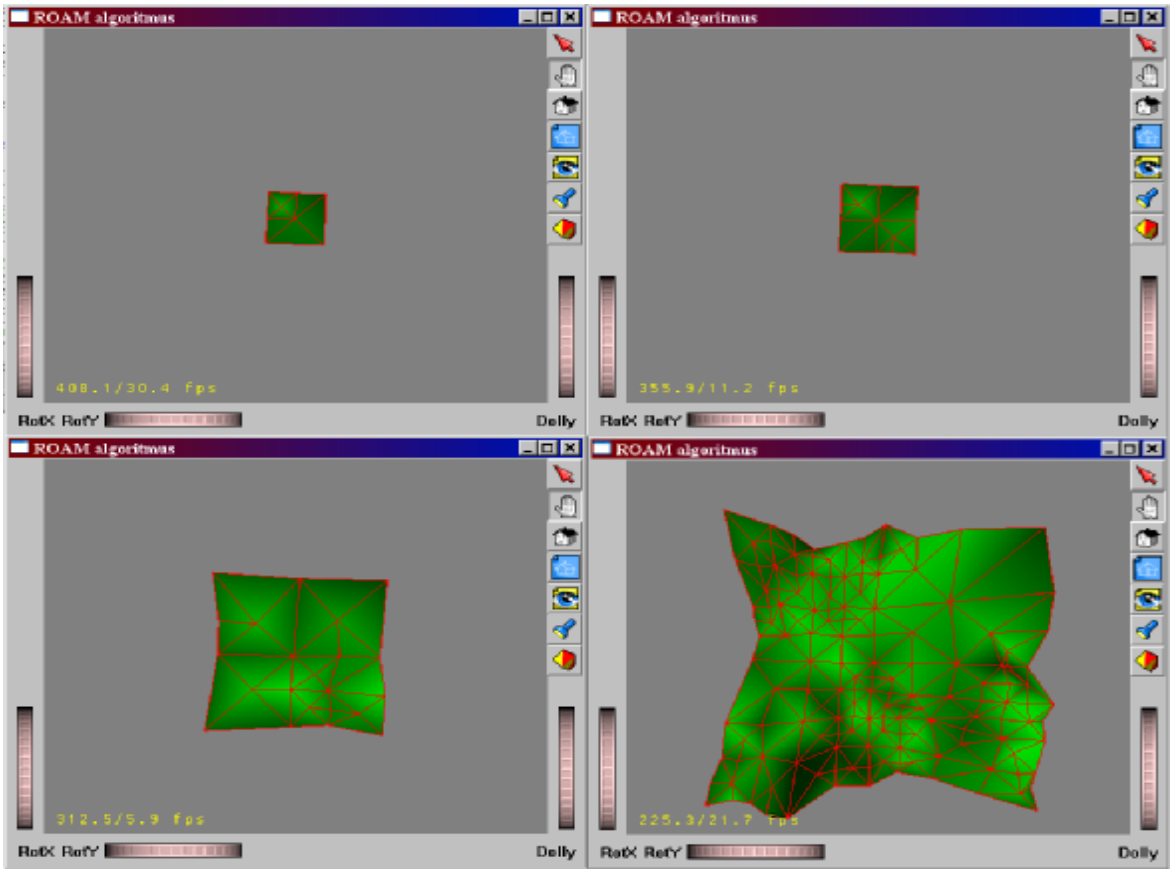
1 Úvod

1.1 Co 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 s vyšším rozlišením (např. 512 x 512 bodů), jednoduše z ní vytvoříme síť trojúhelníků, ve které se musíme pohybovat, a to nám velmi 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.



Obr. 1.: Intuitivní přiblížení ROAM algoritmu
Ukazuje postupné přibližování kamery k mapě, což má za následek zvětšování detailu krajiny.

1.3 Řešení nekonečné krajiny pomocí ROAM

Popis algoritmu v literatuře je velmi obecný a popisuje pouze základní myšlenku. Konkrétní implementace je plně v rukou vývojových pracovníků, neboť je velmi závislá na požadovaném cíli a konkrétní technologii. Je nutné nastavit optimálně kompromis mezi rychlostí a paměťovou náročností či mírou detailu krajiny. Popis ROAM algoritmu pro nekonečnou krajinu není v žádné literatuře popsán, neboť se jedná pouze o specifickou část problému, která není často řešena.

Problém nekonečné krajiny spočívá v návaznosti jednotlivých čtverců mapy. Pokud bychom neřešili návaznosti mezi čtverci krajiny a jednoduše bychom použili ROAM algoritmus na každý čtverec zvlášť, docházelo by k chybám zobrazení jako je například pohled pod krajinu. Proto je nutno vytvořit systém, který takovéto chyby opravuje nebo úplně vylučuje. Teoreticky jsou jednotlivé metody rozebrány v kapitole 3. Prakticky je implementována metoda Čtyř čtverců, s jejímiž výsledky je možno se seznámit v kapitole 4.

1.4 Další metody optimalizace rendrování krajiny

Metody pro urychlení zobrazování krajiny mají za cíl modifikaci trojúhelníků (polygonů), čímž dojde ke snížení zátěže grafické karty. Do této kategorie patří kromě ROAM algoritmu také mnoho dalších metod. Nejblíže ROAM je metoda **Chunked-LOD** (chunk = kus, LOD = Level of detail), jejíž princip spočívá v rozdělení krajiny do menších segmentů, u kterých se podle vzdálenosti od kamery nastaví míra detailu. Pro menší míru detailu dojde k redukci počtu zobrazovaných trojúhelníků. Další metodou je **Triangle-strip method**, která využívá urychlení vykreslování při zadání dat v triangle-strip tvaru. Nedochází zde ke snížení počtu trojúhelníků, nýbrž k jejich přeuspořádání do požadovaného tvaru.

1.5 Návaznost diplomového projektu na ročníkový a semestrální projekt

V ročníkovém projektu byla teoreticky nastudována a prakticky zpracována problematika týkající se ROAM algoritmu. Implementace byla provedena pro jeden čtverec mapy. Dále byly řešeny problémy týkající se výpočtu normál pro hladké zobrazování krajiny.

Semestrální projekt navazuje na projekt ročníkový. Zde byly zpracovány teoretické podklady pro rozšíření ROAM algoritmu na nekonečnou krajinu. Také bylo provedeno implementování filtrování krajiny, řešení kolizí a ovládání pomocí kláves. Dále nastudování projektu Terrain Engine[1], ze kterého jsou využity měřiče parametrů.

Diplomový projekt navazuje na projekt semestrální. Bylo provedeno přepracování ROAM algoritmu z důvodů jeho urychlení a nalezení nových metod optimalizace. Provedena implementace nekonečné krajiny s možností detailního měření jednotlivých částí algoritmu.

[1] Zdroj: HAVLÍČEK, Martin. Knihovna pro práci s výškovými mapami [diplomová práce]. VUT v Brně, Fakulta informačních technologií.

2 ROAM pro jeden čtverec mapy

2.1 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.

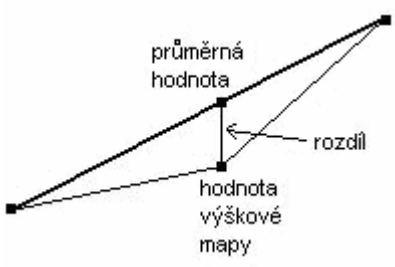
Musíme však počítat i s omezeními či nedostatky tohoto algoritmu. Algoritmus nelze použít pro krajinu s úhly mezi trojúhelníky větším než devadesát stupňů – tvořící strže nebo jeskyně. Je zde také problém umístnění objektů na krajině neboť dochází k neustálým změnám výšky krajiny. A v neposlední řadě také problém filtrace krajiny či výpočtu normál na okraji mapy.

2.2 Obecný popis algoritmu[2]

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 vektorů, 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 představen na následujícím obrázku.

[2] Zdroj: TRENT, Polack. Focus On 3D Terrain Programming, Muska & Lipman/Premier-Trade. 2002, ISBN: 1592000282.



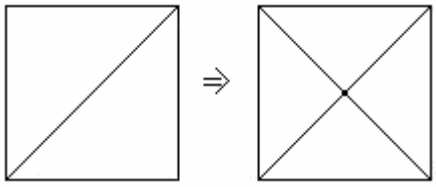
Obr. 2.: Výpočet hodnot v Poli rozdílů
rozdíl = | průměrná .hodnota. – reálná hodnota výškové mapy |

V části dynamické, která musí nutně proběhnout co nejrychleji, je již pouze využíváno datových struktur z části statické. Jsou zde dodatečně nastavovány 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 trojúhelníku od kamery a chyby, která by vznikla při zániku daného trojúhelníku:

$$ne/zobrazit = \frac{vzdálenost\_bodu\_od\_kamery}{hodnota\_z\_pole\_rozdílů} < hodnota\_prahu$$

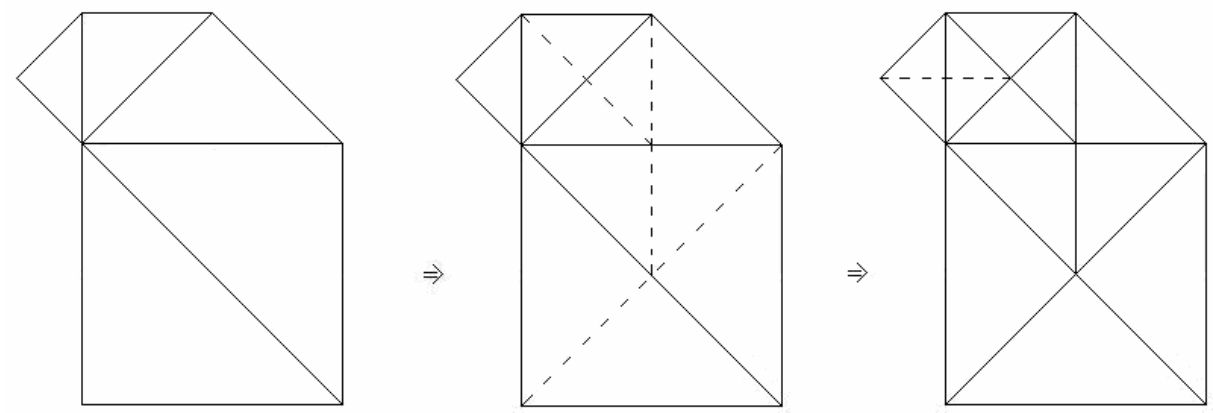
Je-li tento podíl menší než prahová hodnota, je tento bod zrušen a přiléhající čtyři trojúhelníky nahrazeny pouze dvěma. Nahrazení dvěma trojúhelníky je trochu zjednodušeně řečeno. Pro nahrazování jsou definovány tři tzv**. „rozdělovací pravidla**":

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



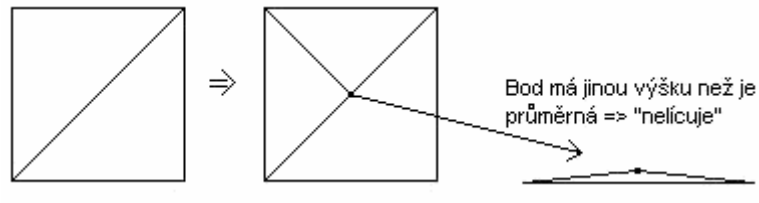
Obr. 3.: Part of a diamond

2. Je-li bod na okraji mapy (at the edge of a mesh). Dělení je analogicky s bodem prvním (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. 4.: Force-split

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



Obr. 5.: 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“.

2.3 Obecný popis implementace pro jeden čtverec mapy

2.3.1 Statická část

2.3.1.1 Výšková mapa

Získat výškovou mapu lze mnoha způsoby, například pouhým načtením obrázku v odstínech šedi (s čtvercovým rozlišením o velikosti 2^n). Tento způsob ovšem není z důvodů testování ROAM algoritmu vhodný. Proto je v mé implementaci použit algoritmus Midpoint displacement, který při každém spuštění vygeneruje jinou výškovou mapu. Z toho plynou vždy jiná výchozí data pro ROAM algoritmus.

2.3.1.2 Rozdílová mapa

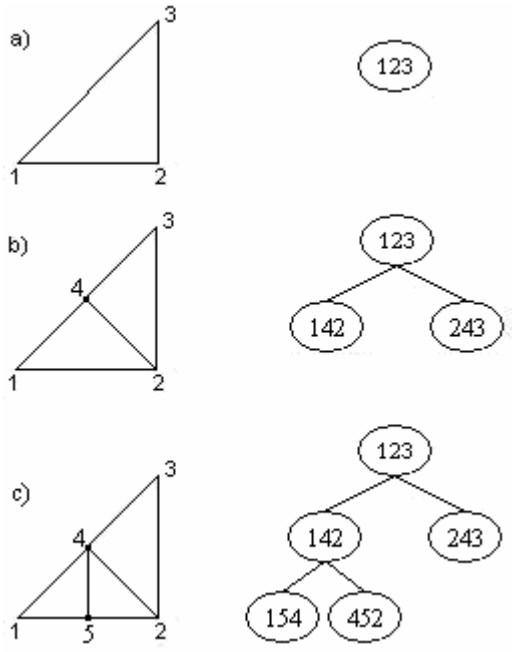
Následuje funkce, jejímž úkolem je vytvoření rozdílové mapy. Tato mapa je využívána při rozhodování, zda daný trojúhelník vykreslit či ne (viz dále). Tvorba této mapy je velmi jednoduchá a již byla popsána v kapitole třetí. Jelikož je nutné při tvorbě procházet všechny body mapy, jakožto i v následující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

2.3.1.3 Tvorba BTT

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 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 o uzlech tvořících s aktuálním uzlem diamond strukturu). 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 dvou 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 trojúhelníku. 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.



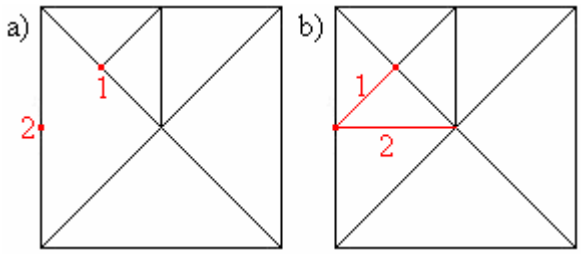
Obr. 6.: Ukázka dělení trojúhelníku a vytváření stromu
Nový uzel vznikne vždy přidáním nového indexu bodu mezi první dva indexy otce (pro levý uzel) a mezi druhé dva indexy otce(pro pravý uzel).

V bodě a) je strom tvořen jediným trojúhelníkem zadaným body 1,2,3. 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).

Aplikací těchto pravidel je v cyklu vytvořen celý BTT, přičemž je dodrženo predixové indexování uzlů stromu (pro připomenutí - nejprve pohled na strom zleva, poté zespodu a nakonec zprava).

2.3.1.4 Korekce

Jedná se o velmi jednoduché pole do kterého se ukládají (pro každý bod výškové mapy) hodnoty indexů trojúhelníků, pro které je ten konkrétní bod prostředním bodem (bod u pravého úhlu). Tato informace je následně přesunuta do BTT, kde je využívána pro diamond dělení. Pokud bychom tuto informaci neměli k dispozici, nemohli bychom dělení provést, neboť by nebylo jasné, které trojúhelníky daný diamond tvoří (trojúhelníky se nachází v různých částech stromu). Po určení, které trojúhelníky tvoří diamond je nutné provést také korekci jejich otců v BTT (neboli force-split dělení).



Obr. 7.: Ukázka korekce

Provedeme korekci – v bodě 1 (obr. a)) zjistíme nesrovnalost, napravíme přímkou 1 (obr. b)), a jelikož postupujeme po otcích v BTT vzhůru napravíme také bod 2 přímkou 2.

2.3.2 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. Dynamická část je provedena dvouprůchodově. První průchod tvoří ohodnocení BTT, druhý poté naplnění pole indexů.

2.3.2.1 Ohodnocení BTT

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

Specifikace pojmů: Otec = nadřazený uzel (aktuální uzel vznikl jeho dělením),
Syn = podřazený uzel (vznikl dělením aktuálního uzlu),
Příznak zobrazení = N – znamená trojúhelník nezobrazovat,
Příznak zobrazení = A – znamená trojúhelník 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 – nic mezi tím neexistuje.
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 synovi. Plyne z pravidla 3.

Pomocí těchto pravidel a „rozhodovacího poměru“ (již byl popsán v odstavci 2.2 Obecný popis algoritmu) jsou tedy postupně ohodnoceny všechny uzly stromu. Tato pravidla mají však za následek dělení pouze sousedních trojúhelníků a jejich otců (force-split). Dělení part-of-diamond je v BTT prováděno náročně, neboť jednotlivé trojúhelníky pro toto dělení jsou od sebe velmi vzdáleny. Proto byla do BTT (pro každý uzel) přidána data z pole Korekce, která nesou informaci o ID dalších tří uzlů, jež tvoří spolu s aktuálním uzlem diamond strukturu. Příznak zobrazení těchto uzlů musí být nastaven na stejnou hodnotu jako uzel aktuální. Pokud je hodnota zobrazení rovna A, je nutné u zbylých nastavit také zobrazení na A a samozřejmě korigovat hodnoty jejich otců.

2.3.2.3 Naplnění pole indexů

Nyní následuje poslední funkce této dynamické části, která naplní pole indexů. Pole indexů je složeno ze 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 synové 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 až do ukončení průchodu celým stromem. Pole indexů je naplněno ukazateli na body jednotlivých trojúhelníků.

2.3.3 Poznámky

Postup ohodnocování a plnění indexů, tak jak je popsáno v odstavci 2.3, odpovídá první rozvaze a implementaci ROAM algoritmu. Existuje mnoho modifikací těchto algoritmů, z nichž mnohé jsem v praxi vyzkoušel a porovnal jejich kvality. Ovšem pro uvedení do problému a objasnění základních principů je tato první rozvaha velmi vhodná a není třeba problém příliš komplikovat.

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. Klasickým způsobem zjistíme průměrnou výšku prostředního bodu (např. výšky krajních bodů děleno čtyřmi) 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.

3 Nekonečná krajina

3.1 Obecné rozdělení metod tvorby

Důležitý pohled na nekonečnou krajinu je realizace jejího zobrazení. Můžeme k již zobrazené mapě stále přidávat další trojúhelníky, tak jak je při pohybu krajinou generujeme. Toto řešení brzy povede k zahlcení paměti. Jiné řešení je zobrazovat vždy jen jeden čtverec krajiny, který se při postupu krajinou nezvětšuje, ale mění se jeho obsah – tvar krajiny. Toto je mnohem praktičtější řešení, které je v praxi také mnohem častější a zastávám ho i já.

Metody tvorby nekonečné krajiny můžeme rozdělit, v závislosti na požadovaném tvaru mapy krajiny:

a) Metody pracující s libovolnou mapou
b) Metody požadující čtvercovou mapu mocniny dvou

Do první skupiny (a) se řadí metody, které nemají žádné zvláštní požadavky na tvorbu krajiny Jsou schopny pracovat s libovolnou krajinou. Do druhé skupiny (b) patří metody, které vyžadují například dodatečné generování kousků krajiny po jiných než čtvercových (např. obdélník 2x65 bodů) nebo vyžadující mezi každým čtvercem krajiny tzv. vyhlazovací pás (viz odstavec 3.4).

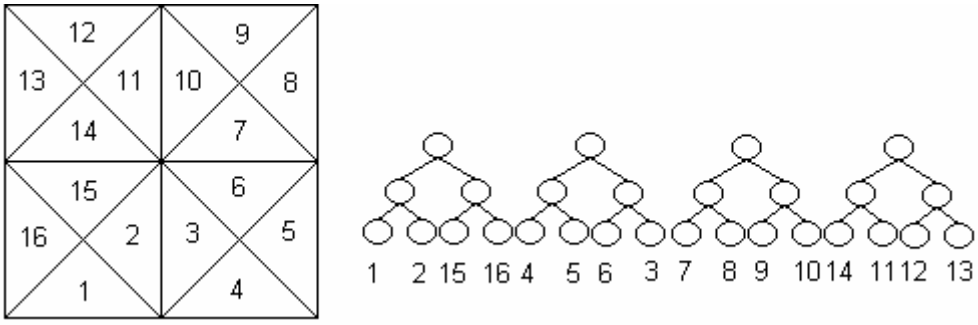
Názvy jednotlivých metod nejsou oficiální, ale jsou mým výtvorem (jakožto i metody samotné) a to z důvodu lepší orientace v textu. Jednotlivé názvy jsou voleny převážně tak, aby vystihovaly hlavní myšlenku dané metody.

Podívejme se tedy nyní blíže na jednotlivé metody a u každé si ukažme její výhody a nevýhody. Výsledkem této analýzy by měla být metoda, která má nejlepší teoretické předpoklady pro nekonečnou ROAM krajinu, a má tedy smysl pokusit se o její praktickou realizaci.

3.2 Metoda Čtyř čtverců

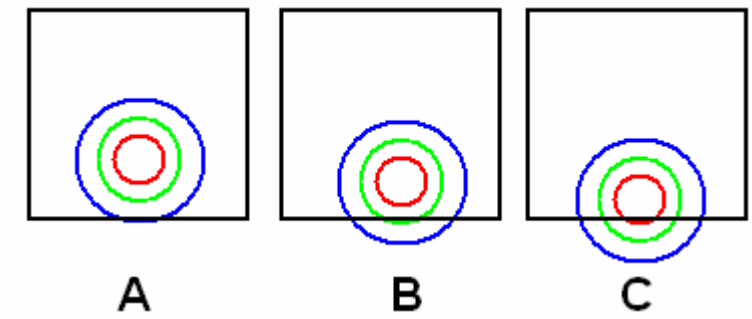
Tato metoda patří do druhé kategorie (b). Požaduje čtvercovou krajinu mocniny dvou – s jinou krajinou pracovat neumí. Je založena na jednoduchém principu: kolem kamery (hráče) je kruhová oblast určující místo v mapě, kde se hráč právě pohybuje. Tato oblast je zpravidla menší než jeden

čtverec mapy. Celá viditelná mapa je tvořena čtyřmi aktivními čtverci mapy, kde pro každý čtverec je vytvořen jeden BTT.



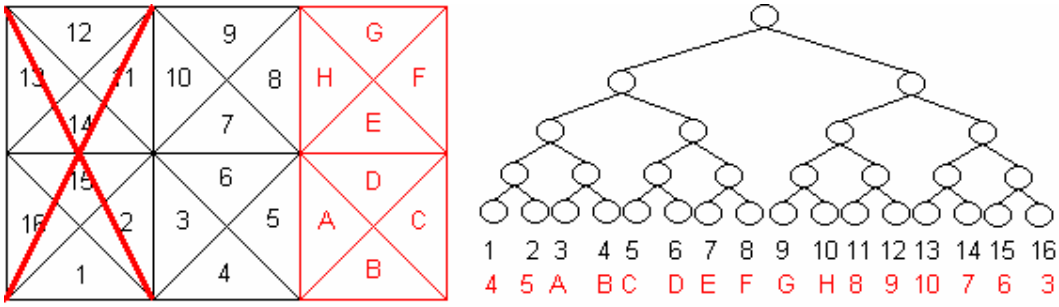
Obr. 8.: Čtyři čtverce – čtyři BTT
Pro každý čtverec je vytvořen jeden BTT. Čísla ve stromu odpovídají číslům trojúhelníků v mapě.

Při pohybu hráče v krajině se zjišťuje, ve které části takto vytvořené mapy se právě nachází. Pokud se hráč příliš přiblíží k okraji mapy, dojde k načtení dalších čtverců mapy, zároveň s tím dojde ke kopírování potřebných dat v BTT a k zahození již nepotřebných částí mapy a BTT. V konečném důsledku to povede opět k mapě, která je tvořena čtyřmi čtverci. Míru přiblížení hráče k okraji mapy lze definovat dříve zmíněnou oblastí kolem hráče. Jakmile dojde k průsečíku oblasti a okraje mapy, dojde k načítání nových čtverců. Velikost oblasti určuje autor hry, ale z logického hlediska musí být menší než jeden čtverec mapy. Při použití více takovýchto oblastí lze jednoduše provést rozdělení úloh. Například při průchodu největší oblastí dojde k vytvoření mapy. Pokud se hráč přiblíží ještě více k okraji (protne další oblast), dojde ke změně BTT a při průchodu třetí oblastí dojde k odstranění starých čtverců a použití nových (viz obr.9). Výhodou takového dělení úkolů je, že nedochází k nežádoucímu zasekávání při pohybu krajinou. Bohužel někdy dochází k tomu, že hráč od okraje mapy opět odstoupí a vykonaná práce byla zbytečná.



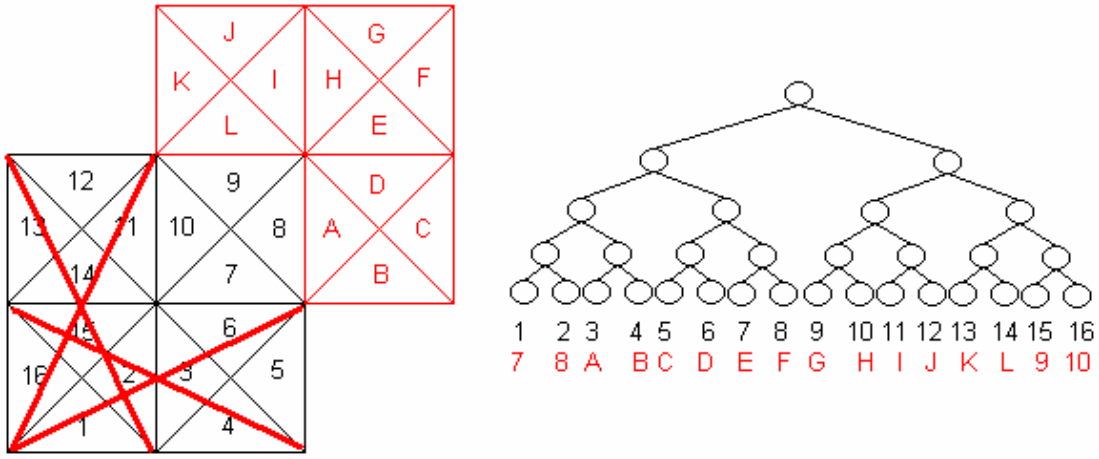
Obr. 9.: Oblasti kolem hráče
Přibližováním hráče k okraji mapy dochází postupným průnikům oblastí s okrajem, což má za následek odstartování provádění akcí potřebných pro zobrazení nových čtverců mapy.

Přiblížením hráče k okraji mapy může dojít v zásadě ke dvěma případům načtení nových čtverců, které lze nejlépe ilustrovat následujícím obrázkem.



Obr. 10.: Načtení dvou čtverců

Při posunu hráče k pravému okraji mapy dojde k načtení čtverců mapy s abecedním označením trojúhelníků, přičemž čtverce mapy červeně přeškrtnuté z mapy zmizí. Ve stromě BTT pak dojde pouze k přeuspořádání starých dat a k nahrazení některých novými daty (zobrazeny červeně).



Obr. 11.: Načtení tří čtverců

Při posunu hráče k pravému hornímu okraji mapy dojde k načtení čtverců mapy s abecedním označením trojúhelníků, přičemž čtverce mapy červeně přeškrtnuté z mapy zmizí. Ve stromě BTT pak dojde pouze k přeuspořádání starých dat a k nahrazení některých novými daty (zobrazeny červeně).

Je tedy zřejmé, že vždy zůstanou pouze čtyři čtverce, přičemž počet čtverců načtených se rovná čtvercům zahozeným. Zda dojde k situaci z Obr. 10 či Obr. 11 záleží opět na nastavení oblasti kolem hráče.

Z obrázků je dále zřejmá změna přístupu k BTT. Nejedná se již o čtyři nezávislé BTT, ale pouze o jeden strom, který byl pomocí několika umělých uzlů vytvořen. Umělým uzlem se rozumí uzel, který nenese informace o odpovídajícím trojúhelníku z mapy. Což odpovídá kořenovému uzlu a jeho dvěma synům, jejichž jediným smyslem je svázat čtverce mapy. Tím, že jsme svázali stromy

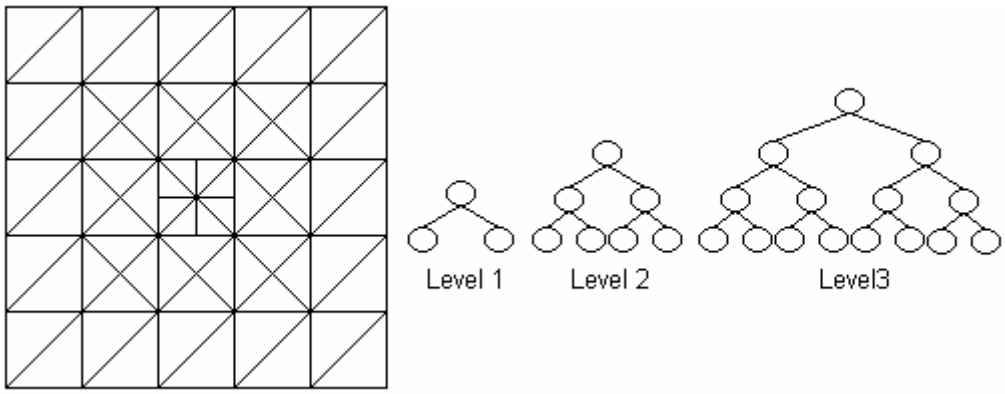
do jednoho, jsme vyřešili veliký problém návaznosti jednotlivých čtverců mapy. Pokud sloučíme čtverce do jednoho BTT, můžeme k němu přistupovat jednoduchými algoritmy použitými v ROAM pro jediný čtverec. Musíme pouze zajistit, aby nedošlo k zobrazení umělých uzlů (lze velmi jednoduše ošetřit). Při nahrazení čtverců krajiny pak dojde pouze k přeuspořádání a nahrazení některých uzlů tak, jak je naznačeno u Obr. 10 a Obr. 11. A to vždy právě na této čtvrté úrovni stromu, neboť pod touto úrovní se již změny neprojeví, což plyne z logiky stromu. Víme totiž v které části mapy se hráč nachází, můžeme tak určit, které čtverce mapy je třeba nahrazovat a které je tedy nutno přesunout. Pokud bychom neprovedli sloučení v jediný strom, museli bychom řešit návaznosti jednotlivých čtverců krajiny. To lze provést například zasíláním zpráv mezi jednotlivými čtverci mapy nebo pomocí speciální korekční funkce. Ta vždy projde všechny body na hranici mezi čtverci mapy a zkontroluje příznaky zobrazení. Pokud se příznaky liší, provede korekci (např. vždy nastaví příznak zobrazení = A).

Obvyklé použití ROAM algoritmus, jak již bylo řečeno dříve, je ve hrách. U her jsou obvykle mapy načítány ze souborů, neboť byly s ohledem na příběh vytvořeny již dříve. Není tedy nejmenší problém, aby byly spolu s mapami uloženy také BTT, což jistě povede ke značnému urychlení.

3.3 Metoda Snižování levelu krajiny

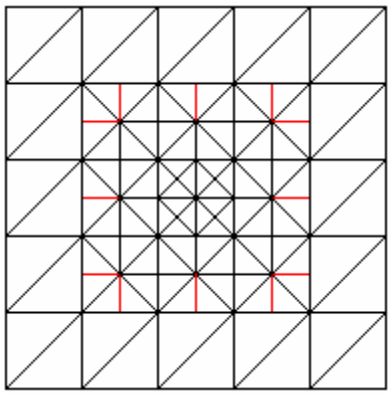
Tato metoda je velmi podobná metodě Čtyř čtverců. Patří také do druhé skupiny (b), požaduje tedy čtvercovou krajinu mocniny dvou, která je generována nějakou externí funkcí nebo načítána ze souboru.

Celá mapa krajiny není složena pouze ze čtyř čtverců mapy, ale z mnohem více. Počet čtverců částečně závisí na velikosti levelu krajiny prostředního čtverce. Levelem krajiny označujeme počet „řádků“ uzlů BTT. Jelikož je BTT jen jiné vyjádření trojúhelníků v mapě, lze level odvodit také přímo z mapy – jako počet dělení základního trojúhelníku. Jedná se o detail krajiny (viz Obr. 12).



Obr. 12.: Metoda Snižování levelu krajiny
BTT levelu 1 odpovídá čtvercům mapy na hranici okraje mapy,
BTT levelu 2 označuje čtverce rozdělené na 4 trojúhelníky,
a BTT levelu 3 odpovídá středovému čtverci.

Jak takovouto mapu vytvořit? Zvolíme si level prostředního čtverce krajiny a poté vždy přidáme další čtverce kolem něj s levelem o jeden menším, až dosáhneme levelu 1. Přičemž pokles nemusí být pouze o jeden level, ale i o více. Při podrobnějším pohledu na mapu z Obr. 12 dojdeme k závěru, že prostřední čtverec (nejvyšší level) má detail zvolen špatně, neboť listové uzly nebude možno nikdy zobrazit s ohledem na chyby zobrazení. To je pravda, a proto je třeba upozornit na nutnost vždy si pořádně promyslet mapu pro tuto metodu. Jedno z vhodných řešení mapy o velikosti uvedené na Obr. 13 je například použití levelů 1, 3, 4.

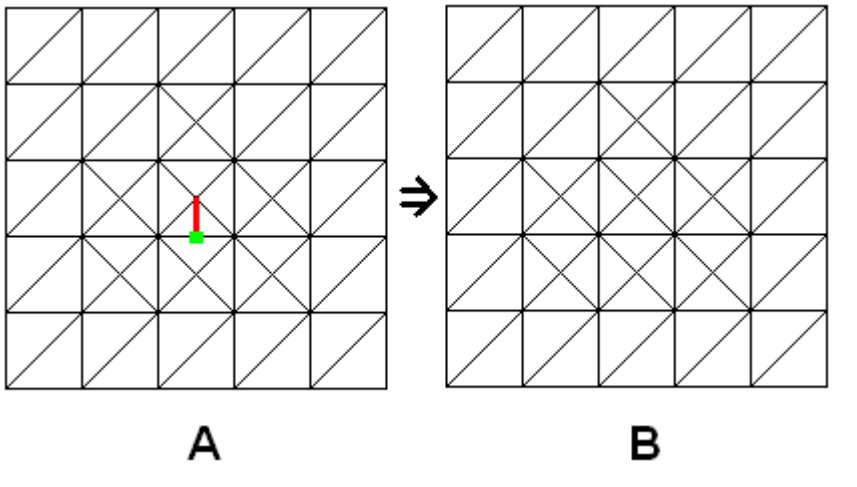


Obr. 13.: Vhodnější volba mapy
Mapa s velikostmi levelů postupně od kraje ke středu: 1, 3, 4.
Červeně zobrazené hrany nepůjde nikdy zobrazit, ale ostatní již ano.

V podstatě jde o ulehčení základní myšlenky ROAM algoritmu, tedy o takový hybridní ROAM. Jelikož se hráč prakticky pohybuje pouze v prostředním čtverci, je v něm logicky nastaven největší detail krajiny. S rostoucí vzdáleností se detail krajiny uměle snižuje, což splňuje jednu ze dvou podmínek ROAM algoritmu, ve kterém je ještě zabudována podmínka velikosti převýšení. Toto umělé snížení detailu má za následek jednu podstatnou věc. Tou je právě zmíněné snížení levelu BTT – tedy snížení počtu uzlů. Pokud snížíme počet uzlů, snížíme také paměťovou náročnost ROAM algoritmu (viz odstavec 4.) a samozřejmě i počet uzlů, které musí tento algoritmus procházet.

Jde ovšem o to, zda je takové umělé snížení detailu vůbec realizovatelné. Při pohybu hráče krajinou je nutné vždy při překročení hranice prostředního čtverce změnit detail všech čtverců a zároveň načíst nové čtverce a staré zahodit (stejně jako u metody Čtyř čtverců viz 3.2). Tato operace vypadá velmi časově náročná. Podíváme-li se však na ni podrobněji, zjistíme, že může být opět rozdělena na více menších částí za pomoci využití oblastí kolem hráče (viz 3.2). Zjistíme také, že operace zvýšení či snížení detailu není tolik náročná, jak by se mohlo zprvu zdát. Jedná se o jedinou věc – úpravu BTT (nikoliv jeho celé předělání). Při snížení levelu jednoduše odstraníme všechny listové uzly stromu. Naopak při zvýšení detailu tyto uzly přidáme. Výpočet nových listových uzlů navíc není nikterak složitý a lze jej dokonce úplně obejít, pokud načítáme mapu ze souboru, kde je již BTT vytvořen.

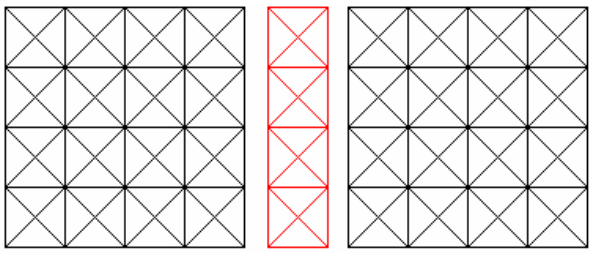
Nadále nám zůstává problém návaznosti jednotlivých čtverců mapy z hlediska ROAM algoritmu. Bohužel nelze použít umělého spojení BTT, tak jak bylo ukázáno v odstavci 3.2, neboť jednotlivé stromy se liší svým levelem. Vznikl by tak strom, který neumí ROAM algoritmus procházet. Zbývá tedy řešení za pomocí Korekční funkce, která navíc musí vždy při neshodě zobrazení končit výsledkem zobraz = N, protože stromy s nižším levelem daný bod zobrazení vůbec neobsahují. Nemají informace potřebné k jeho zobrazení (viz Obr. 14). Případně lze korekční funkci zahrnout přímo do algoritmu ohodnocování uzlů. A to tak, že při ohodnocování uzlu na hranici levelů se vždy nastaví na zobraz = N.



Obr. 14.: Korekční funkce
Obrázek A je mapa po ohodnocení všech čtverců ROAM algoritmem, červená hrana = nepřípustná. Obrázek B je mapa po průchodu korekční funkce – zjistila neshodu zobrazení zeleného bodu u jednotlivých čtverců, nastavila jim proto oběma příznak zobrazení N.

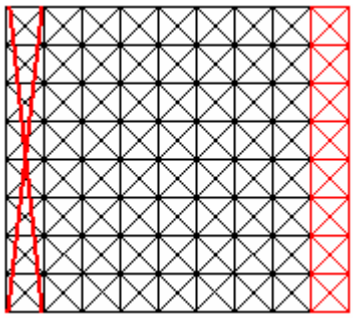
3.4 Pásková metoda

Zde je první metoda, která patří do první kategorie (a), což znamená, že je nezávislá na tvaru krajiny. Tedy skoro nezávislá – platí zde stále omezení velikosti mapy mocniny dvou. Ovšem posun krajiny není prováděn načtením celých čtverců mapy, ale pouze jednoho pásu bodů. Můžeme tak zpracovávat libovolnou krajinu o výšce mocniny dvou. Například krajinu složenou ze čtverců mapy, mezi které jsou vloženy tzv. vyhlazovací pásky.



Obr. 15.: Spojení čtverců mapy vyhlazovacími pásky
Nelze zobrazit metodou typu b).

Bohužel žádná výhoda není zadarmo a je nutno za ni náležitě zaplatit. Zde budeme platit časem. Přidáním jednoho pásku bodů a zrušením jiného se nevyhneme náročné operaci změny BTT, která bude navíc prováděna téměř při každém průchodu tohoto stromu, neboť čtverce této mapy jsou velmi malé.



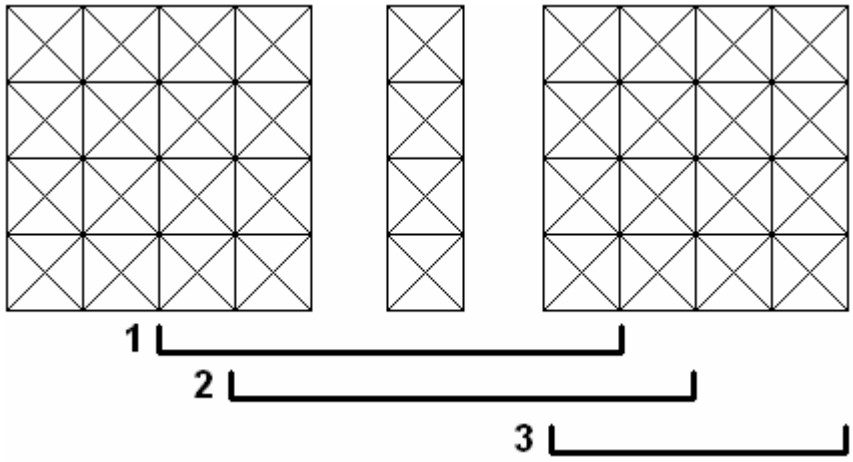
Obr. 16.: Posun krajiny v Páskové metodě
Při posunu krajiny dojde k načtení dvou sloupců bodů –červené a zahození jiných dvou sloupců-přeškrtnuté .

V podstatě se jedná o metodu Čtyř čtverců modifikovanou na použití do první kategorie (a). Výsledným efektem je, že nedochází k zasekávání krajiny, jako je tomu u metody Čtyř čtverců, zejména při načítání nových čtverců (velké kusy krajiny). Zde je krajina načítána více spojitě. Bohužel načítání je časově náročné (teoreticky téměř zdvojnásobení času). Je proto nutné používat menší mapu, která takovéto ztráty času kompenzuje. Výhodou zůstává, stejně jako u metody Čtyř čtverců, že máme pouze jeden BTT, nemusíme proto řešit konflikty mezi čtverci mapy.

3.5 Modifikovaná metoda Čtyř čtverců

Jedná se o pokus změnit, pokud možno minimálně, metodu Čtyř čtverců, pro použití v kategorii první. A to tak, aby byly co nejvíce zachovány její výhody. Na rozdíl od Páskové metody, která je velmi omezující.

Myšlenka je opravdu velmi jednoduchá. Kategorie první (a) vyžaduje práci s libovolnou mapou. Tuto podmínku však málokdy plně využijeme, většinou nám postačuje posun krajiny po jiných než čtvercových kusech, nejčastěji vkládání vyhlazovacích pásů. Což je pro klasickou metodu Čtyř čtverců neřešitelný problém. Stačí ovšem drobná modifikace. Jednoduše lze dát před tuto metodu funkci, která bude mít za úkol řídit přidělování čtverců krajiny. Funkce bude mít k dispozici data o dvou čtvercích krajiny a o vyhlazovacím pásu. Načte buď celý čtverec mapy nebo jen jeho menší část, kterou spojí s částí předchozího čtverce. Provede v podstatě přeuspořádání dat krajiny tak, aby výsledek odpovídal čtyřem čtvercům.



Obr. 17.: Funkce pro tvorbu čtverců mapy
Svorka s číslem 1 naznačuje první posun krajiny (spojí vyhlazovací pás a 1. pás další mapy v jeden čtverec), svorka 2. a 3 naznačuje další kroky v posunu krajiny.

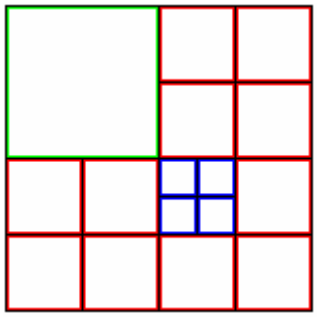
Je jasné, že tato funkce bude mít také za následek mírné zpomalení vykreslování krajiny, ale určitě nebude tak veliké jako u metody 3.4. Zůstanou též zachovány všechny vlastnosti metody Čtyř čtverců, jakožto i řešení návaznosti čtverců z pohledu ROAM algoritmus. Jedinou změnou je možnost ukládání BTT spolu s mapou do souboru. Neboť korekční funkce postupuje i po nečtvercových kusech krajiny. Musí být tedy BTT vytvořen vždy znovu, stejně jako při generování krajiny. Tvorba této korekční funkce navíc není nijak komplikovaná. Jedná se o přeuspořádání několika málo bloků dat dvourozměrného pole.

3.6 Kombinovaná metoda Chunked-LOD a ROAM

Chunked-LOD[3] a ROAM algoritmus jsou dvě nejpoužívanější metody pro urychlení zobrazení 3D krajiny. Každá metoda má své klady i zápory. Pokus o spojení těchto dvou metod, může vést k vhodnému kompromisu. Výsledná metoda bude opět patřit do skupiny druhé (b), ale po drobných úpravách, stejně jako v odstavci 3.5, ji lze zařadit do skupiny první (a).

Vycházíme opět z metody Čtyř čtverců. Lze tedy v krajině najít čtyři základní čtverce, které však již nepůjde sloučit do jediného BTT, neboť jednotlivé čtverce mají různý detail zobrazení (level). Intuitivní pochopení opět necháme na následujícím obrázku:

[3] Zdroj: VRBA, Petr. Chunked-LOD, Perlinova funkce [Ročníkový projekt]. VUT v Brně, Fakulta informačních technologií.



Obr. 18.: Kombinovaná metoda Chunked-LOD a ROAM
Každý zobrazený čtverec na mapě (nezáleží na barvě)
reprezentuje jeden BTT o velikosti mapy např. 32x32

Jak již bylo ukázáno v odstavci 3.3, je zvýšení či snížení detailu krajiny o dvojnásobek velmi rychle proveditelé i na úrovni BTT. Jde o přidání nebo ubrání listových uzlů stromu. Jedná se tedy pouze o BTT, nikoliv o výškovou mapu – ta je stále v plném detailu.

Postup je jednoduchý, v části krajiny nejblíže k hráči je provedeno největší dělení krajiny (modrá oblast). Čím dále od hráče, tím menší dělení krajiny, ovšem stále platí, že pohyb s detailem je možný pouze rozdělením čtverce na čtyři syny. Takto popsaný algoritmus v zásadě odpovídá myšlence Chunked-LOD, my zde však přidáme také ROAM algoritmus, tím že je do každého čtverce vložen BTT. Z čehož plyne, že ne každý trojúhelník ve čtverci se zobrazí. Dojde tak k další korekci počtu zobrazených trojúhelníků, v závislosti na pravidlech ROAM algoritmu.

Řešení konfliktů návaznosti čtverců lze v této metodě provést pouze korekční funkcí, nikoliv již svázáním do jediného BTT. Posun v krajině je prováděn naprosto shodně s metodou Čtyř čtverců (3.2).

Výhoda této metody je převážně v úspoře paměti, která by musela být vynaložena na BTT, vytvořený z krajiny v plném detailu. Bohužel obsluha dělení čtverců vyžaduje čas, o který přijdeme při každém průchodu ohodnocování BTT.

3.7 Výběr nejlepší metody

Metody byly rozděleny podle svých vlastností do skupin, které určují oblasti jejich použití. Byly popsány jejich výhody a nevýhody. Nyní by měl následovat teoretický výběr nejlepší metody pro nekonečnou mapu vytvořenou ROAM algoritmem. Bohužel však nelze říci jednoznačně, která metoda je nejlepší.

Do skupiny první (a), metody pracující s libovolnou mapou, nepatří naprosto přesně žádná metoda. S určitým omezením se k ní blíží metody z odstavce 3.4 a 3.5. Omezení těchto metod je, že výsledná mapa, po posunu krajiny, musí mít opět čtvercový tvar o velikosti mocniny dvou. Z popisu těchto metod lze usoudit, že časově méně náročná bude metoda 3.5 –Modifikovaná metoda čtyř čtverců. A prostorová náročnost, tedy nároky na paměť, budou mít obě metody přibližně shodnou.

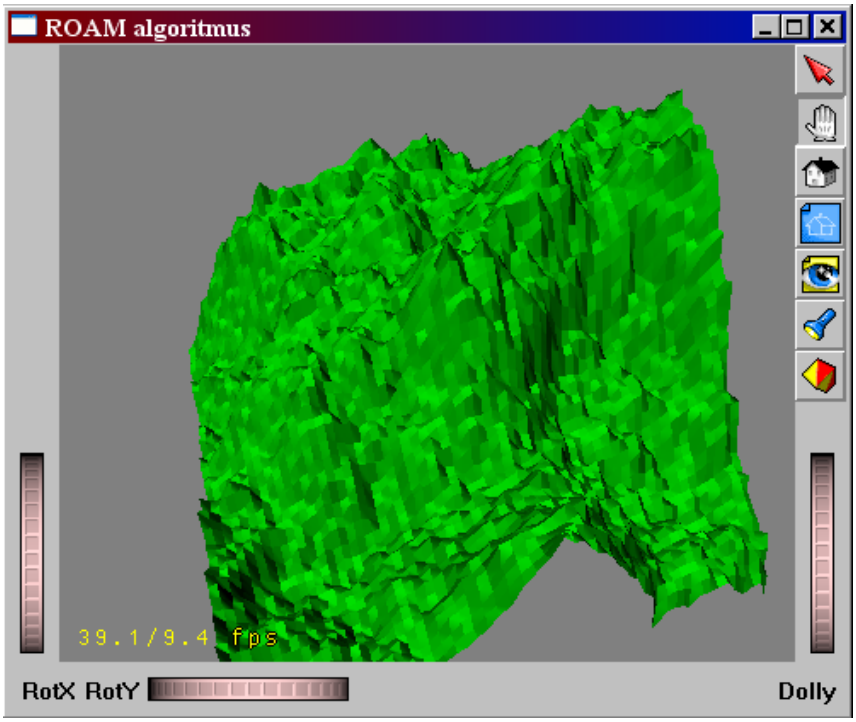
Do skupiny druhé (b) patří metody z odstavců 3.2 a 3.3. Zde je již hodnocení metod náročnější. Metoda Čtyř čtverců (kapitola 3.2), je základní metoda tvorby nekonečné krajiny pomocí ROAM. Ostatní metody z ní přímo či nepřímo vycházejí. Je velmi jednoduchá na pochopení i na vlastní provedení a dovoluje také mnohé způsoby implementace. Zatímco metoda 3.3 (Snižování levelu krajiny), je mnohem komplikovanější a implementačně náročnější. Obecně však lze říci, že metoda 3.2 je rychlejší, avšak paměťově náročnější oproti metodě 3.3.

Tato srovnání jsou pouze teoretická a nejlepším řešením, jak potvrdit jejich věrohodnost je pokusit se implementovat všechny metody a provést jejich následné výkonnostní a paměťové proměření. To je však bohužel velmi časově náročné a není to také přímo cílem mé práce. Já se pokusím pouze o implementaci a důkladné proměření metody čtyř čtverců, neboť jak již bylo řečeno dříve jedná se o základní metodu pro nekonečnou krajinu využívající ROAM algoritmus.

4 Implementace

4.1 ROAM algoritmus pro jeden čtverec mapy

Obecně je implementace popsána v kapitole 2.3. Zde jsou popsány některé možnosti přístupu k řešení tvorby ROAM algoritmus pro jeden čtverec mapy, tak jak byly popsány v mé ročníkové práci. Vycházíme ze situace zobrazení 3D krajiny bez ROAM algoritmu, pouze zobrazení výškové mapy.

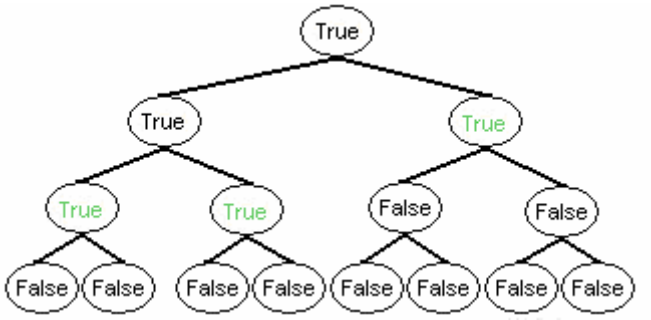


Obr. 19.: Zobrazení samotné výškové mapy
Průměr FPS pro mapu velikosti 128x128 je 10

Při implementaci ROAM algoritmu velmi záleží na metodě procházení BTT. Strom můžeme procházet od listu, od kořene (4.1.1), nebo jinými inteligentnějšími metodami zobrazení, jako například procházet vždy jen část stromu (4.1.2). Dále záleží na způsobu tvorby stromu, který může vést k veliké úspoře času (4.1.3). Následuje výčet metod, které mají jediný cíl – zlepšit výsledné parametry ROAM algoritmu. Patří mezi ně například ohodnocení stromu jen při větším posuvu kamery (4.1.4), specifikace vertexů pomocí indexování (4.1.5), a mnohé další. Bohužel není prakticky možné provést snížení počtu průchodů v dynamické části ze dvou na jeden.

4.1.1 Procházení stromu od kořene

Implementace procházení stromu od kořene velmi zprůhlední ohodnocování stromu. Při této metodě procházíme strom do hloubky (max. k listu), až do doby kdy se změní příznak zobrazení z A na N. Vznikne „čepička stromu“, ve které jsou všechny uzly se zobrazením A. Zobrazují se uzly na hraně této čepičky, což jsou uzly, jejichž synové mají příznak zobrazení N, ale samotné uzly mají příznak A.



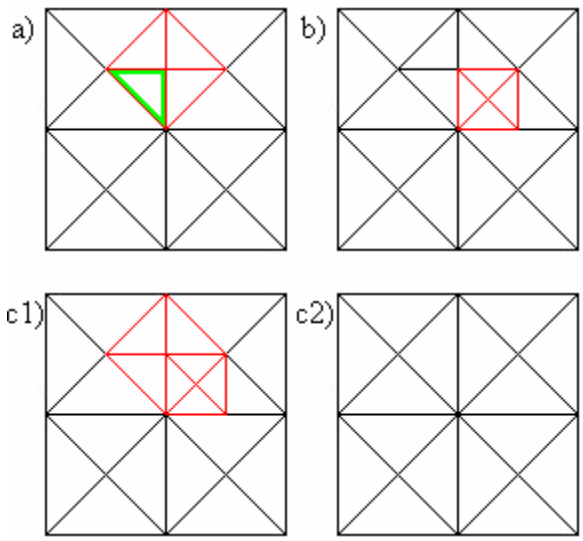
Obr. 20.: Strom ohodnocen od kořene
Uzly tvořící hranici čepičky, tedy ty které je třeba zobrazit jsou obarveny zeleně.

4.1.2 Průchod jen části stromu

Základní myšlenka je založena na faktu, že prohledávání celého BTT je časově velmi náročné. Východisko lze nalézt v rozdělení prohledávání stromu na několik menších částí. V každém kroku se provede prohledání například jen čtvrtiny BTT. Nárůst výkonu je příznivý, přičemž nejsou pozorovatelné žádné vedlejší efekty rušící zobrazování. Detailnější proměření výkonnostních parametrů této metody je uveden v kapitole Měření.

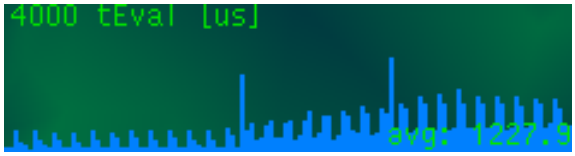
Při implementaci však narážíme na mnohé problémy, které vedou k důležitým změnám kódu. První změnou je zrušení nastavování příznaku zobrazení na N ve funkci Naplnění pole indexů. Důvodem je právě změna ohodnocování. Ohodnocujeme pouze část BTT, tudíž zbylé části musí zůstat se stávajícím příznakem zobrazení. Druhá změna je způsobena právě nemazáním příznaku zobrazení z minulého kroku, tudíž je třeba při ohodnocování tento fakt akceptovat. Při nastavování příznaku na hodnotu A nedochází k žádné změně programu. Ale při nastavování na hodnotu N je nutné zkontrolovat, zda nebyla v minulém kroku A. Pokud ano, je nutné zkontrolovat také všechny trojúhelníky tvořící s aktuálním uzlem diamond strukturu, a také jejich syny zda neměly příznak zobrazení roven A. Touto kontrolou se však dostáváme také do části BTT, kterou nemáme aktuálně ohodnocovat. Pokud bychom automaticky nastavovali všechny syny na příznak N, docházelo by

k velmi nevhodnému efektu problikávání, způsobenému právě změnou sousedních částí BTT. Tomuto efektu lze zabránit prováděním výpočtu, který rozhoduje (pomocí „rozhodovacího poměru“) zda daný uzel zobrazit či nikoliv. Pokud dospěje k rozhodnutí „ano“, je nutné provést zpětné nastavení otců na příznak A (viz Obr. 21). Tuto kontrolu nelze přeskočit neboť nikdy nevíme, zda daný uzel zobrazit či nikoliv (rozhodnutí záleží na synech). Kontrolu lze pouze urychlit – viz odstavec 4.1.3.



Obr. 21.: Ohodnocování pouze části BTT
Ohodnocujeme část BTT – levý horní čtverec (obr. a) - zelený trojúhelník).
S aktuálním trojúhelníkem tvoří diamond strukturu čtyři trojúhelníky (červeně označené v obr. a)).
Provedeme kontrolu příznaku zobrazení, a dále i pro jejich syny tvořící další diamond strukturu (obr. b)).
Pokud je výsledek „rozhodovacího poměru“ roven A je výsledné zobrazení naznačeno na obr. c1) ,
naopak výsledku N odpovídá obr. c2).

Po implementaci se při běhu programu vyskytl zajímavý jev. U měření doby trvání funkce, která provádí ohodnocování BTT, se ukázalo, že výsledné doby ohodnocení čtverců krajiny jsou hodně odlišné. Tento jev se vyskytl pouze při částečném procházení stromu. Vše je způsobeno tvarem krajiny. Jelikož je ohodnocování BTT rozděleno na čtyři části, záleží na tvaru krajiny, jak dlouho bude funkce ohodnocování krajiny trvat. Pokud je část terénu rovinatá a část hornatá dojde k nerovnoměrnému zatížení. Výsledkem je pak graf, v němž se nacházejí periodicky se opakující fragmenty.



Obr. 22.: Různé doby funkce ohodnocování BTT
Jsou zde viditelné opakující se „vlnky“, kde každá je složena ze čtyř volání funkce ohodnocení - jedna „vlna“ tvoří ohodnocení celého BTT.
Počet trojúhelníků 1.čtverec: 390, 2. čtverec: 320, 3. čtverec: 46, 4. čtverec: 92.

4.1.3 Změna ohodnocování stromu

Je důležité mít stále na paměti možné způsoby dělení trojúhelníků. Dělení diamond znamená, že máme skupiny složené z jednoho, dvou nebo čtyř trojúhelníků. Žádné jiné kombinace nejsou možné. Přičemž vždy celá skupina je nebo není zobrazena. Je však zbytečné pro každý trojúhelník testovat, zda jsou otcové také nastaveni na příznak zobrazení = A, neboť vždy dva trojúhelníky mají stejného otce. Pokud je vytváření struktur systematické, víme bez jakéhokoliv testování, kteří synové mají společného otce. Lze tak ušetřit polovinu kontrol (jednotky FPS).

Podobné urychlení lze využít při procházení pouze části stromu, kde se však nastavují synové diamond struktury na hodnotu zobrazení = N. Detailní popis je v odstavci 4.1.2. Jelikož každý uzel nastavuje svoje dva syny, nelze urychlit nastavování hodnoty zobrazení na N. Pokud však jeden ze synů bude mít příznak zobrazení (po výpočtu „rozhodovacího poměru“) nastaven na hodnotu A. Znamená to, že všichni jeho otcové mají také hodnotu A, z čehož vyplývá, že není potřeba pokoušet se nastavovat ostatní uzly tvořící s aktuálním diamond strukturu na hodnotu zobrazení = N.

4.1.4 Změna vytváření stromu

Zde jsem pouze pozměnil myšlenku, že nový bod, který je ukládán do SoCoordninate není vypočítáván postupně při vytváření stromu. Dříve se muselo testovat, zda byl bod již uložen v předchozím kroku. Nyní jsou všechny body uloženy před tvorbou stromu a následně je pouze zjišťována jejich pozice (z důvodů indexování). Tato jednoduchá změna měla za následek mnohonásobné 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

4.1.5 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 na určitou malou oblast, což má za následek zvýšení FPS. Z testů vyplývá, že se projeví velmi příznivě, zvláště v situacích, kdy je kamera prakticky na místě a pouze se otáčí (hráč se rozhlíží po krajině).

4.1.6 Rozdíl ve způsobu specifikování vertexů[4]

Použitím indexované verze (SoIndexTriangleStripSet), která obsahuje dvě tabulky:

1. specifikuje souřadnice bodu,
2. obsahuje indexy jednotlivých bodů.

Ušetříme místo v paměti a dokonce není tato úspora ani poznat na výkonu, neboť je již hardwarově podporována (už od karty GeForce256).

4.2 Nekonečná krajina pomocí metody Čtyř čtverců

Jedná se o rozšíření dosud naprogramovaných algoritmů o možnost práce se čtyřmi mapami. A dále o přidání kódu zajišťujícího načítání nových a zahazování starých čtverců. Práce se čtyřmi čtverci mapy požaduje rozšíření všech potřebných datových struktur (viz odstavec 2.3) a následnou úpravu kódu, do podoby umožňující práci s těmito strukturami. V zásadě lze říci, že se všechny stávající datové struktury rozšířily čtyřnásobně oproti stávající velikosti.

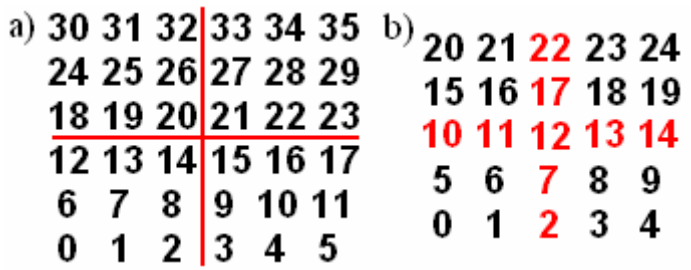
4.2.1 Nový vzhled BTT

Bylo zapotřebí upravit BTT do podoby zobrazeném na Obr. 10. Přidáním tří umělých uzlů dojde ke spojení čtverců do jediného BTT. Tyto uzly však znamenají určité komplikace, neboť je nutné provádět jejich detekci při výpočtech (tyto uzly musí mít vždy příznak zobrazení roven A). Komplikace jsou však mnohonásobně vyváženy výhodami, které přináší jediný BTT.

[4] Zdroj: PEČIVA, Jan. Tutoriál o knihovně Open Inventor. Dokument dostupný na URL http://www.root.cz/clanky/open-inventor/ (14. 3. 2006).

4.2.2 Operace s daty

Velmi důležité je provedení indexování bodů výškové mapy. Je zde nutno uvědomit si, že metoda Čtyř čtverců patří do kategorie b), požaduje tedy čtvercovou krajinu mocniny dvou. Z čehož plyne způsob indexace krajiny (viz Obr. 23).



Obr. 23.: Indexace bodů výškové mapy

Obrázek a) znázorňuje chybnou indexaci. Je sice složen ze čtyř map mocnin dvou, ovšem body na hranici čtverců se neshodují. Vznikne tak oblast (červená čára), která není popsána žádným BTT.

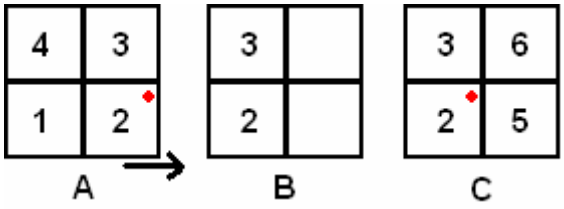
Obrázek b) ukazuje správně indexovanou mapu – body na hranici čtverců patří do obou čtverců (červeně zobrazeny), trojúhelníky mezi všemi body jsou popsány čtyřmi BTT.

Po úpravě kódu tak, aby zobrazoval čtyři mapy vedle sebe je zapotřebí vytvořit funkci provádějící načítání a zahazování čtverců mapy. Které navíc může rozdělit do více kroků, tak jak bylo popsáno v odstavci 3.2. Tato operace je složena ze tří částí. V první části, provedeme zjištění ke které hraně mapy se přibližujeme, zjistíme tak, které čtverce mapy zachovat a které nahradit novými. V druhé části provedeme překopírování dat, určené k zachování, z aktuální pozice na pozici určenou potřebným posunem mapy. Na závěr provedeme naplnění prázdných částí datových struktur novými daty, získanými z výškové mapy.

Nezbytné je určení, které datové struktury při tomto procesu nahrazovat. Je totiž zbytečné přepočítávat data, která se ve výsledku nemění. Změnu je zapotřebí provést v poli Bodů (Z souřadnice), Vektorů (normály bodů), Korekce a v rozdílové mapě. Žádná další datová struktura nemusí být modifikována. Například BTT zůstává bez změny, neboť složky každého uzlu jsou: indexy na tři body trojúhelníku (bez změny), indexy na otce a syny (bez změny), a příznak zobrazení (bez změny).

4.2.3 Operace s kamerou

Velmi důležité je provést s posunem mapy posun kamery opačným směrem, aby vznikl pocit pohybu v krajině. Pokud bychom tuto operaci neprovedli, ocitl by se hráč v neznámé krajině a navíc by byl opět na hranici mapy. Když provedeme posunutí kamery o velikost jednoho čtverce zpět, ve směru proti přidávání čtverců, zůstane hráč na stejném místě v krajině, jen se proti němu objeví nová krajina. Vznikne tak dojem neustálého pohybu vpřed, ačkoliv hráč nikdy neopustí prostor vymezený čtyřmi čtverci mapy.



Obr. 24.: Ukázka posunu krajiny
A – posun krajiny vpravo (hráč je na hranici čtverců 2,3 – červená tečka)
B – kopie dat z místa čtverců 2, 3 na místo čtverců 1, 4
C – naplnění novými daty 5, 6 a posun kamery proti směru pohybu

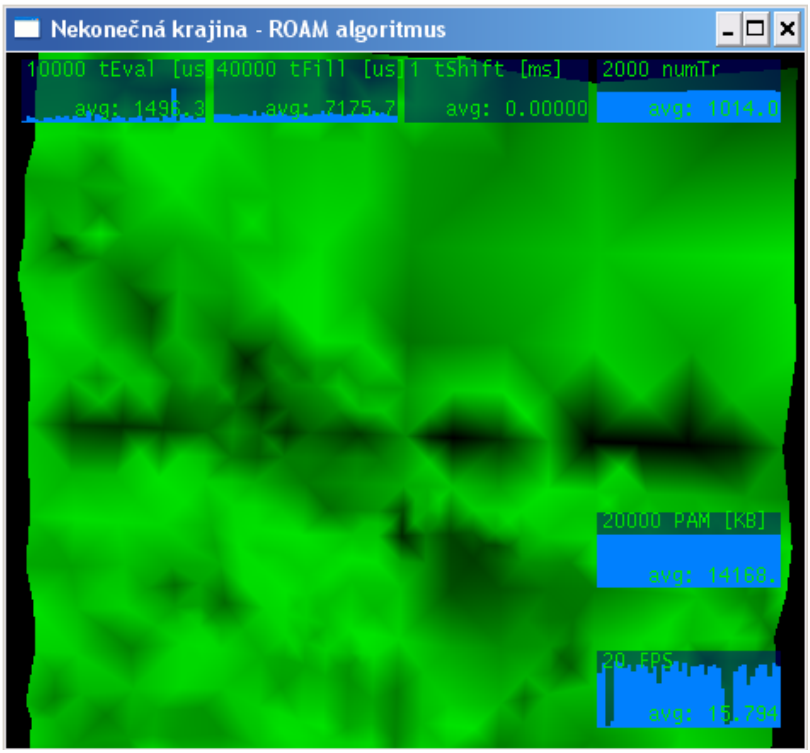
4.3 Další části implementace

4.3.1 Měřiče parametrů

Pro zobrazení měřičů parametrů jsem využil knihovnu SoPerfGraph.h, která byla již dříve implementována v projektu Terrain Engine[5]. Provedl jsem pouze malé úpravy této knihovny, spočívající v přidání označení k jednotlivým grafům z důvodu zlepšení přehlednosti, a změnu popisu materiálů. V levém horním rohu grafu je vidět maximální hodnota rozsahu, tato hodnota se upravuje automaticky podle právě měřených hodnot. Hned za touto hodnotou je uvedeno krátké označení měřené veličiny spolu s jednotkami. V dolní části grafu lze sledovat okamžitou hodnotu veličiny. Měření je prováděno za pomocí šesti grafů – každý pro jeden parametr. Grafy v horní části indikují vnitřní vlastnosti algoritmů. Naopak větší grafy v dolní části znázorňují vlastnosti viditelné z vnějšku aplikace. Vlevo nahoře je umístněn graf znázorňující měření doby průchodu funkcí ohodnocení (Evaluation). Vedle něj se nachází graf měření doby funkce plnění indexů bodů

[5] Zdroj: HAVLÍČEK, Martin. Knihovna pro práci s výškovými mapami [Diplomová práce]. VUT v Brně, Fakulta informačních technologií.

(Fill_arrayOfIndex). Napravo od něj je znázorněna doba potřebná pro průchod na další čtverce mapy, tedy posun krajiny. Posledním grafem v horní části okna je graf zobrazení počtu renderovaných trojúhelníků. První dva grafy jsou nefunkční při zobrazování krajiny bez ROAM algoritmu. Při tomto zobrazení je počet trojúhelníků konstantní (roven maximálnímu počtu trojúhelníků). V dolní části okna se pak nachází graf aktuální FPS (spodní) a graf obsazené paměti (spodní) – její výpočet je v příloze třetí. Obsazená paměť je při zobrazení bez ROAM algoritmu konstantní.



Obr. 25.: Ukázka měřičů parametrů

Měření FPS, neboli počtu snímků za vteřinu, je prováděno změřením doby potřebné k provedení jednoho snímku a následným výpočtem: jedna dělená touto dobou. Touto metodou získáme FPS parametr v reálných číslech, tedy přesněji nežli počítání kolik snímků se vleze do jedné sekundy.

Měření paměťových nároků aplikace je složeno z výpočtu velikosti statických dat a následné přičtení velikosti dat měnících se dynamicky v závislosti na pohybu v krajině. Výpočet dat je přiložen v příloze třetí. Statická data záleží pouze na velikosti zobrazované krajiny. Data dynamická jsou složena pouze z pole Indexů, které se mění vždy při ohodnocení BTT. Tedy podle počtu zobrazených trojúhelníků.

4.3.2 Ovládání pomocí kláves

Pro pohyb v krajině podobný leteckým simulátorům je zapotřebí, zajistit odpovídající reakce na stisk předvolených kláves. Při tvorbě této funkce jsem se inspiroval ročníkovým projektem[6], ve kterém řešil podobný problém ve hře Tank Hunters. V mém případě jsem vypustil řešení zrychlení, prohloubil jsem však ovládání z 2D na 3D prostor.

Pro zachytávání stisku kláves jsou využity knihovny SoEventCallback a SoKeyboardEvent. Za pomoci první knihovny si zaregistrujeme funkci, která bude volána pokaždé, když je stisknuta některá klávesa. Druhá knihovna pak informuje o tom, která klávesa byla stisknuta. Využití těchto knihoven však vedlo k radikální změně, neboť z nějakého důvodu nejsou schopny pracovat s renderovacím oknem vytvořeným za pomocí SoWinExaminerViewer, ale pouze s SoWinRenderArea.

Pro samotný pohyb ve scéně jsou využity metody perspektivní kamery camera->orientation a camera->position. Za pomocí těchto dvou operací lze jednoduše kameru otáčet či posunovat vpřed a vzad. Spojíme-li tedy odchytávání kláves a ovládání kamery, dostaneme funkci řešící pohyb v krajině. Vlastní ovládání programu je přiloženo v příloze, a také zobrazováno v konzoli při zapnutí programu. V příloze jsou popsány parametry určující vlastnosti krajiny, které jsou nastavovány při kompilaci. Jedním z parametrů je také konstanta posun, určující jakou rychlostí se kamera v krajině pohybuje (kolik pixelů na jeden stisk klávesy).

4.3.3 Filtrace výškové mapy

Tato funkce je využívána v případě, kdy nenačítáme výškovou mapu ze souboru, ale provádíme její generování až při běhu programu. Generování je prováděno Midpoind displacement algoritmem, který je převzat z tutoriálu Ing. Jana Pečivy[7]. Tento algoritmus funguje na jednoduchém principu: Rekurzivní dělení hrany čtverce na polovinu, přičemž výška prostředního bodu je určena náhodně v daném intervalu.

---

[6] Zdroj: VÝCHOPEŇ, Daniel. Tank Hunters [Ročníkový projekt]. VUT v Brně, Fakulta informačních technologií.

[7] Zdroj: PEČIVA, Jan. Tutoriál o knihovně Open Inventor. Dokument dostupný na URL http://www.root.cz/clanky/open-inventor/ (14. 3. 2006).

Filtrování znamená určité vyhlazení tvaru krajiny. Je prováděno nad body výškové mapy a je určeno počtem iterací a velikostí filtrovacího koeficientu. Počet iterací znamená, kolikrát algoritmus projde výškovou mapu a kolikrát dojde k její změně. Filtrovací koeficient pak určuje, jak velká změna je s body provedena, neboli jak moc se sousední body ovlivňují. Průměrná hodnota koeficientu se pohybuje v rozmezí 0.6 (velká změna) 0.9 (malá změna). Změnu výšky jednoho bodu lze jednoduše popsat za pomocí rovnice:

BodVýškovéMapy = SousedníBodVýškovéMapy * (1-koeficient) + BodVýškovéMapy *koeficient

V případě nekonečné krajiny vytvořené za pomocí metody Čtyř čtverců, bylo nutno použít dvojí filtrování. Jednak klasické filtrování krajiny pro vyhlazení tvaru, ale také filtrování krajiny pro vyhlazení přechodů mezi čtverci mapy. Jak již bylo řečeno dříve, je nekonečnost krajiny vytvořena přidáváním nových čtverců mapy. Tyto čtverce mapy lze vyfiltrovat z důvodů vyhlazení tvaru, což však nezaručí plynulou návaznost výšky krajiny s již existujícími čtverci. Je proto nutné tuto návaznost nějakým způsobem zajistit. Jednou z možností je použití jiného generujícího algoritmu a druhou je provést filtrování. Pokud bychom provedli při každém posunu krajiny filtrování celé krajiny, došlo by ke změně krajiny, kterou již hráč prošel, což samozřejmě není možné. Musíme provést filtraci krajiny pouze na hranách čtverců mapy tak, aby byla co nejméně změněna krajina, kterou již hráč prošel. Toto filtrování je prováděno v blízkém okolí hrany. Mění se tedy obě přiléhající mapy. Aby byla mapa kterou hráč prochází měněna co nejméně, je při každé iteraci snižována oblast filtrace na této mapě směrem k hraně. Tímto způsobem získáme informaci o tvaru krajiny, její změna je však minimální.

4.3.4 Obarvení krajiny

Pro zvýšení efektu vzhledu krajiny bylo provedeno obarvení krajiny třemi různými barvami. Byly použity barvy:

- zelená – znázorňující trávu
- hnědá – hornatá krajina
- bílá – zasněžené vrcholky hor

Provedením obarvení však dojde k zpomalení ROAM algoritmu. Je to způsobeno nutností nastavování indexů materiálu při každém novém ohodnocení. Pokud provedeme ohodnocení stromu, dostaneme nové indexy trojúhelníků, které mají být zobrazeny. Ke každému takovému bodu je nutno přiřadit index materiálu, který udává barvu daného bodu. Přidáním barev získáme

další činnost, kterou je zapotřebí provádět při každém ohodnocení stromu. Zpomalení ROAM algoritmu je pokusy zjištěno asi 26%, dojde ovšem také ke zpomalení vykreslování bez ROAM algoritmu, které je asi 19%. Rozdíl hodnot však vychází v neprospěch ROAM algoritmu.

Přidání barev u krajiny bez ROAM algoritmu je provedeno stejným postupem. Tento krok je však proveden pouze jednou, stejně jako nastavení indexů bodů. Při pohybu krajinou již nedochází k přeindexování. K nastavení nových indexů dochází pouze při posunu krajiny na nové čtverce – získáme nový tvar krajiny. Je proto nutné změnit zabarvení terénu podle výšky.

5 Měření

Měřením ROAM algoritmu označujeme zjišťování parametrů při zobrazování 3D krajiny. Mezi základní parametry patří FPS (frame per second), což značí počet zobrazených snímků za sekundu. Dále pak velikost obsazené operační paměti. Zjišťování parametru FPS lze provádět měřičem zabudovaným do knihovny Coin, který v levém dolním rohu vypisuje parametry FPS. Toto měření je však pouze orientační, neboť tento parametr má velké výkyvy hodnot a my máme k dispozici pouze okamžitou hodnotu, nikoliv průměrnou. Proto je nutné využit měřič parametrů z knihovny Terrain Engine pro přesnější zjištění hodnot (viz kapitola 4.3.1). Stejný měřič je využit i pro zjišťování obsazené paměti. Toto měření je neméně důležité, neboť paměťové nároky jsou jedním z hlavních nedostatků ROAM algoritmu.

Měření samotné je velmi náročné na detaily provedení. Důvodem je samotná funkce ROAM algoritmu, který je velmi citlivý na tvar výškové mapy, na vzdálenost kamery od daného bodu krajiny, či rychlosti pohybu kamery. Musíme provádět opakovaná měření pro různé druhy krajiny. Provedeme tedy umístění kamery do jedné konstantní pozice a nastavíme její posun na konstantní rychlost. Poté již stačí pouze sledovat měnící se parametry pro různé výškové mapy. Výsledné hodnoty parametrů by měly odpovídat průměrným hodnotám v dané situaci (např. pro velikost mapy 32x32 ve vzdálenosti kamery 50 pixelů).

Měření času provádíme za pomocí funkce: SbTime::getTimeOfDay().getValue(), která vrací hodnotu času v mikrosekundách. Samotné volání této funkce trvá asi jednu mikrosekundu, měření tedy není příliš ovlivněno samotným zjišťováním času. Dobu trvání měřeného úseku zjistíme z rozdílu dvou časů, kdy jeden je zjištěn před voláním funkce a druhý ihned po dokončení.

5.1 Počítače na kterých bylo prováděno měření

Počítače pro měření	Procesor [GHz]	Paměť [MB]	Harddisk [GB]	Grafická karta [MB]
Notebook UMAX 575t (slabá grafická karta, silný procesor)	Intel Celeron 2	512	20 (IDE 100)	SIS 650 64 MB sdílené operační paměti (ovladač: 6.14.10.3650, datum: 4.1.2005)
Stolní počítač 1 (silná grafická karta, slabý procesor)	AMD Duron 0,8	256	80 (IDE 100)	NVIDIA GeForce 3 Ti 200 64MB (ovladač: 8.1.8.5, datum: 10. 10. 2005)
Stolní počítač 2 školní (vyvážené parametry)	Intel Pentium 4 2,8	504	10 (IDE 100)	Intel 828656 (ovladač: 6.13.1.3485, datum: 13. 3. 2003)

Tabulka 1.: Parametry počítačů pro měření.

5.2 Druhy testů

Nastavení parametrů ROAM algoritmu (detailní popis parametrů viz příloha):

- TERRAIN DIMENSION - velikost jednoho čtverce mapy (tedy velikost celé krajiny je dvojnásobná – příklad TERRAIN_DIMENSION = 65, krajina je 129x129).
- TERRAIN HEIGHT_Q - max. výška a hloubka krajiny.
- CONST DISPLAY - konstanta podmínky zobrazení.
- CONST INSENSITIVITY - max. změna polohy kamery, bez řešení BTT.
- DISTANCE POINT - konstanta vzdálenosti bodů v krajině.
- NUM_DIVISION_TREE – na kolik částí je rozdělen jeden průchod BTT.

Nastavení měření:

- POCATECNI UMISTNENI KAMERY:
  - X souřadnice
  - Y souřadnice
  - Z souřadnice
- POSUN - po kolika pixelech se pohybuje kamera.
- Směr posunu kamery – vždy plynulý pohyb směrem vpřed v ose Y.

Výsledky měření parametru FPS závisí na mnoha parametrech. Jedním z nich, který možná není viditelný na první pohled, je velikost zobrazovaného okna. Přičemž platí nepřímá úměrnost – čím větší je okno, tím menší je hodnota FPS. Mnou prováděná měření jsou pro okno o velikosti 400 x 400 pixelů.

Test	Parametry									
	TERRAIN DIMENSION	TERRAIN HEIGHT	CONST DISPLAY	NUM DIVISION TREE	CONST INSENSIT-IVITY	DISTANCE POINT	INITIAL PLACE OF CAMERA			MOVE CAMERA
							X	Y	Z	
A	65	30	70	1	1	1	64	64	130	2
B	129	50	150	1	1	1	128	128	258	2
C	257	90	230	1	1	1	256	256	514	2
D	512	110	300	1	1	1	512	512	1026	2
Change height of camera										
E	65	30	70	1	1	1	64	64	30	2
F	129	50	150	1	1	1	128	128	50	2
G	257	90	230	1	1	1	256	256	90	2
H	512	110	300	1	1	1	512	512	110	2
Change const display										
I	65	30	40	1	1	1	64	64	130	2
J	129	50	70	1	1	1	128	128	258	2
K	257	90	140	1	1	1	256	256	514	2
L	512	110	220	1	1	1	512	512	1026	2
Change num division tree										
M	65	30	40	4	1	1	64	64	130	2
N	129	50	70	4	1	1	128	128	258	2
O	257	90	140	4	1	1	256	256	514	2
P	512	110	220	4	1	1	512	512	1026	2
Change other parameter										
Q	129	50	70	1	1	3	128	128	258	2
R	129	50	70	1	1	1	128	128	258	10
S	129	50	70	1	10	1	128	128	258	1

Tabulka 2.: Parametry testů.

Testy A, B, C, D jsou pro krajinu z velké výšky, kdy krajina není zobrazovaná ROAM algoritmem ve velkém detailu, jsou zobrazeny pouze hrubé kontury kopců – tedy pro malý počet trojúhelníků. Testy E, F, G, H jsou pro krajinu z malé výšky, v malé blízkosti kamery je krajina ve velkém detailu, podobná krajině bez ROAM – tedy velký počet trojúhelníků. Následují testy I, J, K, L kdy se sleduje vliv konstanty zobrazení. Další jsou M, N, O, P, zde je prověřován vliv funkce ohodnocování na celkový výkon systému. Poslední skupinou testů jsou testy smíšené, kde jsou prověřovány méně významné parametry, jako jsou: změny uspořádání bodů v krajině (body jsou vzdáleny o 3 pixely), test rychlejšího posunu kamery (kdy kamera „skáče“ po 10 pixelech). Dále je test při pomalém posunu kamery (1 pixel) a nastavené konstantě necitlivosti na hodnotu 10 – což znamená že BTT je přepočítáván při každém 10 volání funkce.

5.3 Výsledky měření

Měření byla prováděna na třech počítačích (sloupce) a bylo provedeno devatenáct testů (řádky). Měřené veličiny byly zaznamenány jako průměrná hodnota z několika provedených pokusů.

Krajina s ROAM algoritmem

Test	Počítač					
	I.		II.		III.	
	FPS	Paměť [kB]	FPS	Paměť [kB]	FPS	Paměť [kB]
A	26,0	3560,0	26,0	3560,0	76,0	3560,0
B	12,0	14212,0	12,0	14203,0	35,0	14215,0
C	8,5	56710,0	6,2	56683,0	19,5	56700,0
D	6,5	226440,0	2,9	226430,0	15,0	226430,0
E	12,0	3618,0	10,0	3640,0	29,0	3622,0
F	4,2	14434,0	3,5	14430,0	7,5	14440,0
G	2,5	57400,0	1,8	57320,0	4,5	57400,0
H	2,0	226790,0	1,3	226810,0	3,5	226820,0
I	37,0	3551,0	35,0	3552,0	108,0	3552,0
J	29,0	14176,0	22,0	14175,0	78,0	14178,0
K	14,0	56651,0	8,5	56652,0	33,0	56652,0
L	7,6	226410,0	3,1	226420,0	18,0	226420,0
M	40,0	3552,0	37,0	3552,0	112,0	3551,0
N	29,0	14180,0	23,0	14179,0	79,0	14175,0
O	15,0	56650,0	9,5	56655,0	35,0	56657,0
P	8,0	226430,0	3,1	226420,0	19,0	226420,0
Q	40,0	14168,0	25,0	14168,0	104,0	14168,0
R	30,0	14174,0	20,0	14177,0	80,0	14177,0
S	40,0	14175,0	70,0	14175,0	130,0	14177,0

Tabulka 3.: Výsledky testů – krajina s ROAM algoritmem.

Z testu vyplývají některé souvislosti, které jsou podloženy teoretickými předpoklady:

- Testy E, F, G, H ukazují oproti A, B, C, D jasný pokles FPS. To je způsobeno větším detailem krajiny – tedy větším počtem trojúhelníků. S tím je spojen také nárůst spotřebované paměti.
- Opačný výsledek ukazují testy I, J, K, L, kde snížením konstanty zobrazení dochází k nárůstu FPS.
- U testů M, N, O, P, které demonstrují změněný přístup k ohodnocování BTT, je vidět mírný nárůst výkonu. Nárůst však není nikterak ohromující. Je to způsobeno pouze mírným snížením časů funkce provádějící ohodnocení. Přičemž tato funkce zabírá pouze malou část z celkové doby rendrování.

Krajina bez ROAM algoritmu

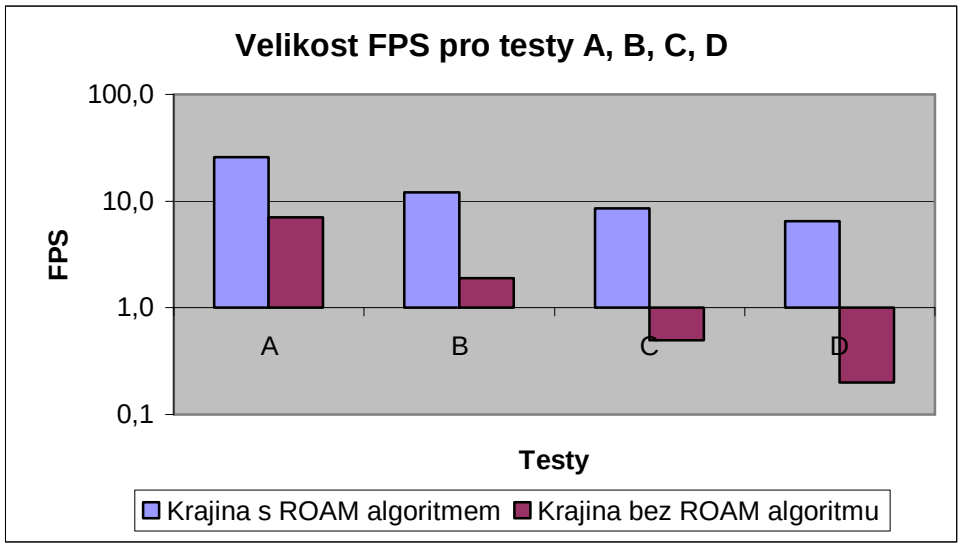
Test	Počítač					
	I.		II.		III.	
	FPS	Paměť [kB]	FPS	Paměť [kB]	FPS	Paměť [kB]
A	7,0	81,0	29,0	81,0	54,0	81,0
B	1,9	322,0	10,5	322,0	17,5	322,0
C	0,5	1 283,0	3,0	1 283,0	5,1	1 283,0
D	0,2	5 125,0	0,8	5 125,0	1,3	5 125,0
E	10,0	81,0	27,0	81,0	68,0	81,0
F	3,1	322,0	10,0	322,0	27,5	322,0
G	0,7	1 283,0	3,0	1 283,0	8,1	1 283,0
H	0,2	5 125,0	0,7	5 125,0	2,2	5 125,0
I	7,0	81,0	30,0	81,0	54,0	81,0
J	2,0	322,0	10,5	322,0	18,0	322,0
K	0,5	1 283,0	3,2	1 283,0	4,9	1 283,0
L	0,2	5 125,0	0,9	5 125,0	1,2	5 125,0
M	7,0	81,0	29	81,0	54,0	81,0
N	2,0	322,0	10,5	322,0	18,0	322,0
O	0,5	1 283,0	3,6	1 283,0	5,3	1 283,0
P	0,2	5 125,0	0,9	5 125,0	1,3	5 125,0
Q	2,0	322,0	11	322,0	18,0	322,0
R	2,0	322,0	10,5	322,0	18,0	322,0
S	2,0	322,0	12	322,0	18,0	322,0

Tabulka 4.: Výsledky testů – krajina bez ROAM algoritmu.

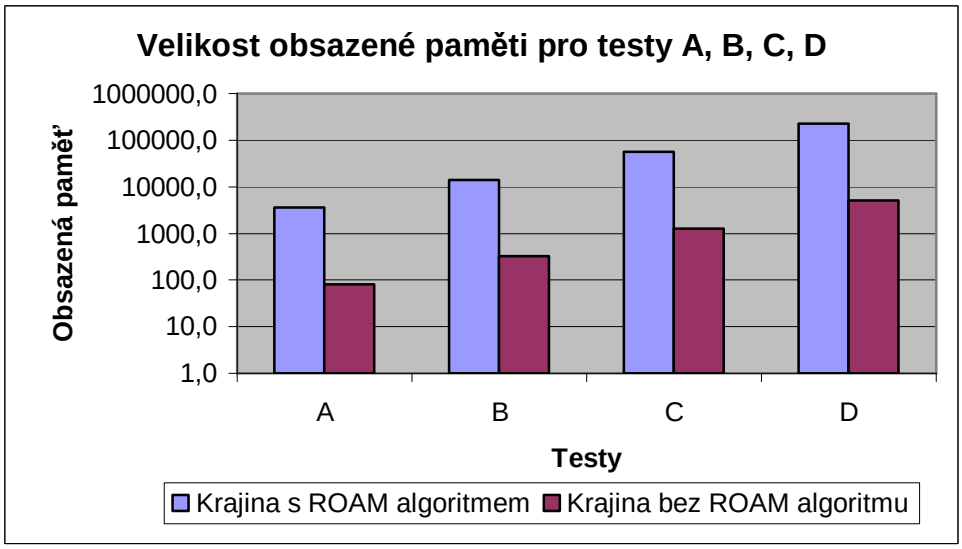
Také u krajiny zobrazované bez ROAM algoritmu, tedy v plném detailu, jsou viditelné jasné souvislosti mezi naměřenými parametry:

- Měření obsazené paměti je pro danou velikost krajiny shodné. To je způsobeno neměnným vzhledem krajiny. Jsou zde zastoupeny pouze statické části algoritmu. Počet trojúhelníků je vždy konstantní:
  - Krajina 129 x 129 bodů obsahuje 32768 trojúhelníků.
  - Krajina 257 x 257 bodů obsahuje $1,3 * 10^5$ trojúhelníků.
  - Krajina 513 x 513 bodů obsahuje $5,2429 * 10^5$ trojúhelníků.
  - Krajina 1025 x 1025 bodů obsahuje $2,0972 * 10^6$ trojúhelníků.
- Nejsou rozdíly FPS parametru mezi testy A, B, C, D a I, J, K, L, M, N, O, P. Je to způsobeno zadáním testů, ve kterých se mění parametry ROAM algoritmu, nikoliv parametry krajiny. Není tedy důvod, aby byly výsledky měření odlišné.
- Jediné testy, které jsou odlišné jsou E, F, G, H. To je způsobeno změnou pozice kamery, která je umístěna blíže ke krajině a směřuje směrem dolů (ke krajině). Proto je viditelná pouze část krajiny. Dochází tedy k ořezání krajiny zobrazovacím oknem – nejsou viditelné všechny trojúhelníky. Což má za následek nárůst hodnoty parametru FPS.

Pokud zobrazíme pouze první čtyři testy, získáme graf závislosti algoritmů na velikosti krajiny. Což je základní parametr všech outdoor algoritmů.



Graf 1.: Velikost FPS pro testy A, B, C, D. Měřeno na počítači 1.



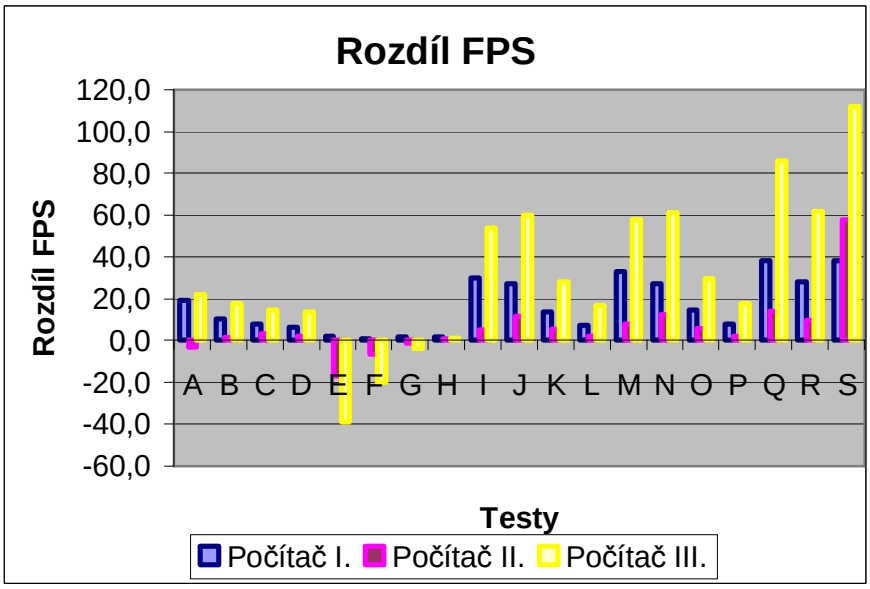
Graf 2.: Velikost obsazené paměti pro testy A, B, C, D. Měřeno na počítači 1.

Cílem celé problematiky ROAM algoritmu je snaha o zvýšení parametru FPS. Pokud provedeme jednoduchý výpočet jehož výsledkem bude rozdíl FPS s/bez ROAM algoritmu, dostaneme objektivní zhodnocení nárůstu (poklesu) tohoto parametru:

Rozdíl FPS

Test	Počítač		
	I.	II.	III.
A	19,0	-3,0	22,0
B	10,1	1,5	17,5
C	8,0	3,2	14,4
D	6,3	2,1	13,7
E	2,0	-17,0	-39,0
F	1,1	-6,5	-20,0
G	1,8	-1,2	-3,6
H	1,8	0,6	1,3
I	30,0	5,0	54,0
J	27,0	11,5	60,0
K	13,5	5,3	28,1
L	7,4	2,2	16,8
M	33,0	8,0	58,0
N	27,0	12,5	61,0
O	14,5	5,9	29,7
P	7,8	2,2	17,7
Q	38,0	14,0	86,0
R	28,0	9,5	62,0
S	38,0	58,0	112,0

Tabulka 5.: Rozdíl FPS = FPS s ROAM – FPS bez ROAM.



Graf 3.: Rozdíl FPS vynesený do grafu.

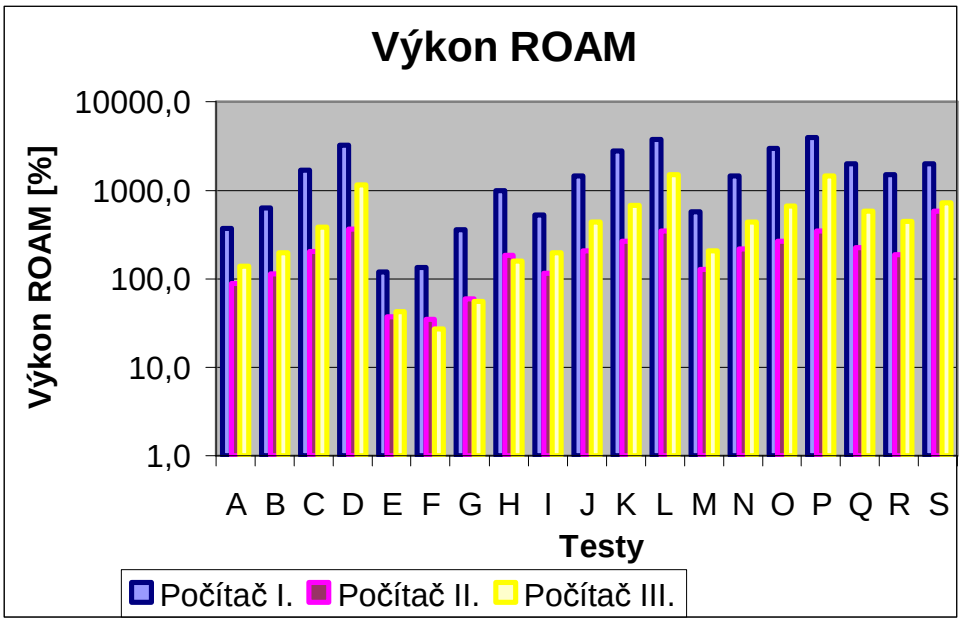
Rozdíl FPS znamená výpočet: FPS s ROAM algoritmem – FPS bez ROAM algoritmu. Je zřetelný pokles rozdílu FPS s narůstající velkostí mapy. Není to však způsobeno nečekaným chováním ROAM algoritmu, jehož kvality by měly vyniknout právě u větších map. Je to způsobeno snížením rozdílu mezi FPS. Při větší mapě je FPS velmi malé, a proto je i rozdíl malý.

Přehlednější pohled na zrychlení ROAM algoritmu přináší Tabulka 6. a Graf 4, ve které je znázorněn procentuální nárůst výkonu. Přičemž hodnota 100% je myšlena jako nulové zrychlení algoritmu:

Výkon ROAM

Test	Počítač		
	I.	II.	III.
A	371,4	89,7	140,7
B	631,6	114,3	200,0
C	1700,0	206,7	382,4
D	3250,0	362,5	1153,8
E	120,0	37,0	42,6
F	135,5	35,0	27,3
G	357,1	60,0	55,6
H	1000,0	185,7	159,1
I	528,6	116,7	200,0
J	1450,0	209,5	433,3
K	2800,0	265,6	673,5
L	3800,0	344,4	1500,0
M	571,4	127,6	207,4
N	1450,0	219,0	438,9
O	3000,0	263,9	660,4
P	4000,0	344,4	1461,5
Q	2000,0	227,3	577,8
R	1500,0	190,5	444,4
S	2000,0	583,3	722,2

Tabulka 6.: Výkon ROAM = FPS s ROAM * 100 / FPS bez ROAM.



Graf 4.: Výkon ROAM vynesený do grafu.

Výpočtem lze zjistit průměrné zrychlení algoritmu ROAM oproti krajině bez jeho použití:

- u počítače prvního dochází k zrychlení o 22,6 násobek FPS,
- u počítače druhého o 2,8 násobek,
- u třetího o 7,1.

Krajina s ROAM algoritmem - více měřených veličin

Test	Počítač					
	FPS	Paměť [kB]	Počet trojúhelníků	Doba ohodnocení [µs]	Doba plnění [µs]	Doba nové mapy [ms]
A	26,0	3 560,0	1 200,0	1 200,0	5 100,0	36,0
B	12,0	14 212,0	4 000,0	4 900,0	14 400,0	142,0
C	8,5	56 710,0	6 000,0	9 000,0	25 000,0	620,0
D	6,5	226 440,0	4 500,0	8 000,0	22 000,0	2 900,0
E	12,0	3 618,0	6 200,0	8 000,0	25 000,0	36,0
F	4,2	14 434,0	22 000,0	27 000,0	90 000,0	144,0
G	2,5	57 400,0	60 000,0	70 000,0	200 000,0	610,0
H	2,0	226 790,0	31 800,0	44 000,0	150 000,0	2 890,0
I	37,0	3 551,0	330,0	440,0	1 600,0	36,0
J	29,0	14 176,0	590,0	1 300,0	3 100,0	146,0
K	14,0	56 651,0	2 800,0	4 400,0	12 000,0	610,0
L	7,6	226 410,0	2 200,0	4 000,0	10 000,0	2 800,0
M	40,0	3 552,0	330,0	170,0	1 300,0	36,0
N	29,0	14 180,0	650,0	200,0	3 200,0	143,0
O	15,0	56 650,0	2 600,0	1 200,0	11 500,0	600,0
P	8,0	226 430,0	2 400,0	1 700,0	11 000,0	2 700,0
Q	40,0	14 168,0	14,0	50,0	100,0	153,0
R	30,0	14 174,0	700,0	1 100,0	3 100,0	149,0
S	40,0	14 175,0	650,0	1 100,0	3 000,0	145,0

Tabulka 7.: Výsledky testů – krajina s ROAM algoritmem – detailní proměření. Měření je prováděno pouze na počítači 1. - Notebook UMAX 575t.

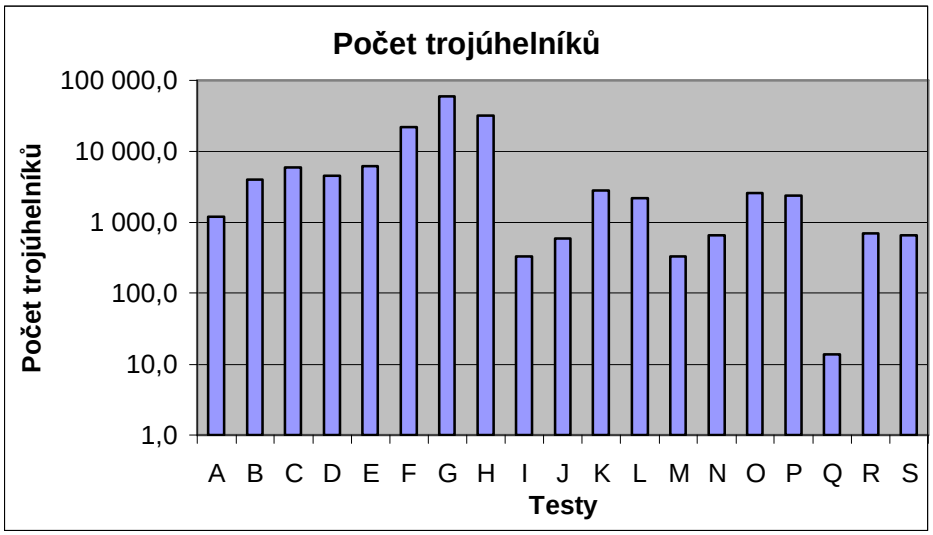
Pro detailnější pochopení ROAM algoritmu je nutné podívat se na vnitřní parametry programu, které lépe objasní souvislosti mezi funkcemi. Jsou zde k základním parametrům FPS a paměti přidány další, jejichž úkolem je monitorovat vztah k těmto dvěma základním. Je to průměrný počet zobrazovaných trojúhelníků, doba funkce ohodnocení BTT (Evaluate), doba funkce plnění pole indexů Open inventoru (Fill_arrayOfIndex) a doba potřebná k přidání nových čtverců mapy a zahození starých.

Velikost obsazené paměti je přímo úměrná velikosti mapy. Neboť podle konstanty velikosti mapy jsou alokovány datové struktury jako je BTT, pole rozdílů a další. Zdálo by se, že stejná závislost by měla platit také pro počet zobrazovaných trojúhelníků. Jak je ovšem viditelné z Tabulka 7 pro test D, H, L, P (maximální velikost mapy) tato přímá úměrnost neplatí. Je to způsobeno navržením testů, kdy pro mapu s dvojnásobnou velkostí by měla být dvojnásobná velikost maximální výšky terénu (TERRAIN HEIGHT) a také dvojnásobná pozice umístnění kamery v ose „Z“. U všech testů je tato podmínka téměř dodržena, až na testy D, H, L, P, kdy je výška terénu pouze 22% své předpokládané dvojnásobné výšky. Tento nepoměr se projeví poklesem počtu zobrazovaných trojúhelníků. K poklesu dojde neboť rozhodovací poměr ovlivňuje hodnota z rozdílové mapy, která značí velikost převýšení v krajině. Při stejném parametru maximální výšky terénu a při

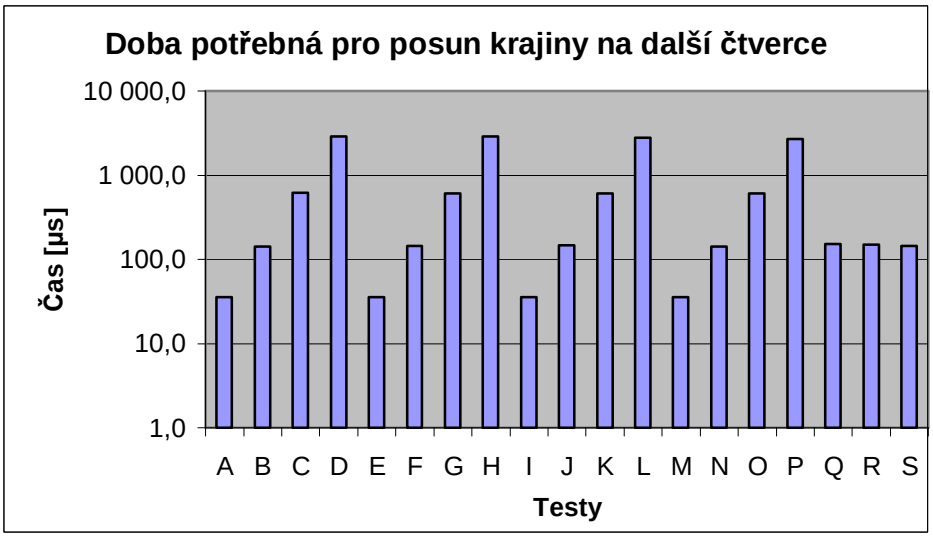
dvojnásobné velikosti krajiny, dostaneme asi poloviční převýšení krajiny (v závislosti na aktuálním tvaru krajiny), což vyplývá z principu Midpoind displacement algoritmu.

Kromě doby načítání nové mapy, která je nezávislá na ostatních funkcích, jsou všechny ostatní parametry závislé na počtu trojúhelníků. To je logický důsledek, neboť funkce ohodnocení a plnění určují kolik trojúhelníků bude zobrazených. Doba načítání nové mapy je závislá pouze na velikosti zobrazované krajiny, a proto dochází k téměř přesnému opakování časů u krajin stejné velikosti. Rozdíly jsou způsobeny pouze nedokonalým měřením, které by mělo být průměrem nekonečného počtu krajin, což není v praxi proveditelné.

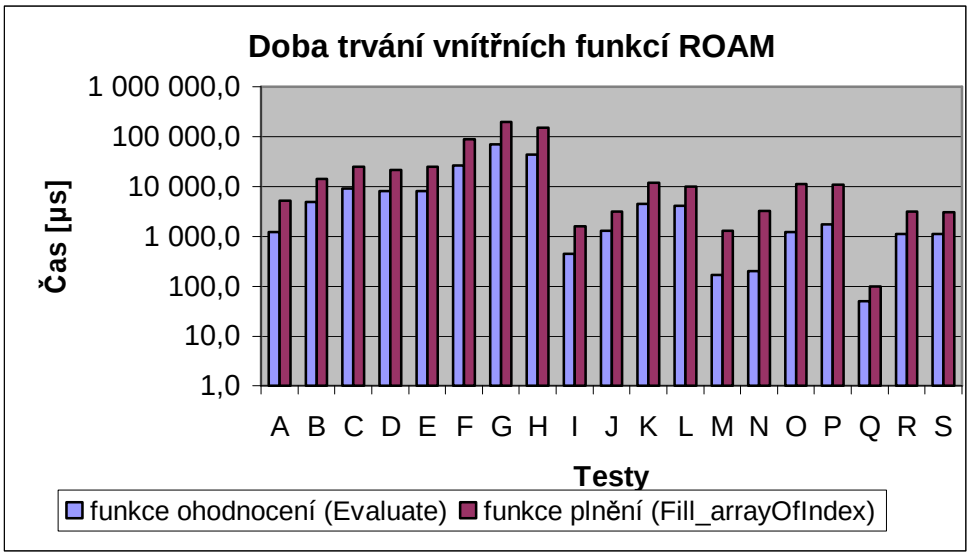
Pro lepší názornost jsou parametry vyneseny do grafů:



Graf 5.: Počet zobrazovaných trojúhelníků.

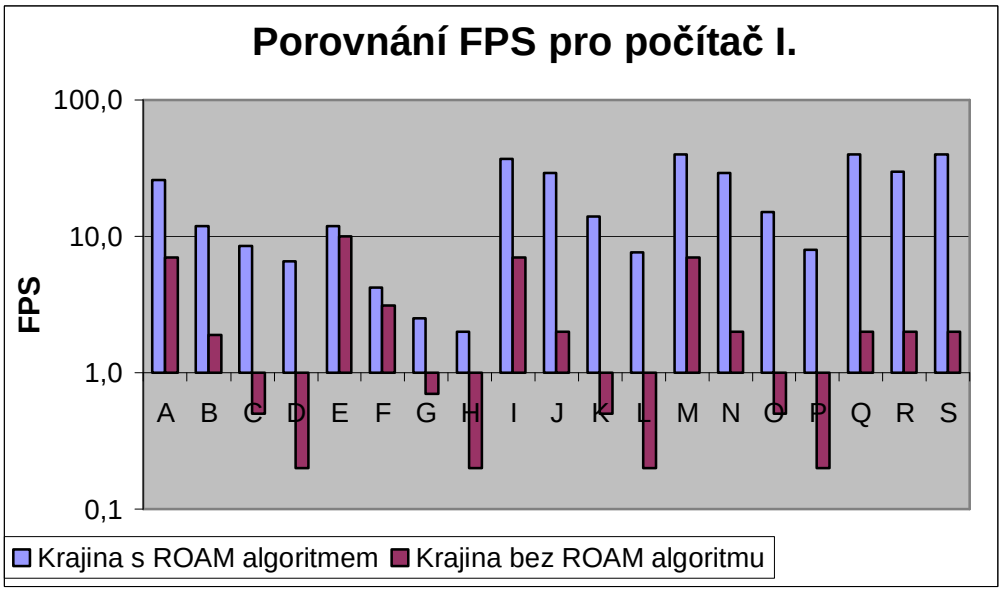


Graf 6.: Doba posunu krajiny (kopie a generování nových čtverců mapy).



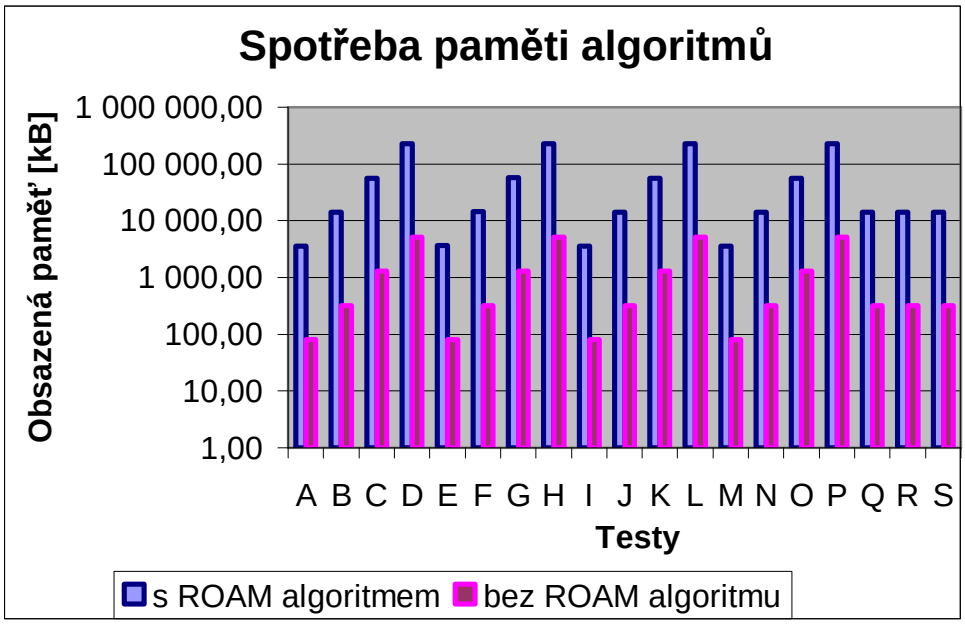
Graf 7.: Doby trvání vnitřních funkcí ROAM.

Je jasně vidět, že doba ohodnocení BTT trvá kratší dobu než doba plnění. Je to způsobeno logikou urychlující ohodnocování, kdy při ohodnocení uzlu nastavíme výslednou hodnotu také jeho bratrovi. Při plnění již tuto logiku nevyužijeme neboť je nutné získat hodnoty indexů ze všech zobrazovaných uzlů stromu. U testů M, N, O, P je viditelný pokles časů ohodnocování.



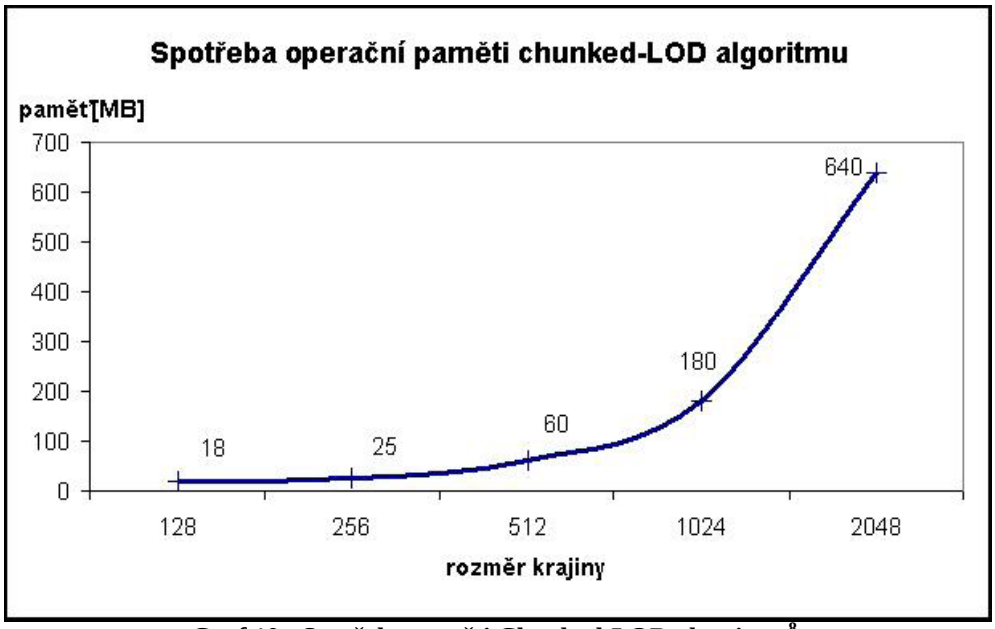
Graf 8.: Porovnání FPS pro krajinu s/bez ROAM algoritmu.

Pro úplnost uvádím Graf 9. znázorňující spotřebu paměti algoritmů pro zobrazování krajiny. Paměť ROAM algoritmu je dána průměrem měření ze všech tří počítačů. Pro krajinu bez ROAM algoritmu je obsazenost paměti u všech počítačů stejná. Stupnice paměti je logaritmická. Vychází tedy exponenciální nárůst obsazené paměti.



Graf 9.: Spotřeba paměti algoritmů.

Pro porovnání s ROAM algoritmem uvádím obsazení paměti Chunked-LOD algoritmem.



Graf 10.: Spotřeba paměti Chunked-LOD algoritmu[8].

[8] Zdroj: VRBA, Petr. Chunked-LOD, Perlinova funkce [Ročníkový projekt]. VUT v Brně, Fakulta informačních technologií.

6 Závěr

Vypracováním projektu bylo dosaženo získání teoretických i praktických znalostí algoritmů pro rendrování krajiny, zvláště pak algoritmu ROAM. Použitím ROAM algoritmu na krajinu získanou metodou Midpoint displacement, byl 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 podle výšky mapy, náhodně, či jiného důmyslného klíče.
2. Možnost provádět ohodnocování BTT některou sofistikovanou metodou. Například pouze v blízké oblasti hráče.
3. Rozdělení přechodu na nové čtverce mapy do více kroků (viz kapitola 3.2 - Obr. 9.).
4. Pokusit se navrhnout ROAM algoritmus na nečtvercovou krajinu, případně na libovolný objekt.
5. Řešit problematiku umisťování objektů do krajiny s ROAM algoritmem.
6. Spolupráce ROAM algoritmu s jinými algoritmy pro urychlení zobrazování krajiny.
7. Možnosti paralelizace výpočtů na více procesorů.

ROAM výrazně urychluje rendrování krajiny ve chvílích, kdy jsou jeho parametry nastaveny na zobrazování nízké úrovně detailu. Pokud je detail krajiny (počet zobrazovaných trojúhelníků) blízký krajině zobrazované bez ROAM (plný detail), může dojít až k zpomalení vykreslování, zvláště na počítačích s výkonnou grafickou kartou. Pokud chceme využít maximálně možnosti ROAM algoritmu, musíme velmi citlivě nastavovat parametry ROAM. Potřebujeme nalézt optimální poměr mezi počtem zobrazovaných trojúhelníků a požadovaným urychlením zobrazování.

Obecně lze říci, že ROAM algoritmus se spíše hodí na počítače s výkonným procesorem, dostatkem paměti a slabou grafickou kartou. Naopak není vhodný pro počítače s výkonnou grafickou kartou. Jako nejvíce omezující prvek celého algoritmu považuji jeho paměťovou náročnost, kdy je nutné si pro každý bod mapy pamatovat velmi mnoho informací. Se zvětšující se mapou velmi roste velikost obsazené paměti (viz Graf 9). Program je funkční pouze pod operačním systémem Windows, kde byl také vyvíjen a testován. Pro jiné operační systémy by bylo třeba provést drobné úpravy v implementaci.

7 Literatura

- PEČIVA, Jan. Tutoriál o knihovně Open Inventor. Dokument dostupný na URL http://www.root.cz/clanky/open-inventor/ (14. 3. 2006).
- SÁNCHEZ., CRESPO. Core Techniques and Algorithms in Game Programming. 2003, ISBN: 0131020099.
- TRENT, Polack. Focus On 3D Terrain Programming, Muska & Lipman/Premier-Trade. 2002, ISBN: 1592000282.
- WERNECKE, J. The Inventor Mentor. 1994, ISBN: 0201624958.
- DUCHAINEAU, M., WOLINSKY, M., SIGETI, D., E., MILLER, M., C., ALDRICH, C., MINEEV-WEINSTEIN, M., B. Los Alamos National Laboratory, Lawrence Livermore National Laboratory. ROAMing Terezin:Real-time Optimally Adapting Nesnes. Dokument dostupný na http://www.llnl.gov/graphics/ROAM/roam.pdf.
- HAVLÍČEK, Martin. Knihovna pro práci s výškovými mapami [diplomová práce]. VUT v Brně, Fakulta informačních technologií.
- VRBA, Petr. Chunked-LOD, Perlinova funkce [ročníkový projekt]. VUT v Brně, Fakulta informačních technologií.
- VÝCHOPEŇ, Daniel. Tank Hunters [ročníkový projekt]. VUT v Brně, Fakulta informačních technologií.

8 Seznamy

8.1 Seznam obrázků

Obr. 1.: Intuitivní přiblížení ROAM algoritmu ..... - 9 -
Obr. 2.: Výpočet hodnot v Poli rozdílů ..... - 12 -
Obr. 3.: Part of a diamond ..... - 12 -
Obr. 4.: Force-split ..... - 13 -
Obr. 5.: Ukázka chyby zobrazení ..... - 13 -
Obr. 6.: Ukázka dělení trojúhelníku a vytváření stromu ..... - 15 -
Obr. 7.: Ukázka korekce ..... - 16 -
Obr. 8.: Čtyři čtverce – čtyři BTT ..... - 20 -
Obr. 9.: Oblasti kolem hráče ..... - 20 -
Obr. 10.: Načtení dvou čtverců ..... - 21 -
Obr. 11.: Načtení tří čtverců ..... - 21 -
Obr. 12.: Metoda Snižování levelu krajiny ..... - 23 -
Obr. 13.: Vhodnější volba mapy ..... - 23 -
Obr. 14.: Korekční funkce ..... - 25 -
Obr. 15.: Spojení čtverců mapy vyhlazovacími pásky ..... - 25 -
Obr. 16.: Posun krajiny v Páskové metodě ..... - 26 -
Obr. 17.: Funkce pro tvorbu čtverců mapy ..... - 27 -
Obr. 18.: Kombinovaná metoda Chunked-LOD a ROAM ..... - 28 -
Obr. 19.: Zobrazení samotné výškové mapy ..... - 30 -
Obr. 20.: Strom ohodnocen od kořene ..... - 31 -
Obr. 21.: Ohodnocování pouze části BTT ..... - 32 -
Obr. 22.: Různé doby funkce ohodnocování BTT ..... - 33 -
Obr. 23.: Indexace bodů výškové mapy ..... - 35 -
Obr. 24.: Ukázka posunu krajiny ..... - 36 -
Obr. 25.: Ukázka měřičů parametrů ..... - 37 -

8.2 Seznam tabulek

Tabulka 1.: Parametry počítačů pro měření. - 42 -
Tabulka 3.: Výsledky testů – krajina s ROAM algoritmem. - 44 -
Tabulka 4.: Výsledky testů – krajina bez ROAM algoritmu. - 45 -
Tabulka 5.: Rozdíl FPS = FPS s ROAM – FPS bez ROAM. - 47 -
Tabulka 6.: Výkon ROAM = FPS s ROAM * 100 / FPS bez ROAM. - 48 -
Tabulka 7.: Výsledky testů – krajina s ROAM algoritmem – detailní proměření. - 50 -

8.3 Seznam grafů

Graf 1.: Velikost FPS pro testy A, B, C, D. Měřeno na počítači 1. - 46 -
Graf 2.: Velikost obsazené paměti pro testy A, B, C, D. Měřeno na počítači 1. - 46 -
Graf 3.: Rozdíl FPS vynesený do grafu. - 47 -
Graf 4.: Výkon ROAM vynesený do grafu. - 49 -
Graf 5.: Počet zobrazovaných trojúhelníků. - 51 -
Graf 6.: Doba posunu krajiny (kopie a generování nových čtverců mapy). - 51 -
Graf 7.: Doby trvání vnitřních funkcí ROAM. - 52 -
Graf 8.: Porovnání FPS pro krajinu s/bez ROAM algoritmu. - 52 -
Graf 9.: Spotřeba paměti algoritmů. - 53 -
Graf 10.: Spotřeba paměti Chunked-LOD algoritmu. - 53 -

Přílohy

1 Licence:

Main.cpp, ROAM.cpp, Midpoind.cpp, main.h, ROAM.h, Midpoind.h

Author: Pavel Černý (xcerny19 _AT stud.fit.vutbr.cz)

Contributors:

THIS SOFTWARE IS NOT COPYRIGHTED

This source code is offered for use in the public domain. You may use, modify or distribute it freely.

This source code is distributed in the hope that it will be useful but WITHOUT ANY WARRANTY. ALL WARRANTIES, EXPRESS OR IMPLIED ARE HEREBY DISCLAIMED. This includes but is not limited to warranties of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you find the source code useful, authors will kindly welcome if you give them credit and keep their names with their source code, but do not feel to be forced to do so.

2 Nastavování parametrů krajiny a ROAM algoritmu

soubor: ROAM-parametry.h:

TERRAIN_DIMENSION

Udává rozměr krajiny. Krajina 32x32 čtverečků 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, …). Rozměrem krajiny je myšlena velikost jednoho ze čtyř čtverců, výsledná velikost krajiny je tedy dvojnásobná

LAST_ID

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,…).

CONST_DISPLAY

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.

CONST_INSENSITIVITY

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.

TERRAIN_HEIGHT_Q

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

DISTANCE_POINT

Koeficient určuje, jak jsou od sebe vzdáleny (kolik pixelů) body v krajině.

NUM_DIVISION_TREE

Značí počet kroků na kolik je rozděleno ohodnocení celého BTT. Povolené hodnoty jsou 1 a 4.

soubor: ROAM.h:

low_border_map, hight_border_map

Konstanty určující vzdálenost kamery od kraje mapy, kdy je již třeba provést operace posunu krajiny na další čtverce.

movement_camera

Udává počet pixelů pro posun kamery při jednom stisku klávesy pohybu.

white, brown

Značí výšku krajiny, při které dochází ke změně barvy terénu. Vytváří tak efekt hornaté krajiny.

3 Ovládání programu

Popis ovládání programu je zobrazován v konzoly spouštěné zároveň s oknem aplikace.

- šipky nahoru, dolu – pohyb vpřed a vzad
- šipky doleva, doprava – otáčení vlevo a vpravo
- klávesy q, a – otáčení nahoru a dolů
- klávesy w, s – stoupání, klesání
- klávesa Mezerník – zapínání / vypínání konstantní výšky letu
- klávesa Ester – zapínání / vypínání ROAM algoritmu
- klávesa TAB – nastavení kamery do výchozí pozice
- klávesa Esc – vypnutí programu

4 Výpočet paměťových nároků

Je prováděno pro měření paměťových nároků ROAM algoritmu. Tvoří ji výpočet velikosti základních statických datových struktur (pole rozdílu, pole souřadnic bodu, BTT, atd.) a také dat dynamických (pole indexu bodů). Teoretické výpočty nemusí přesně odpovídat praxi, protože některá data mohou být uložena v registrech nebo jinak modifikována kompilátorem. Jsou zobrazovány velikosti dat pro krajinu s/bez ROAM.

Detailní výpočet:

statická_velikost_s ROAM = (sizeof(diamondInfo) + sizeof(differenceMap) + sizeof(coordPoint) + sizeof(vectorPoint) + sizeof(tree) + sizeof(hmap)) / 1024

statická_velikost_bez ROAM = (sizeof(hmap)) / 1024

dynamická_velikost = 3*sizeArrayOfIndex / 1024

Poznámka:

SizeArrayOfIndex je velikost pole obsahujícího indexy vertexů jednotlivých trojúhelníků. Je násoben třikrát, protože po naplnění indexů (funkce Fill_arrayOfIndex) je kopírován do pole (podruhé), které je předloženo OpenInventoru (strip -> coordIndex.setValues (0, (sizeArrayOfIndex*4), PoleIndices);). OpenInventor si poté provede vlastní kopii dat (potřetí).

celková_velikost_s ROAM = statická_velikost_s ROAM + dynamická_velikost

celková_velikost_bez ROAM = statická_velikost_bez ROAM + dynamická_velikost