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

Bakalářská práce

Level of Detail modely terénů

Plzeň, 2006 Hana Markuzziová

prostor pro originál práce

Abstract

Level of Detail Terrain Models

Terrain rendering is in the last years a very discussed topic. Computer generated terrains are used especially in flight simulators, games and also for virtual reality. In the first part of my work I describe how fractal geometry helps to generate terrain that is very similar to a real terrain.

Rendering terrain means to render large amounts of data that often do not fit in memory. So a number of algorithms have been proposed that render simplified representations of terrain. They are known as Level of Detail algorithms. I desribe some of them in my work.

I implemented a modified version of ROAM. I use a triangle quadtree representation and I chose a metric called variance.

Obsah

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

V Plzni dne …….

podpis

1. Úvod

S počítačovou grafikou se v dnešní době setkáváme na každém kroku. Vývoj moderních metod pro počítačovou grafiku nezaostává a dennodenně vznikají nové a lepší. Úkolem této práce je shrnout některé základní poznatky v oblasti generování fraktálních krajin a využití Level of Detail algoritmů pro jejich optimalizaci. Dále pak prostudování používaných algoritmů a implementace některého z nich.

Práce je dělena na dvě základní části, na **teoretickou** a **praktickou** část.

Teoretická část obsahuje především popisy fraktální geometrie a algoritmů pro generování fraktálních krajin. Dotýká se i metod Level of Detail – metod pro optimalizaci grafiky vzdálených bodů. A v neposlední řadě uvádím stručný výčet známějších používaných metod.

V **praktické** části se věnuji přímo vlastní implementaci algoritmu ROAM, zhodnotím tuto metodu, pokusím se nastínit její výhody oproti jiným metodám. Popisuji zde použité datové struktury a reprezentaci trojúhelníkové sítě, základní myšlenku metody nastiňuji pomocí pseudokódu. Dále popisuji řešení problémů, které mohou nastat při vykreslování.

2. Procedurální modelování

Jednou z podtříd procedurálního modelování je automatické generování přírodních objektů. Tyto algoritmy můžeme dále rozdělit na algoritmy vycházející z gramatik, tzv. L-systémy, které slouží k modelování rostlin. Druhou skupinou je fraktální geometrie využívaná ke generování krajiny, hor, kamenů apod. Poslední třídou jsou systémy částic, které nacházejí uplatnění v generování explozí, hejn ptáků nebo simulaci ohně.

2.1 Fraktální geometrie

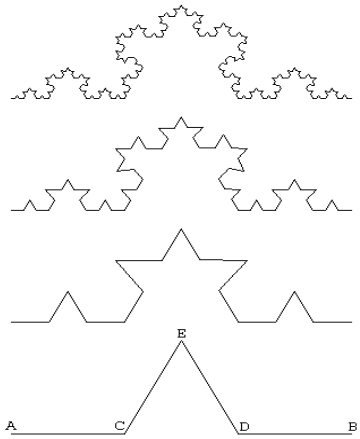
Hlavním důvodem, proč je fraktální geometrie využívána k modelování přírody, je, že objekty v přírodě se nevyznačují popsatelným geometrickým tvarem a bylo by je tedy těžké definovat jako množiny geometrických těles nebo pomocí parametrických ploch.

Typickou vlastností fraktálů je soběpodobnost. To znamená, že pokud zvětšíme libovolnou část fraktálu, bude podobný tvaru původnímu.

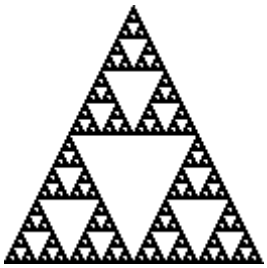
Běžná tělesa kolem nás můžeme popsat pomocí konečného množství parametrů. Platí, že výsledky jsou nezávislé na použitých jednotkách, což znamená, že budeme-li např. počítat délku v různých jednotkách, bude fyzicky stále stejná. Každému útvaru tak můžeme přiřadit hodnotu zvanou topologická dimenze. Bod má dimenzi nula, přímka dimenzi jedna, čtverec dva, krychle má dimenzi tři atd. Pro běžné útvary tak vystačíme s celočíselnými hodnotami dimenze. Bylo však zjištěno, že pokud budeme například měřit pobřeží nějakého ostrova tyčí délky jeden metr a poté tyčí délky jeden centimetr, dostaneme pokaždé jinou hodnotu. Důvodem je aproximace hodnot – některé menší zátoky jsme delší tyčí prostě eliminovali a „zahladili“. S tyčí délky jeden centimetr bude naše měření přesnější. Dostáváme se tak k poznatku, že pro měřítko blížící se k nule dostaneme délku blíží se k nekonečnu. Tuto myšlenku můžeme vyjádřit vztahem:

$$K = N\,\varepsilon^{D} \qquad (2.1)$$

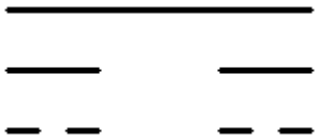
kde K je výsledná délka, N je počet úseček, pomocí kterých jsme objekt aproximovali, ε je délka úsečky a D je tzv. fraktální dimenze. Pro málo členité útvary je fraktální dimenze rovna dimenzi topologické. Vysoce členité objekty se nazývají fraktály. Příklady fraktálů jsou na obrázcích 2.1 až 2.5.



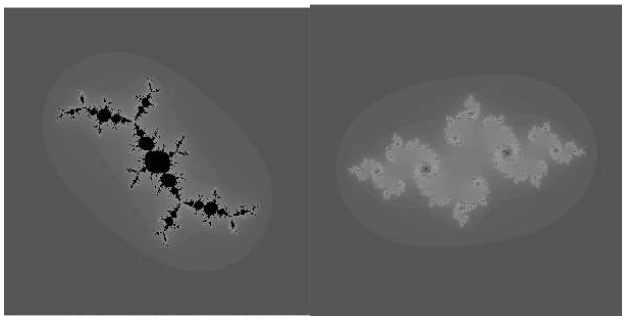
Obr. 2.1 - Kochova vločka [3]



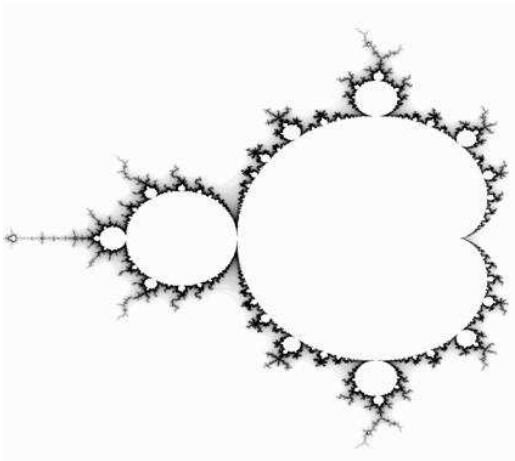
Obr. 2.2 - Sierpinského trojúhelník [3]



Obr. 2.3 - Cantorova množina [3]



Obr. 2.4 - Juliova množina: [2]



Obr. 2.5 - Mandelbrotova množina [1]

2.2 Využití fraktální geometrie v počítačové grafice pro generování krajiny

Virtuální krajiny nacházejí uplatnění v mnoha oblastech, ale především se jedná o letecké simulátory a hry. Nejjednodušším způsobem je generování krajiny pomocí výškové mapy (heightfield, heightmap). Máme k dispozici dvourozměrnou matici, jejíž prvky udávají výšku terénu nad základní úrovní (např. bitmapový soubor, kde odstíny jednotlivých pixelů udávají výšku bodu). Nevýhodou této struktury je, že neumožňuje reprezentaci objektů, jako jsou jeskyně nebo převisy. Výškové mapy se zobrazují pomocí polygonů, nejčastěji sítí trojúhelníků.

Pokud chceme vygenerovat náhodnou krajinu, můžeme použít některý z následujících algoritmů. Mezi nejznámější patří metoda přesouvání středního bodu (midpoint displacement) a metoda náhodných poruch (fault formation).

Metoda přesouvání středního bodu se v trojrozměrném prostoru aplikuje na trojúhelníky nebo na čtverce. V prvním kroku máme čtverec a každý jeho roh má přiřazenu určitou výšku. Vezmeme prostřední bod čtverce a vypočítáme průměrnou výšku ze čtyř rohových bodů a připočteme náhodnou hodnotu, která má Gaussovské rozdělení. Rozptyl těchto hodnot je závislý na hloubce rekurze. V dalším kroku vypočítáme výšku bodů, které leží na hranách zadaného čtverce, jako aritmetický průměr hodnot okolních bodů a opět připočteme náhodnou hodnotu. Dostaneme čtyři nové čtverce a takto rekurzivně pokračujeme. Rekurzi můžeme například ukončit, pokud jsou rohy čtverce příliš blízko u sebe.

Algoritmus náhodných poruch se aplikuje na dvourozměrnou matici, která má na začátku všechny hodnoty výšky nastaveny na nulu. Matici náhodně rozdělíme na dvě libovolné části, nejjednodušší je přetnout ji přímkou. Výšku všech bodů na jedné straně zvýšíme o stejné náhodné číslo. Tímto způsobem uděláme několik desítek kroků.

3. Level of Detail

Abychom ošetřili množství dat, která vlastně ani nepotřebujeme, použijeme některou metodu typu Level of Detail(dále LOD). Princip těchto metod je jednoduchý. Není totiž důležité zobrazovat v dálce jakkoli malý trojúhelník – je to totiž zbytečné, protože i např. rozlišení monitoru není tak velké, aby dokázalo tak malé objekty zobrazit. V paměti by ale přesto zabíraly zbytečně moc místa a navíc by se vykreslovaly. Proto je dobré zobrazit v lepším rozlišení jen trojúhelníky, co se nacházejí v aktuálním okolí pozorovatele a objekty vzdálenější optimalizovat právě tím, že se zobrazí v horším rozlišení. Přínos této metody je dvojí. Jednak je to zrychlení celého procesu vykreslování, vykreslujeme totiž méně polygonů, druhým je i graficky lepší vzhled scény. Lidské oko je totiž zvyklé zaměřovat pozornost na předměty v blízkosti, takže předměty v dálce spíše eliminuje a přílišné zobrazení nedůležitých detailů v dálce je spíše překážkou než přínosem.

3.1 Druhy LOD

Metody LOD můžeme rozdělit na dva druhy – diskrétní a spojité.

Hlavní myšlenkou diskrétních metod je, že máme určitý počet statických trojúhelníkových sítí v různých rozlišeních. Během vykreslování pouze vybereme úroveň, která je vzhledem k vzdálenosti a poloze pozorovatele nejvhodnější a tu vykreslíme. Výhodou tohoto přístupu je, že máme vše uloženo v paměti už před započetím vykreslování a během samotného vykreslování již nemusíme nic znovu generovat. Pokud se však pozorovatel pohybuje a chceme-li pro každou část modelu zobrazit určitý stupeň detailu, začíná být ukládání do paměti nepraktické. Diskrétní LOD se používají pro jednodušší aplikace, protože nejsou tak výpočetně náročné.

Spojité LOD metody generují rozdíly v trojúhelníkové síti za běhu programu. Pro dosažení stejné kvality modelu jako při diskrétních metodách tak spotřebujeme méně trojúhelníků. Rozšířením spojitých LOD jsou pohledově závislé LOD, které dynamicky mění množství polygonů pro danou pozici pozorovatele, takže místa blíže k pozorovateli jsou zobrazována ve větším rozlišení než místa vzdálenější. Při zobrazování terénu je nejvhodnější použít spojité pohledově závislé LOD.

3.2 Chybové metriky

K rozhodnutí, kdy a do jaké části sítě máme přidat detaily, nebo je naopak odebrat, nám slouží chybové metriky. Jsou konstruovány tak, aby změna v síti nebyla tak nápadná a zároveň bylo dosaženo požadovaného detailu. První, co by nás napadlo, je použít eukleidovskou vzdálenost. To je ovšem vzhledem k tomu, že chceme zjišťovat vzdálenost od plochy, nemožné. Jednou z používaných metrik je Hausdorffova vzdálenost. Hausdorffovu vzdálenost definujeme na množině bodů A a B, kde pro každý bod z A najdeme v množině B nejbližší bod a totéž určíme pro celou množinu B. Pro každou dvojici bodů spočteme jejich vzdálenost. Maximum z těchto hodnot je hledaná Hausdorffova vzdálenost. Alternativou k Hausdorffově vzdálenosti je tzv. vzdálenost zobrazení (mapping distance). Vzdálenost zobrazení je spojité zobrazení, přesněji bijekce, mezi dvěma povrchy.

Máme-li dáno zobrazení F

$$F : A \rightarrow B \qquad (3.1)$$

definujeme vzdálenost zobrazení jako

$$D(F) = \max_{a \in A} \| a - F(a) \| \qquad (3.2)$$

D je tedy vzdálenost mezi body *A* a *B*.

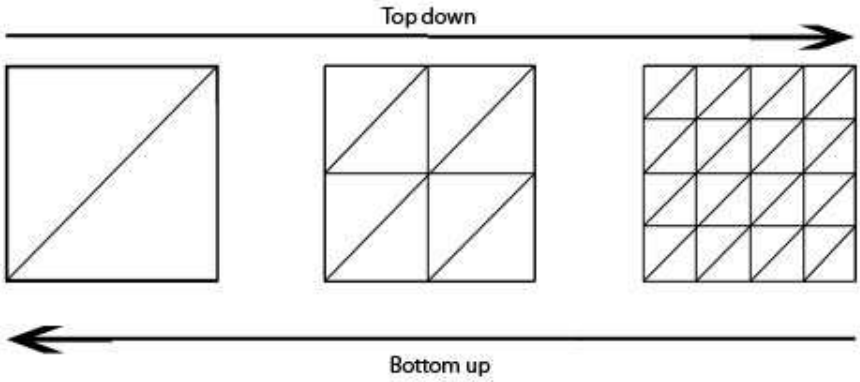
Hausdorffova vzdálenost i vzdálenost zobrazení vybírají maximum z hodnot, ale někdy je vhodnější použít průměrnou vzdálenost nebo odmocninu z průměru.

Další používanou metrikou je chyba obrazového prostoru (screen space error). Nejprve je nutné vysvětlit pojem chyba prostoru objektů (world space error). Pod pojmem chyba prostoru objektů si můžeme představit míru rozdílu mezi skutečným objektem a objektem aproximovaným pomocí trojúhelníkové sítě. Chyba obrazového prostoru je potom promítnutí chyby prostoru objektů.

4. Level of Detail modely terénů

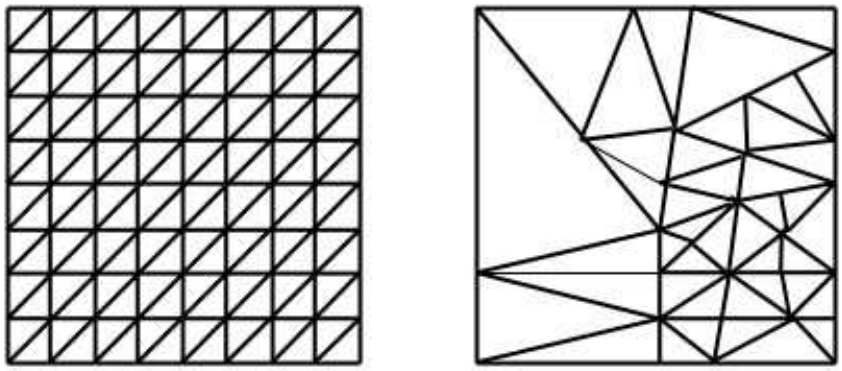
Zobrazování terénů je na rozdíl od zobrazování jiných 3D modelů jednodušším úkolem. Terén totiž často reprezentujeme jako pravidelnou trojúhelníkovou síť a tudíž je uložení dat a následná manipulace s nimi jednoduchá. Problém, který u terénů řešíme, je velké množství dat, která se nám nemusejí vejít do paměti. Proto se při zobrazování terénů využívá pohledově závislých LOD, aby se krajina v okolí pozorovatele zobrazovala ve větším rozlišení než krajina vzdálená.

V LOD modelech terénů se používají dva přístupy ke zjednodušování – shora dolů (top down) a zdola nahoru (bottom up). Příklad je na obrázku 4.1. V algoritmu shora dolů začínáme pouze s několika trojúhelníky (obvykle dva nebo čtyři) a pak přidáváme nové, dokud není dosaženo určitého rozlišení. Tyto metody jsou často označovány jako tzv. subdivision nebo metody zjemňování. Naproti tomu zdola nahoru algoritmus začíná s nejvyšším rozlišením a jsou odstraňovány vrcholy, dokud není dosaženo požadovaného zjednodušení. Pomocí algoritmu zdola nahoru jsme schopni nalézt minimální množství trojúhelníků, které jsou potřebné pro danou přesnost, na druhé straně ale má algoritmus velké nároky na paměť a dobu výpočtu.



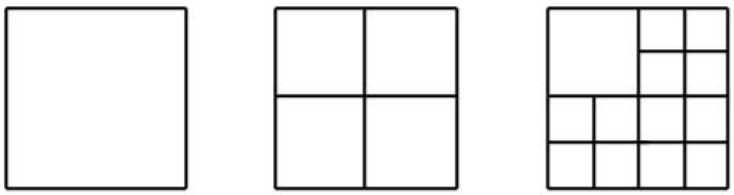
Obr. 4.1 - Znázornění přístupů top down a bottom up

Dalším rozdílem mezi LOD algoritmy je struktura používaná k reprezentaci terénu. Používají se buď pravidelné sítě nebo nepravidelné trojúhelníkové sítě (triangulated irregular network - TIN). Příklad trojúhelníkových sítí je na obrázku 4.2. Pravidelné sítě využívají pole s hodnotami výšek a pravidelné rozmístění, kdežto TIN umožňují mezi vrcholy proměnné vzdálenosti. TIN mohou aproximovat povrch požadované přesnosti použitím méně polygonů než sítě pravidelné. Výhodou pravidelných sítí je, že mohou být jednoduše uloženy a snadno se s nimi manipuluje. K uložení nám stačí dvourozměrné pole, ve kterém si uchováváme pouze hodnoty výšek místo všech tří souřadnic. Chceme-li například zjistit změnu výšky v nějakém bodě, provedeme to jednoduše interpolací čtyř sousedních bodů. Z těchto důvodů většina současných LOD metod používá pravidelné sítě.



Obr. 4.2 – Pravidelná trojúhelníková síť a TIN

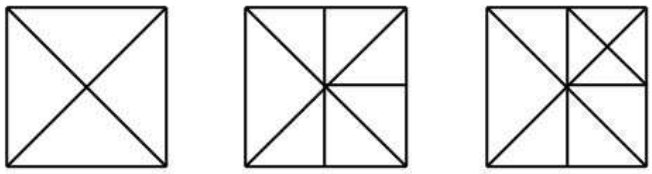
K tomu, abychom mohli implementovat pravidelné sítě, kde LOD je závislé na poloze pozorovatele, musíme reprezentovat různé části sítě s různým rozlišením. K tomu slouží hierarchické datové struktury, které nám umožňují přidávat nové detaily. Nejpoužívanějšími strukturami jsou quadtree a binary triangle tree. Příklad struktury quadtree je na obrázku 4.3 a struktury binary triangle tree je na obrázku 4.4. Quadtree je struktura, ve které čtvercovou plochu dělíme na čtyři stejné části. Každou z těchto částí můžeme opět rekurzivně dělit.



Obr. 4.3 – Quadtree

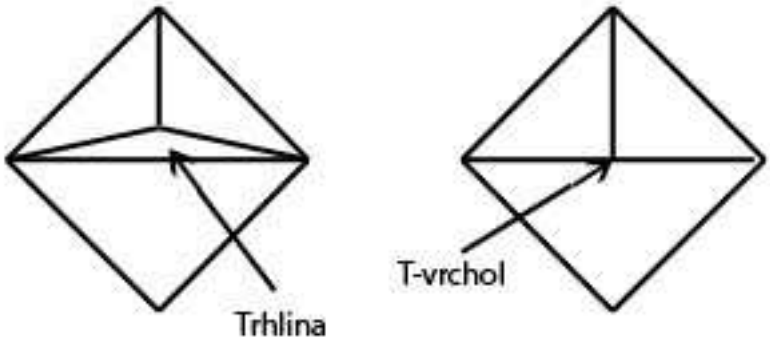
Struktura binary triangle tree (dále bintree) pracuje na stejném principu jako quadtree, ale místo dělení čtverce na čtyři části dělíme trojúhelník rekurzivně na dvě části. Jako základní trojúhelník se často používá pravoúhlý trojúhelník a dělení probíhá tím způsobem, že se spojuje vrchol se středem základny. Jednou z výhod struktury bintree je, že během generování

trojúhelníkové sítě nevznikají trhliny. Jejich vlastností je také to, že sousedi se mohou lišit maximálně o jeden stupeň rozlišení.



Obr. 4.4 – Binary triangle tree

Běžnými problémy, se kterými se můžeme v LOD setkat, jsou trhliny a T-vrcholy (T-junction). Příklad trhliny a T-vrcholu je na obrázku 4.5. Trhliny vznikají v místech, kde vyšší LOD vyžadovalo přidání nového vrcholu, který ovšem neleží na hraně s nižší LOD. Během vykreslování se trhliny projeví jako díry v terénu. T-vrcholy vzniknou, pokud vrchol trojúhelníku, který má vyšší LOD, nesdílí tento bod s trojúhelníkem s nižším LOD.



Obr. 4.5 – Trhlina a T-vrcholy

Existuje několik způsobů, jak trhliny ošetřit

a) Trojúhelníky v okolí zlomu jsou rekurzivně děleny, dokud není dosaženo souvislého terénu. Tento způsob je často používán v algoritmech, které využívají strukturu bintree.

b) Výška nového vrcholu v trojúhelníku s vyšším LOD je upravena tak, aby byla stejná jako výška hrany trojúhelníku s nižším LOD. Tato metoda zapříčiňuje vznik T-vrcholů.

c) Podobnou operací jako předešlá je přidání nového vrcholu na hranu trojúhelníku s nižším LOD do stejného místa, jako jsme přidali vrchol k trojúhelníku s vyšším LOD.

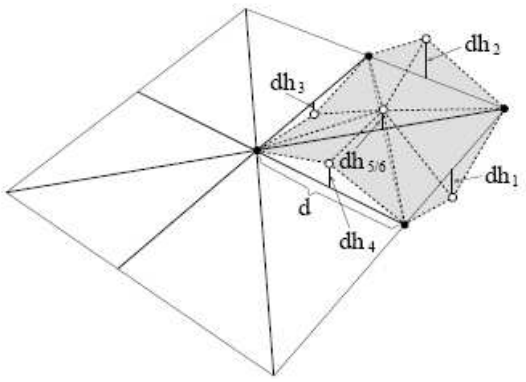
d) Do sítě je vložen nový trojúhelník, tak, aby zakryl trhlinu. Vzniká tak sice souvislá trojúhelníková síť, ale tyto nově přidané trojúhelníky jsou kolmo v povrchu a mohou tak vypadat nepřirozeně.

5. Přehled nejznámějších algoritmů

5.1 Generování spojitých Level of Detail pro výšková pole v reálném čase

Algoritmus generování spojitých LOD pro výšková pole v reálném čase (Real-time generation of continuous Levels of Detail for height fields) je převzat z [7]. Jako datovou strukturu využívá quadtree, který je reprezentován booleovskou maticí nazývanou quadtree matrix. Každému uzlu v quadtree přísluší jedna hodnota v matici, která udává, jestli má být uzel rozdělen nebo ne. Hodnota nula znamená, že se jedná o list stromu, hodnota větší než nula signalizuje, že jde o uzel s potomky. Otazníkem jsou označeny dosud nepoužité pozice.

K tomu, abychom mohli vypočítat hodnoty v quadtree matrix, musíme rekurzivně procházet quadtree a v každém uzlu se rozhodnout, jestli má správnou úroveň detailu nebo je uzel potřeba rozdělit na čtyři potomky. Algoritmus zohledňuje nerovnosti terénu (nazývanou v článku error - chyba), což znamená, že plochá místa používají méně polygonů, než místa členitější. Nerovnost terénu se nemění, můžeme ji tedy vypočítat předem. Chybu v určitém místě spočteme jako rozdíl ve vyvýšení hran a diagonál. Dostáváme tak šest hodnot, ze kterých vybereme maximum a to prohlásíme za hledanou chybu v daném místě. Označme si ji d2.

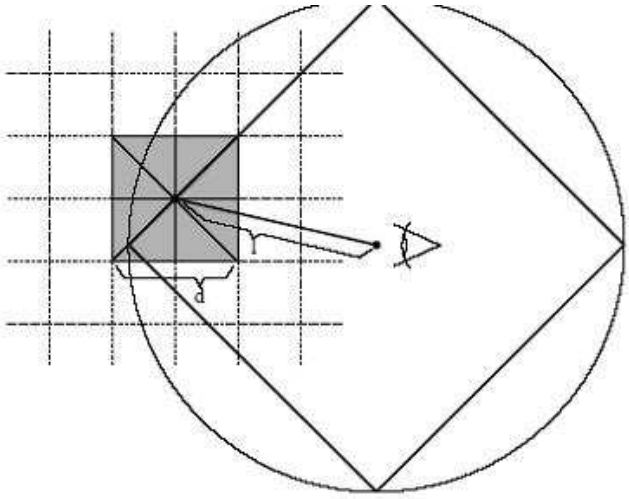


Obr. 5.1 - Zjišťování nerovnosti povrchu [7]

Během generování matice se využívají dvě konstanty – minimální globální rozlišení (*C*) a požadované globální rozlišení (*c*). Při stanovování konstant bereme v úvahu, že rozlišení by mělo klesat v závislosti na tom, jak se zvětšuje vzdálenost od kamery. Tuto podmínku můžeme zapsat jako

$$\frac{l}{d} < C \qquad (5.1)$$

kde *l* je vzdálenost od kamery a *d* je rozměr bloku (obr. 5.2).



Obr. 5.2 - Minimální globální rozlišení [7]

Pokud se konstanta *C* zvětšuje, zvětšuje se kvadraticky počet vrcholů.

Konstanta *c* nám ovlivňuje množství polygonů, které jsou v každém snímku vykreslovány.

Abychom mohli spočítat konečnou chybu pro každý uzel, zavádí se konstanta K definovaná následujícím způsobem:

$$K = \frac{C}{2\,C - 2} \qquad (5.2)$$

V každém uzlu je konečná chyba počítána jako maximum lokální chyby uzlu a K násobku předtím spočítané chyby v nižších úrovních. Tento krok zajistí, že LOD sousedních uzlů nebude větší než jedna, v opačném případě by se v terénu objevily trhliny.

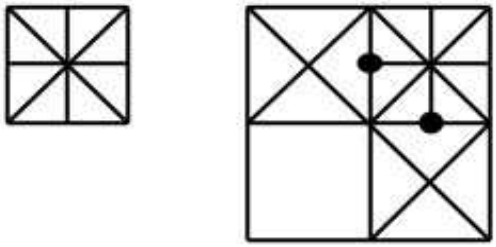
K tomu, jestli má být uzel rozdělen nebo ne využíváme metriku. V každém snímku počítáme následující rovnici

$$f = \frac{L}{d\,C\,\max(1,\, c\,d2)} \qquad (5.3)$$

kde *L* je vzdálenost uzlu od kamery, *d* je délka bloku (viz obr 5.2), *C* je minimální globální rozlišení, *c* je požadované globální rozlišení a *d2* je chyba vypočítaná výše. Uzel je rozdělen pokud je splněna podmínka, že $f < 1$.

Při vykreslování procházíme strom a pokud narazíme na list, vykreslíme vějíř trojúhelníků (triangle fan) - viz obr. 5.3.

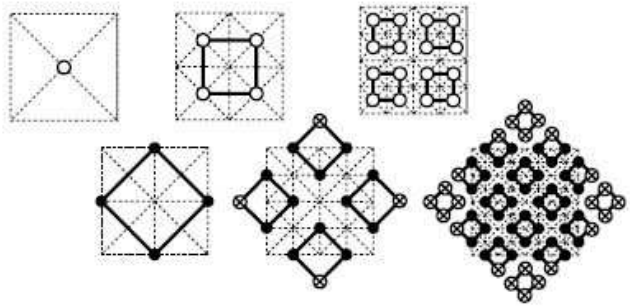
Abychom se vyhnuli trhlinám v místech, kde se dotýkají části s jinou úrovní detailu, vrchol na hraně přeskočíme. Jestli k tomuto jevu dochází, zjistíme z matice tím, že zkontrolujeme jestli jsou sousední uzly rovny nule.



Obr. 5.3 – Vějíř trojúhelníků a vykreslování uzlů v různých LOD

5.2 Zobrazování velkých terénů jednoduše

Algoritmus zobrazování velkých terénů jednoduše (Visualization of large terrains made easy) je převzat z [8]. Metoda se snaží místo ukládání uzlů do stromu vyrobit jeden velký pruh trojúhelníků (triangle strip). K tomu využívá speciální datovou strukturu nazvanou interleaved quadtree. Výhodou této struktury je, že vrcholy, které jsou geometricky blízko, jsou vedle sebe uloženy i v paměti a tím se omezí na minimum stránkování. Struktura se vytváří během preprocessingu. Vrchol je označen buď jako bílý, jestliže je přidán do pruhu trojúhelníků na liché úrovni, nebo jako černý, pokud je přidán na úrovni sudé. Dostáváme tak dva stromy označované jako black quadtree a white quadtree.



Obr. 5.4 - Nahoře: první tři úrovně white quadtree
Dole: black quadtree s přidanými vrcholy z white quadtree [7]

Ve white quadtree jsou některé pozice nevyužité, a pokud implementujeme white quadtree jako pole, vzniknou nám díry. Místo toho, abychom použili dva stromy, využijeme jen black quadtree a do prázdných míst uložíme vrcholy z white quadtree (viz obr. 5.4).

Terén je reprezentován jako orientovaný graf vektorů, díky tomu mohou být indexy potomků spočteny pomocí jednoduché rovnice.

K výpočtu indexu, kde se nacházejí potomci, pokud známe index rodiče, slouží vztah:

$$c_{p,k} = 4p + k + m \qquad (5.4)$$

kde p je index rodiče, k je číslo potomka, jehož index hledáme ($k = 0 \ldots 3$, protože máme čtyři potomky), m je konstanta, která závisí na indexu vrcholu a vzdálenosti indexů mezi jednotlivými LOD.

Ve fázi preprocessingu jsme tedy vygenerovali strom. Algoritmus o každém uzlu rozhoduje, jestli je aktivní. Pokud ano, rekurzivně rozdělí levého potomka, přidá vrchol aktuálního uzlu do pruhu trojúhelníků a rekurzivně dělí pravého potomka. K rozhodnutí, jestli je uzel aktivní nebo ne, používáme parametry chyba obrazového prostoru a chyba prostoru objektů.

Chyba prostoru objektů je míra rozdílu mezi původním terénem (výškové pole) a aproximovaným terénem (vykreslovaný terén). Pro obě chyby musí platit, že chyba uzlu je menší než chyba jeho předka. Z toho plyne, že pokud je vrchol aktivní, všichni jeho předci musí být také aktivní a vzniká tak souvislá síť trojúhelníků. Otázkou tedy je, jak tuto hierarchii zajistit. Může se totiž stát, že vrchol potomka je ke kameře blíž než jeho rodič. Kvůli perspektivnímu promítání bude chyba vyšší pro bližší vrchol. Protože promítání závisí na aktuální poloze pozorovatele, musí se chyby pro každý snímek přepočítat. Algoritmus přináší řešení v podobě ohraničujících koulí kolem každého vektoru, které si můžeme předem předpočítat. U každé koule máme uložen pouze její poloměr. Pokud nyní promítneme chybu prostoru objektů každého vektoru z toho bodu na jeho kulové ploše, který je nejblíže kameře, potom výsledná chyba bude menší než chyba rodiče tohoto uzlu.

Uzel je aktivní, jestliže chyba obrazového prostoru je větší než určitá tolerance *T*. *T* může být každý snímek měněna a tím ovlivňováno množství vykreslovaných trojúhelníků a přesnost terénu.

5.3 Shlukové LOD

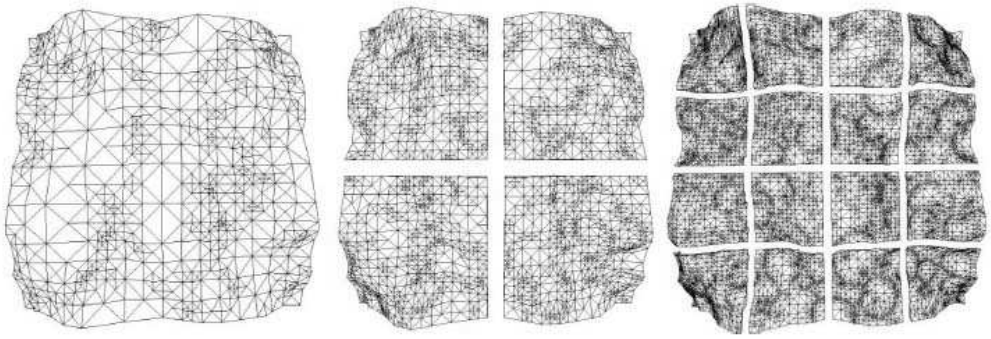
Algoritmus shlukových LOD (Chunked LOD) je převzat z [9]. Jedná se o metodu založenou na diskrétních LOD. Jejím cílem není dosáhnout optimální triangulace, ale maximalizovat množství trojúhelníků a snížit přitom zatížení CPU. Hlavní myšlenkou je, že při dnešních výkonných grafických systémech je lepší vykreslovat více trojúhelníků než ztrácet čas CPU odstraňováním některých z nich. LOD není aplikováno na jednotlivé vektory, ale na celé části terénu, tzv. chunks (viz obr. 5.5).

Během preprocessingu je vygenerován quadtree, který má v každém uzlu část terénu z určitým rozlišením. Části terénů v každé úrovni mají i jinou velikost.

Pokud je terén složen z různých částí, mohou se v něm vyskytnout zlomy. Autor je řeší přidáním dalšího trojúhelníku tak, aby se zakryla mezera.

Velkým problémem u diskrétních LOD je „poskočení“ terénu, protože se najednou mění velké množství vektorů. Jako řešení se použije morfování, přičemž u každého vrcholu je uložena hodnota, která udává vzdálenost povrchu od rodičovského uzlu v tomto bodě. Tato hodnota se využívá při přechodu z jedné úrovně terénu do druhé.

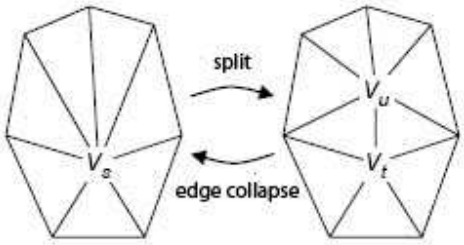
Vykreslování je velmi jednoduché, kdežto předzpracování je dosti náročné.



Obr. 5.5 - První tři úrovně chunked LOD [8]

5.4 Progresivní sítě

Metoda progressivních sítí (Progressive meshes) je převzata z [10]. Hlavní myšlenkou je použití speciální datové struktury, která uchovává nejenom geometrii, ale i další atributy, které se k vrcholu vážou např. informace o materiálu. Začínáme s trojúhelníkovou sítí v plném rozlišení a postupně ji zjednodušujeme pomocí metody zvané kolapse hran (edge collapsing) tj. že vektor degradujeme na vrchol. (viz obr. 5.6).



Obr. 5.6 – Kolapse hran

Během této operace si uchováváme potřebné informace tak, aby bylo možno provést i inverzní operaci. Pro to, jaké hrany a vrcholy při operacích použít, existují různé algoritmy.

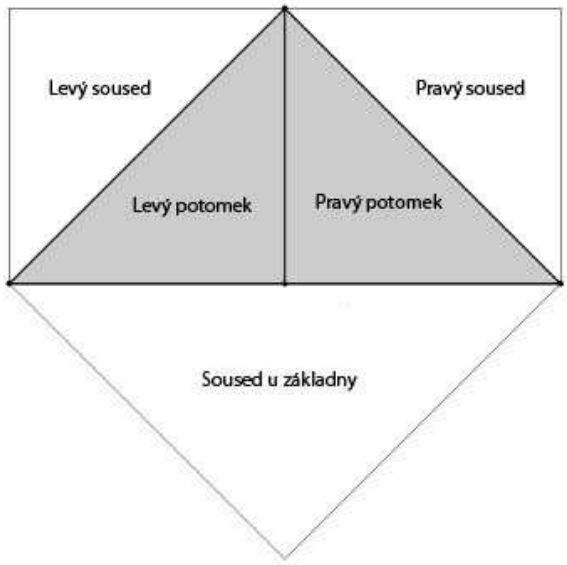
Vytvoření celé trojúhelníkové sítě je složité a musí se provádět při preprocessingu. Nevýhodou je velké množství dat, které se nevejde do paměti.

Hlavní výhoda je, že se nejedná o pravidelnou trojúhelníkovou síť, a k zobrazení některých částí terénu tak stačí mnohem méně polygonů než u metod s pravidelnými sítěmi.

Jinou vylepšenou metodou je VIPM [11] (view independet progressive meshes), která zahrnuje dobré vlastnosti původních progresivních sítí a zlepšuje komplikovaný výpočet při pohledově závislém zjemňování.

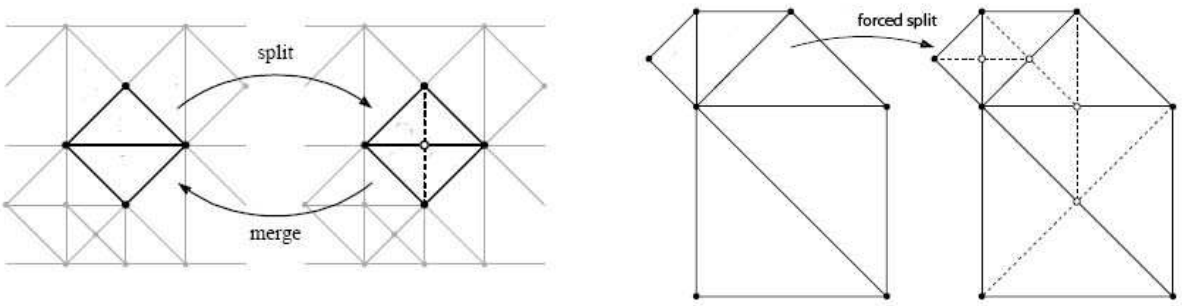
5.5 ROAM

Algoritmus převzat z [12]. Zkratka ROAM pochází z anglického Real-time Optimally Adapting Meshes (Optimálně se přizpůsobující sítě v reálném čase). Algoritmus používá jako datovou strukturu binary triangle tree, což je struktura, ve které je uzel pravoúhlý trojúhelník, který může mít dva potomky. Potomky vytvoříme tak, že spojíme vrchol trojúhelníku se středem základny. U každého uzlu si uchováváme pět ukazatelů - na dva potomky a tři sousedy (obr. 5.7).



Obr. 5.7 – Uzel stromu s ukazateli

K dispozici máme výškové pole, jehož rozměry nám udávají maximální možný počet vrcholů, který můžeme vykreslovat. Výškové pole si rozdělíme na dva pravoúhlé trojúhelníky. Oba tyto trojúhelníky rekurzivně dělíme do určité přesnosti. Nad trojúhelníkovou sítí definujeme dvě operace - Split a Merge (obr. 5.8). Operace Split nám trojúhelník rozdělí na dva menší, operace Merge naopak ze dvou trojúhelníků vytvoří jeden větší.

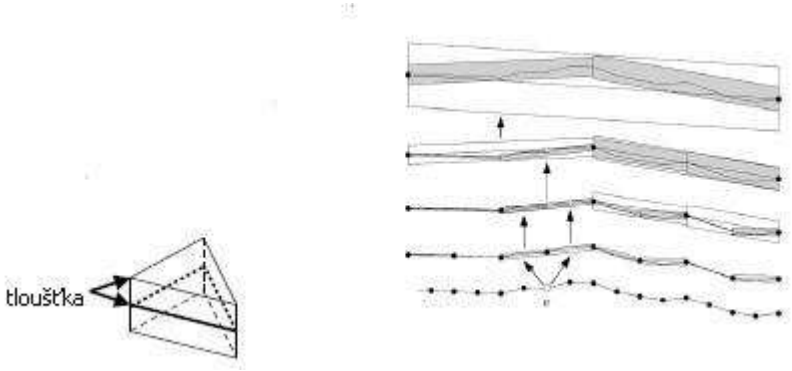


Obr. 5.8 – Operace split a merge, vynucené rozdělení (forced split)

Dva sousední trojúhelníky nazveme diamant, jestliže jsou ve stejné úrovni detailu a sdílejí společnou základnu. Pokud tvoří trojúhelník *A* s trojúhelníkem *B* diamant a chceme rozdělit trojúhelník *A*, musí být současně s ním rozdělen i trojúhelník *B*. V opačném případě by vznikly v terénu trhliny. Po tomto dělení se však může stát, že jiné dva trojúhelníky budou tvořit diamant, ale jeden z nich bude rozdělený a druhý nikoliv. Tento jev se řeší rekurzí, dokud všechny trojúhelníky v síti nemají své sousedy v diamantu rozděleny. Tento proces označujeme jako vynucené rozdělení (forced split) - viz obr.5.8. Pro velké trojúhelníkové sítě se ale může stát rekurze velmi drahou operací.

Trojúhelníky jsou ukládány do dvou front. V první frontě jsou trojúhelníky určené pro operaci Split a v druhé pro operaci Merge. Trojúhelníky jsou ve frontách seřazeny podle priorit.

K výpočtu priorit lze použít různé metriky. Jednou z nich je použití tzv. klínků (wedgie), které si definujeme pro každý trojúhelník. Klínek je ohraničující prostor okolo trojúhelníku s určitou tloušťkou. Tloušťka klínku rodiče je vždy větší, než tloušťka potomka. Klínky počítáme ve fázi preprocessingu. Jak klínek vypadá, je patrné z obrázku 5.9.



Obr. 5.9 – Klínek a jejich hierarchie

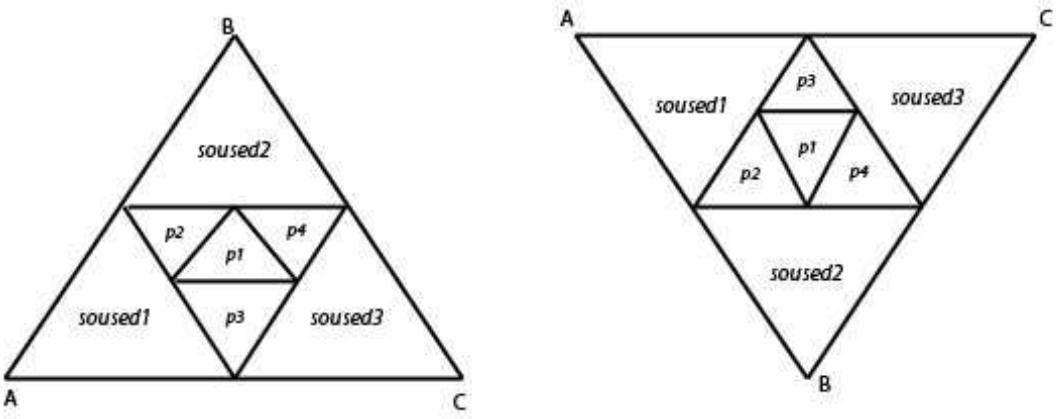
Další metrikou je geometrická obrazová deformace (geometric screen distortion). Geometrická obrazová deformace je definována jako vzdálenost mezi místem, kde by se měl nacházet bod terénu, a místem, kde se bod ocitnul, pokud jsme terén aproximovali trojúhelníkovou sítí. Jako deformace celého povrchu se bere maximum deformací všech bodů. Jaké maximální hodnoty může deformace nabývat zjistíme, pokud promítneme klínek každého trojúhelníku na obrazovku.

6. Vlastní implementace ROAM

Algoritmus ROAM jsem si vybrala z několika důvodů. Jedním z nich je, že při plynulém pohybu kamery nemusíme znovu generovat celou trojúhelníkovou síť, ale pouze několik trojúhelníků. Množství zobrazovaných trojúhelníků stejně jako rozlišení terénu lze jednoduše měnit, protože jsou trojúhelníky uloženy ve frontách. Algoritmus je tedy lehce implementovatelný a přehledný.

6.1 Reprezentace trojúhelníkové sítě

Jako datovou strukturu jsem použila triangle quadtree. Základem je trojúhelník, který je dělen na čtyři menší trojúhelníky. K uchování vztahů mezi nimi v síti využívám ukazatelů na rodiče, tři sousedy a čtyři potomky. Potomci jsou ty trojúhelníky, které mají společného rodiče. Rodič je tedy trojúhelník, s jehož pomocí můžeme nadefinovat potomky. Sousedé jsou trojúhelníky, které můžou mít společného rodiče, ale nemusí. Hlavní vlastností sousedů je, že se nacházejí vedle sebe tj. mají jednu společnou hranu. Na obrázku 6.1 jsou vztahy znázorněny pro trojúhelník směřující vrcholem dolů a pro trojúhelník směřující vrcholem nahoru. Potomci jsou označeni p1, p2, p3 a p4.



Obr. 6.1 – Potomci a sousedi

Každý trojúhelník reprezentuji v programu jako objekt, který kromě již zmíněných ukazatelů nese i informaci o souřadnicích bodů trojúhelníku. Objekt vypadá takto:

```
Triangle
{
        Triangle potomek1;
        Triangle potomek2;
        Triangle potomek3;
        Triangle potomek4;
        Triangle soused1;
        Triangle soused2;
        Triangle soused3;
        Triangle rodic;
        Vector a, b, c;

}
```

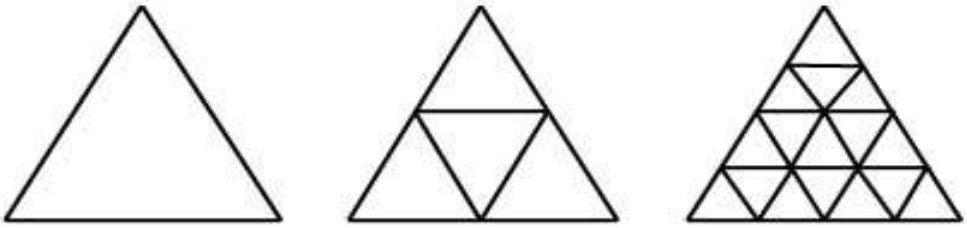
Výhodou této reprezentace trojúhelníkové sítě je, že vrchol je sdílen šesti trojúhelníky, což se projeví tím, že pro vykreslování budeme potřebovat ukládat menší množství bodů. Nevýhodou naopak je, že pokud mám vedle sebe dva trojúhelníky, každý s jiným LOD, vznikne vždy trhlina, kterou musím ošetřit.

6.2 Generování trojúhelníkové sítě

Algoritmus je typu shora dolů, začínáme tedy s jedním základním trojúhelníkem. Ten je výhodné volit buďto jako pravoúhlý nebo rovnostranný, aby bylo možno jednoduše spojit trojúhelníky do čtverce, ze kterých můžeme pak vytvořit rozsáhlý terén. Místo výškového pole využívám matematickou funkci dvou proměnných $z = f(x, y)$, kde x a y jsou souřadnice bodu a z je výška terénu v daném bodě. Pokud bychom místo funkce dvou proměnných použili generování náhodné hodnoty, museli bychom ošetřit případy, kdy je tentýž bod součástí více trojúhelníků a je tak generován vícekrát s jinou náhodnou hodnotou. Tento problém by se vyřešil například použitím matice, ve které bychom si uchovávali, zda byla

hodnota již vygenerována. Proto je jednodušší použít buď výškovou mapu nebo funkci dvou proměnných.

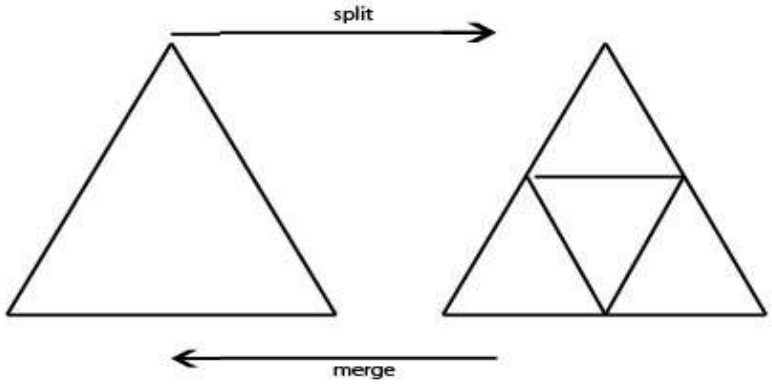
Trojúhelníkovou síť vytvoříme následujícím způsobem. Vezmeme základní trojúhelník a každou jeho hranu rozdělíme na polovinu. Získáme tak souřadnice *x* a *y* a z nich pomocí matematické funkce vypočteme výškovou souřadnici *z*. Vzniknou nám tři nové body, které spojíme a dostaneme čtyři nové trojúhelníky. Rekurzivně takto pokračujeme pro každé čtyři nově vzniklé trojúhelníky. Několik kroků dělení je na obrázku 6.2.



Obr. 6.2 – Dělení trojúhelníku

6.3 Operace split a merge

Nad trojúhelníkovou sítí definujeme dvě operace – Split a Merge (viz obr. 6.3). Operaci Split použijeme, pokud chceme síť zjemnit. Split spočívá v nahrazení trojúhelníků jeho čtyřmi potomky. Vzniknou nám tak tři nové body. Opačnou operací je Merge, v níž nahrazujeme naopak potomky jejich rodičem. Operaci Merge lze aplikovat pouze na trojúhelníky, které mají potomky. Pomocí těchto dvou operací lze přejít z libovolné reprezentace sítě na jinou.



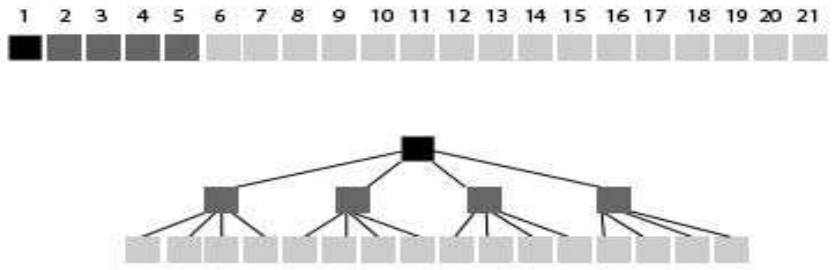
Obr. 6.3 – Operace split a merge

6.4 Prioritní fronty

Algoritmus využívá ke generování sítě dvě prioritní fronty. V jedné jsou trojúhelníky, které budeme dělit, a v druhé máme uloženy trojúhelníky, které budeme slučovat. Každý trojúhelník má přiřazenu určitou prioritu (výpočet priorit viz kapitola 6.6). Trojúhelníky jsou podle těchto priorit ve frontách seřazeny. Ve frontě s trojúhelníky určenými pro dělení je řazení sestupné, ve frontě s trojúhelníky pro slučování jsou řazeny vzestupně. Priority trojúhelníků jsou monotónní, to znamená, že priorita potomka není nikdy větší než priorita rodiče.

6.5 Metrika

Abychom se mohli rozhodnout, zda máme trojúhelník rozdělit či nikoliv, musíme použít nějakou metriku. Ve své implementaci jsem si zvolila metriku nazvanou variance, protože je podle [13] rychlejší než metrika použitá v původním článku [12]. Varianci určujeme pro každý trojúhelník v síti. Vypočítáme ji jako rozdíl mezi současnou reprezentací terénu a skutečným terénem - v našem případě matematickou funkcí. Jelikož se variance již dále nemění, můžeme si jí vypočítat předem. Jak již bylo zmíněno výše, při zjemňování sítě nahrazuji trojúhelník čtyřmi novými trojúhelníky a vznikají tak tři nové body. Varianci si vypočítám jednotlivě pro každý ze tří bodů a jako výslednou varianci trojúhelníku vezmu maximum z těchto hodnot. Výpočet začínáme základním trojúhelníkem a pak aplikujeme rekurzivně na všechny jeho potomky. Hodnoty variance nám vytvoří strukturu quadtree, kterou si uložíme do jednorozměrného pole (viz obr. 6.4) a u každého trojúhelníku si uchováme index do tohoto pole.



Obr. 6.4 – Přepsání quadtree do pole

Plochá místa terénu budou mít menší varianci a proto může být k jejich zobrazení použito menší množství trojúhelníků než na místa členitější.

6.6 Výpočet priorit

Prioritu přepočítáváme pro každý snímek pro všechny trojúhelníky v síti. Platí pro ní následující vztah:

$$priorita = \frac{variance}{vzdálenost} \qquad (6.1)$$

Vzdálenost počítáme jako eukleidovskou vzdálenost trojúhelníku od kamery. Souřadnici *x* vypočítáme jako jednu třetinu součtu *x* souřadnice bodu *A*, *x* souřadnice bodu *B* a *x* souřadnice bodu *C*. Stejným způsobem i pro souřadnice *y* a *z*. Díky varianci a vzdálenosti mají trojúhelníky dále od pozorovatele menší prioritu.

Pokud chceme při výpočtu priority zohlednit rozměr terénu, můžeme pro její výpočet použít vztah 6.2 [13].

$$priorita = \frac{2 \cdot rozmerMapy \cdot variance}{vzdalenost} \qquad (6.2)$$

kde rozmerMapy je šířka terénu, variance a vzdálenost mají stejný význam jako ve vztahu 6.1.

6.7 Algoritmus

V následujícím odstavci budu kvůli zjednodušení nazývat prioritní frontu, ve které jsou uloženy trojúhelníky určené pro dělení, frontu split. Frontu s trojúhelníky určenými pro slučování budu nazývat frontou merge.

Hlavní myšlenka algoritmu ROAM se dá vyjádřit pomocí dvou funkcí – funkce Split a funkce Merge. V hlavním programu využíváme těchto funkcí k vytvoření terénu s požadovanou přesností.

Split

Úkolem funkce Split je rozdělit trojúhelník na čtyři nové potomky, přiřadit novým trojúhelníkům sousedy, případně nějakému již existujícímu trojúhelníku přiradit nový trojúhelník jako souseda, vypočítat pro nové trojúhelníky priority a zařadit je správně do front. Do funkce Split posíláme trojúhelník, který má ve frontě split nejvyšší prioritu.

```
Function Split (trojúhelník)
    odstraníme trojúhelník z fronty split
    if trojúhelník má rodiče
        odstraníme rodiče z fronty merge
    if trojúhelník má sousedy, které mají rodiče
        odstraň rodiče sousedů z fronty merge
    vytvoř nové potomky
    přiřaď novým potomkům sousedy a rodiče
    vypočítej prioritu nových potomků
    vlož potomky do fronty split
    if  trojúhelník je slučitelný
        vlož trojúhelník do fronty merge
```

Merge

Funkce Merge má za úkol ze čtyř trojúhelníků vytvořit jeden, odstranit ukazatele na potomky, upravit vazby mezi sousedy a správně zařadit trojúhelníky do front. Do funkce Merge posíláme trojúhelník, který má ve frontě merge nejnižší prioritu.

```
Function Merge(trojúhelník)
        odstraníme potomky z fronty split
        odstraníme trojúhelník z fronty merge
        if trojúhelník má sousedy
                uprav vazby mezi sousedy
        if existují nové slučitelné trojúhelníky
                vlož je do fronty merge
        if trojúhelník má rodiče
                vlož rodiče trojúhelníku  do fronty merge
        vlož trojúhelník do fronty split
```

Main

V hlavní funkci využíváme funkcí Split a Merge, abychom dosáhli buď požadovaného množství trojúhelníků v síti nebo určité přesnosti. Funkce má dvě části, první část se provádí pouze při prvním framu, druhá část po zbytek běhu algoritmu.

```
if prvniFrame
{
        odstraň u  základního trojúhelníku odkazy na potomky a sousedy
        smaž frontu split i frontu merge
        vypočítej prioritu základního trojúhelníku
        vlož základní trojúhelník do fronty split
}
jinak
        přepočítej priority ve frontách split a merge

dokud není dosaženo požadované přesnosti nebo požadovaného počtu trojúhelníků nebo
maximální priorita ve frontě split je větší než minimální priorita ve frontě merge
{
        if terén je nepřesný nebo obsahuje velké množství trojúhelníků
                Merge
        jinak
                Split
}
```

Pro menší terény můžeme algoritmus ROAM zjednodušit. Místo použití dvou prioritních front využijeme pouze frontu split (popřípadě nemusíme fronty vůbec použít) a pro každý snímek nastavíme všechny proměnné na počáteční hodnoty a začneme dělit základní trojúhelník do požadované přesnosti.

Modifikovaný algoritmus bude vypadat takto:

odstraň u základního trojúhelníku odkazy na potomky a sousedy
smaž frontu split i frontu merge
vypočítej prioritu základního trojúhelníku
vlož základní trojúhelník do fronty split
dokud není dosaženo požadované přesnosti
    Split

6.8 Vykreslování terénu

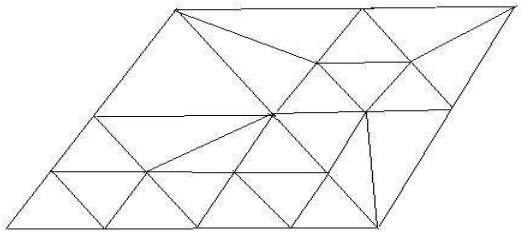
Díky zvolené reprezentaci trojúhelníkové sítě je výhodné použít k uložení bodů indexbuffer. Jeden bod může být totiž sdílen až šesti trojúhelníky a tím, že každý bod uložíme pouze jednou, ušetříme značné množsví paměti.

Nejprve musíme body uložit do vertexbufferu. Procházím tedy frontu split, do vertexbufferu vložím všechny tři body trojúhelníku a uložím si pozici, na které se ve vertexbufferu nacházejí. Pokud narazím na bod, který je součástí více trojúhelníků, díky příznaku poznám, zda se ve vertexbufferu nachází a pokud ano, tak ho přeskočím.

Současně s vertexbufferem si plním i indexbuffer, ze kterého pak trojúhelníky vykresluji.

6.9 Řešení trhlin

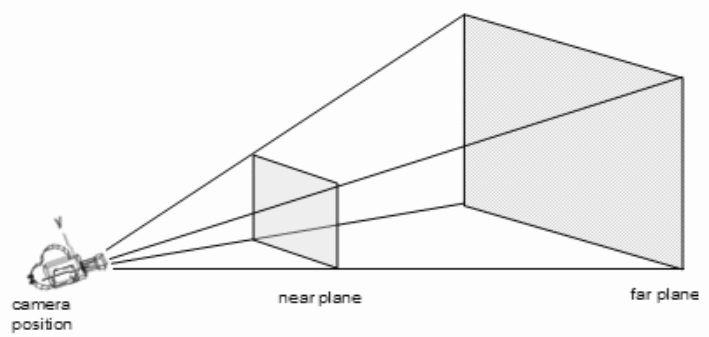
Nevýhodou zvolené reprezentace je, že pokud mám vedle sebe dva trojúhelníky, každý v jiné úrovni detailu, vznikne vždy trhlina. Trhliny ošetřuji až ve fázi vykreslování. Při procházení fronty split si u každého trojúhelníku zjistím, jestli má rozdělené sousedy. Pokud je některý ze sousedů rozdělený, vznikne v trojúhelníkové síti trhlina, která se projeví jako díra v terénu. Jako řešení se nabízí přidání nového bodu na hranu trojúhelníku do místa, kde k trhlině dochází. Pokud jsou rozděleni dva sousedi, přidávám dva nové body, pokud tři sousedi, v síti se objeví tři body navíc. Ukázka několika možností je na obrázku 6.5. V některých existujících modelech dělení trojúhelníků k trhlinám nedochází vůbec, ale výše uvedené dělicí schéma bylo požadováno vedoucím práce.



Obr. 6.5 – Řešení trhlin

6.10 View frustum

V počítačové grafice a zvláště pak při vykreslování terénů je dobré použít ořezávání a odstranit tak objekty, které pozorovatel nevidí. Prostor, který je kamerou viditelný, nazýváme view frustum. Pokud používáme perspektivní projekci, má tvar jehlanu s vrcholem v místě kamery. View frustum je definováno 6 stranami označovanými jako – near, far, bottom, top, left a right (obr. 6.6)



Obr. 6.6 – View frustum [14]

Ořezáváme tedy objekty, které leží mimo tento jehlan. Ořezávání provádíme předtím, než posíláme objekty na vykreslování. Jelikož prostor, ve kterém se pozorovatel nachází, je často mnohokrát větší než ten, co právě vidí, je ořezávání značným urychlením. Stěny jehlanu mohou být určeny pomocí matice projekce a matice pohledu.

6.11 Možná vylepšení

Geomorphing

Pokud měníme jednu úroveň detailu na jinou, může se změna projevit „poskočením“ vektoru. Aby změna nebyla tak patrná, používá se geomorphing, pomocí kterého přecházíme plynule z jedné úrovně do druhé. Někdy ale může být výhodnější místo použití morfování vykreslování přesnější trojúhelníkové sítě.

Textury

Dalším možným vylepšením této práce je zlepšení způsobu aplikace textury tak, aby krajina působila realističtějším dojmem.

6.12 Výsledky

Algoritmus [12] byl testován na počítači Indigo2Silicon Graphics s Maximum Impact grafickým hardwarem a procesorem R10000. Ve výsledcích v [12] jsou uvedeny doby provádění jednotlivých částí algoritmu, jako je ořezávání, počítání priorit a operace split a merge při počtu 3000 trojúhelníků. Na počítači Silicon Graphics Onyx s procesorem R10000 a grafickou kartou Infinite Reality byly prováděny testy se 6000 trojúhelníky. Je zjišťován také průměrný počet operací split a merge v jednom snímku. Podobné testy jsem provedla i se svým programem.

Jako testovací stroj jsem použila počítač s procesorem AMD Athlon 1,36GHz, RAM 384MB, grafická karta NVidia GeForce FX 5200 a operačním systémem Windows XP. Při vykreslování 3000 trojúhelníků se doba provádění algoritmu pohybovala okolo 32ms, z čehož přibližně polovinu zabralo naplňování vyrovnávací paměti vrcholy a samotné vykreslování. Pokud jsem počet trojúhelníků zvětšila na 6000, klesl počet snímků za sekundu k hodnotě 12. Dále jsem zjišťovala ke kolika změnám v trojúhelníkové síti během jednoho snímku dojde. Průměrný počet operací split a merge v jednom snímku byl okolo 0,8. Počty operací split a merge se pohybovaly mezi 0 až 40.

Pokud své výsledky srovnám s výsledky uvedenými ve [12], tak při testování s 3000 trojúhelníky mi vyšly velmi podobné hodnoty, při vykreslování 6000 trojúhelníků jsem dosáhla jen třetinového počtu snímků za sekundu.

7. Závěr

V bakalářské práci byly popsány metody generování krajin pomocí fraktální geometrie. Dále bylo uvedeno základní rozdělení Level of Detail metod, jejich aplikace při vykreslování terénů a přehled některých používaných algoritmů při zjednodušování terénů.

Implementován byl algoritmus ROAM s několika modifikacemi. Změny se týkaly hlavně reprezentace sítě a použitých metrik.

Výhodou zvolené reprezentace trojúhelníkové sítě je menší množství bodů, které je nutné ukládat při vykreslování, protože je bod téměř vždy sdílen šesti body. Další výhodu spatřuji v jednodušším způsobu ošetřování trhlin, protože k ošetření trhliny potřebuji maximálně přidat do sítě tři nové vektory. V původní verzi algoritmu ROAM se řeší trhliny rekurzí, která může být pro některé terény drahou operací.

Dalšími možnými vylepšeními práce je geomorphing nebo zlepšení textur.

Přehled zkratek

LOD - Level of Detail

TIN - nepravidelná trojúhelníková síť

ROAM - Really-Optimally Adapting Meshes

VIPM - View independent Progressive Meshes

CPU - procesor

Literatura

[1] http://kmlinux.fjfi.cvut.cz/~pausp1/html/skola/fraktaly/reserse.htm
[2] http://www.chaos.host.sk
[3] http://www.gymnachod.cz/~preclik/czech/mff_mgr/fraktaly.htm
[4] Žára, J., Beneš, B., Sochor, J., Felkel, P.: Moderní počítačová grafika, Computer Press, 2004
[5] Luebke, D., Reddy, M.: Level of Detail for 3D Graphics, 2003
[6] Masarykova univerzita v Brně, přednášky předmětu Počítačová grafika, www.fi.muni.cz/~ptx/PA010/Slides/PA010_8.pdf
[7] Röttger, S., Heidrich, W., Slusallek, P., Seidel, HP.: Real-Time Generation of Continuous Level of Detail for Height Fields, Technical report 13/1997, Universität Erlangen-Nürnberg
[8] Lindstrom, P., Pascucci, V.: Visualization of Large Terrains Made Easy, Proceedings of IEEE Visualization 2001, October 2001
[9] Ulrich, T.: Rendering Massive Terrains using Chunked Level of Detail Control, Presented at "Super-size it! Scaling up to Massive Virtual Worlds" course at SIGGRAPH 02, 2002
[10] Hoppe, H.: Progressive Meshes, International Conference on Computer Graphics and Interactive Techniques, 1996
[11] Hoppe, H.: Smooth View-Depend Level of Detail Control and its Application to Terrain Rendering, IEEE Visualization 1998, Oct 1998
[12] Duchaineau, M., Wollinski, M., Sigeti DE., Miller, M.,: ROAMing Terrain: Real-time Optimally Adapting Meshes, In Proceeding of the Conference on Visualization ´97, Oct 1997
[13] http://www.cs.sun.ac.za/~henri/
[14] http://www.gamasutra.com/features/20000403/turner_01.htm
[15] http://www.lighthouse3d.com/opengl/viewfrustum/

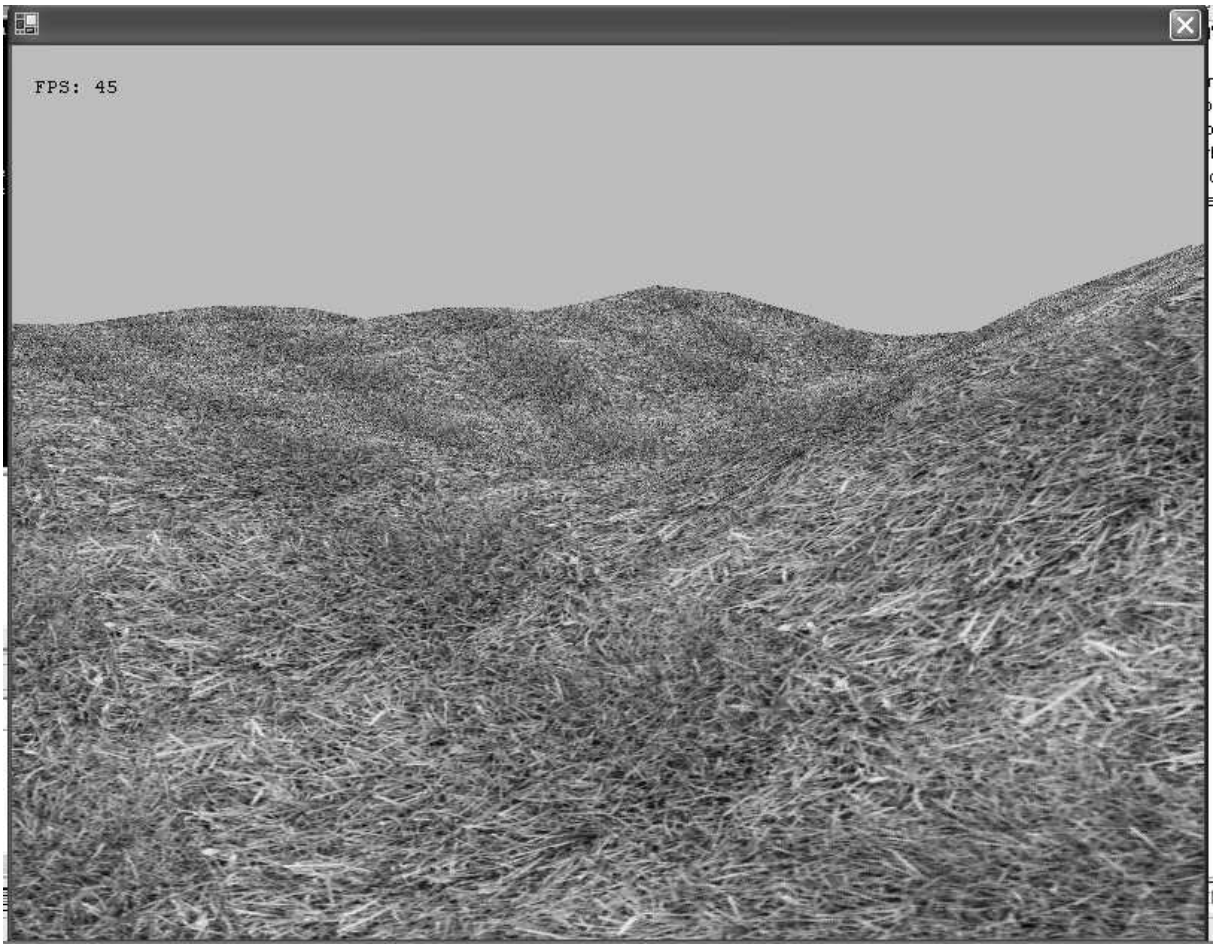
Příloha A - Uživatelská dokumentace

1. Přeložení a spuštění programu

Na přiloženém CD je k dispozici spustitelný soubor Roam.exe. Pokud si chcete program sami přeložit, je nutné mít nainstalované vývojové prostředí Microsoft Visual Studio .NET a DirectX 9.0c SDK.

2. Ovládání programu

Po spuštění programu se objeví následující obrazovka:



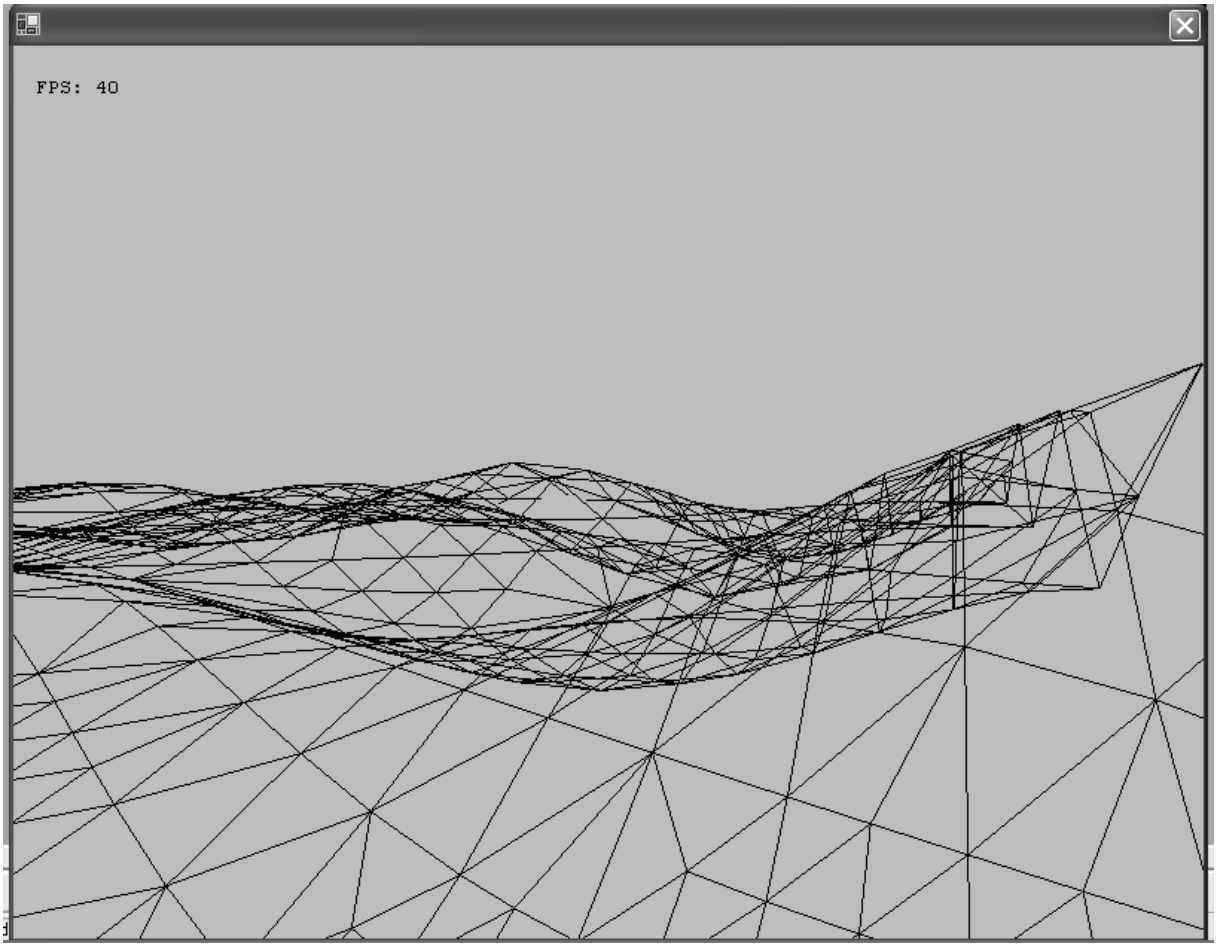
Obr. 25 - Okno po spuštění programu

Uživatel se může po krajině pohybovat. Způsobů je hned několik. Implicitně je nastaveno ovládání, kdy si uživatel sám pomocí kláves volí směr pohybu. Pohyb dopředu je možný pomocí klávesy W, dozadu pomocí klávesy S, doprava pomocí klávesy D a doleva pomocí klávesy A. Pomocí myši lze ovládat směr pohybu. Pohyb myší je však omezen tak, aby se nebylo možné s kamerou protočit. Pokud chcete, aby se kamera pohybovala sama plynule dopředu, stiskněte klávesu O.

Aby bylo možné pozorovat činnost algoritmu, je výhodné přepnout ze zobrazení solid, které je implicitně nastaveno, do zobrazení wireframe. Tuto akci může uživatel provést pomocí klávesy U. Vrátit se s kamerou do počáteční polohy je možné stisknutím mezerníku.

Vlevo nahoře v okně je vidět počítadlo snímků za sekundu (FPS). Program lze ukončit pomocí klávesy Esc.

Nápověda k programu je po celou dobu běhu programu zobrazena na konzoli.



Obr. 26 - Krajina v zobrazení wireframe

Příloha B - Programátorská dokumentace

Program byl vytvořen v programovacím jazyce C# s použitím DirectX 9.0c a odladěn ve vývojovém prostředí Microsoft Visual Studio .NET.

Program je rozdělen do těchto tříd:

RenderForm.cs
obsahuje hlavní smyčku programu, inicializaci DirectX, reakce na události myši a klávesnice, nastavení textur, počítání frameratu a vykreslování

TriangleNode.cs
definuje strukturu pro uložení trojúhelníku

Roam.cs
obsahuje funkce potřebné pro samotný algoritmus ROAM tj. výpočet priorit, operace Split a Merge, řešení trhlin, ořezávání

Program lze nalézt na přiloženém CD. Obsah CD je uveden v souboru readme.txt.

Souhlasím s prezenčním půjčováním práce v knihovně