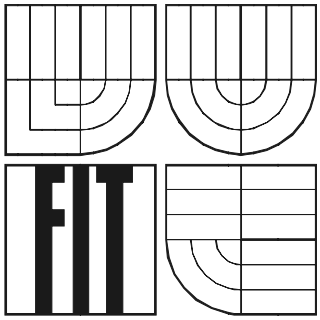
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Diplomová práce

2005 Martin Havlíček

Knihovna pro práci s výškovými mapami

1. Prostudujte si teorii renderování otevřených virtuálních scén a virtuální krajiny. Využijte zkušeností z předchozí práce v této oblasti.
2. Na základě předchozího studia navrhněte knihovnu pro práci s výškovými mapami s výhledem na možné budoucí rozšíření o algoritmy „nekonečné“ krajiny a technik LOD.
3. Implementujte knihovnu a vybrané algoritmy pro „nekonečnou“ krajinu.
4. Výsledky demonstrujte v jednoduché grafické aplikaci.
5. Vyhodnoťte zkušenosti a diskutujte výsledky.
6. Práci publikujte na internetu.

Tato strana je ve vytištěné zprávě nahrazena originálem zadání.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pečivy.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Martin Havlíček

Děkuji Ing. Janu Pečivovi za cenné rady a připomínky, které přispěly k dokončení této diplomové práce a také předcházejícího ročníkového a semestrálního projektu.

Abstrakt

Práce spadá do oblasti počítačové grafiky a úzce souvisí s problematikou virtuální reality.

Jsou v ní představeny výškové mapy, které tvoří základ pro vytvoření 3D modelu virtuální krajiny. Je naznačeno poslání třídy zastřešující operace s výškovými mapami.

Předložen je postup, kterým lze zajistit hladké navazování samostatně generovaných výškových map, a řešení problémů, které se při jeho praktické realizaci vyskytly.

V neposlední řadě je představen návrh grafického 3D enginu, který zajišťuje iluzi nekonečné virtuální krajiny. Jsou popsány jeho komponenty, naznačeno jeho využití a navrženy náměty pro jeho případné budoucí rozšíření.

Klíčová slova:

Virtuální realita, terén, krajina, 3D model, 3D engine, Open Inventor.

Abstract

This work comes under the area of computer graphic and it is closely related to virtual reality problems.

It introduces height maps, that are essential for creating 3D models of virtual landscape. The purpose of the class that provides operations with height maps is mentioned.

The work describes the way how to ensure correct connecting of independently generated height maps and the solution of problems that came out during its practical realization.

3D engine that assures the illusion of pseudo-infinity of virtual terrain is introduced as well. Its components are described, its usage is mentioned and suggestions for further development in the future are discussed.

Key Words:

Virtual reality, terrain, landscape, 3D model, 3D engine, Open Inventor.

Obsah

1. Úvod ... 8
2. Výchozí stav, projekty ve světě ... 9
2.1 Terragen ... 9
2.2 SIM Scenery a TGS TerrainViz ... 10
2.3 FlightGear ... 11
2.4 Stanovení základních cílů projektu ... 12
3. Výškové mapy ... 13
3.1 Základní přiblížení ... 13
3.2 Algoritmy generování výškových map ... 13
3.2.1 Midpoint Displacement ... 13
3.2.2 Circles algoritmus ... 15
3.2.3 Fault Formation ... 16
3.3 Vyhlazování hodnot výškových map ... 17
3.3.1 Lineární filtrování (filtrování po pásech) ... 17
3.3.2 Maticové filtry ... 17
3.4 Třída pro práci s výškovými mapami ... 18
4. Navazování výškových map ... 20
4.1 Porovnání možných přístupů ... 20
4.2 Navazování v okamžiku generování mapy ... 21
4.2.1 Průběh vývoje algoritmu ... 21
4.2.2 Mapy s překryvy - konečné řešení ... 24
4.3 Navazování nezávisle generovaných map ... 25
4.3.1 Principy algoritmu ... 25
4.3.2 Matice vah ... 27
4.3.3 Normály na okrajích čtverců ... 29
5. Engine pseudonekonečné krajiny ... 31
5.1 Schema enginu ... 31
5.2 Základní schema aplikace ... 32
5.3 Generátory, fronta úloh ... 33
5.4 Databáze výškových map ... 34
5.5 Objekty, zpracování událostí ... 38
5.6 Možnosti využití enginu ... 40
6. Implementace ... 41
6.1 Open Inventor ... 41
6.2 Správa grafu scény ... 41
6.3 Terrain Engine API ... 42
6.4 Náměty na pokračování projektu ... 43

7. Závěr..... 45
8. Literatura..... 46
A. Jak napsat aplikaci využívající engine?..... 47
B. Jak napsat nový generátor?..... 49
C. Ovládání ukázkových aplikací..... 50
D. Obrazová příloha..... 51

1. Úvod

Práce spadá do oblasti počítačové grafiky a úzce souvisí s problematikou virtuální reality. Zabývá se generováním a zobrazováním virtuálního zemského povrchu. Mým dlouhodobým cílem v této oblasti bylo navrhnout algoritmy, které by poskytovaly iluzi nekonečnosti produkovaného modelu. Tento cíl se v rámci diplomového projektu podařilo zrealizovat, důkazem budiž aplikace demonstrující funkčnost grafického 3D enginu, jehož teoretické předpoklady a myšlenky jsou hlavním obsahem této zprávy.

Diplomová práce přímo navazuje na můj předchozí výzkum v rámci ročníkového projektu s názvem „Generátor virtuální krajiny“ a semestrálního projektu.

V ročníkovém projektu jsem se zabýval algoritmy, které slouží k výpočtu výškových map a problematikou jejich vylepšení do té míry, aby bylo možno mapy spojovat a vytvářet tak rozsáhlé scenérie, v ideálním případě neomezeně rozlehlé. Byl navržen způsob, kterým lze kýženého cíle dosáhnout. Tento postup se ale v průběhu další práce ukázal jako neperspektivní, a proto byl v rámci semestrálního projektu navržen způsob jiný. Obsahem semestrálního projektu byl zčásti taktéž teoretický návrh enginu, který byl posléze implementován a zajišťuje nyní dynamické generování terénu dle pohybu uživatele ve scéně.

Kapitola 2, nazvaná „Výchozí stav, projekty ve světě“ slouží k porovnání práce s projekty, které se zemským povrchem v počítačové grafice zabývají a jejich specifikace je dostupná na internetu. Porovnány jsou různé přístupy k tématu.

Výškové mapy obecně jsou předmětem třetí kapitoly této práce. Obsahově vychází z ročníkového projektu a je obohacena o popis třídy, která byla navržena a implementována letos. Jsou v ní popsány vybrané algoritmy generování výškových map s důrazem na postupy důležité pro další úvahy.

Čtvrtá kapitola se zabývá navazováním výškových map a skládáním rozlehlých oblastí povrchu z menších celků. V první části je popsán algoritmus navržený v rámci ročníkového projektu. Ten se snaží zajistit návaznost map již v okamžiku jejich výpočtu. Následuje pak popis postupu preferovaného v současné době a prakticky ověřeného v implementovaném enginu. Ten starost o hladké navázání jednotlivých stavebních bloků terénu odkládá na později a výškové mapy jsou tedy generovány absolutně nezávisle na sobě.

Popis enginu, který je hlavním mozkem výsledné aplikace, je obsahem následující kapitoly. Jsou v ní uvedeny principy fungování jeho jednotlivých částí a je vysvětlen princip tzv. generátorů, které jsou jeho stěžejními komponentami. Díky nim lze dynamicky tvořit terén aniž by byla ohrožena plynulost běhu aplikace. V závěru této kapitoly jsou nastíněny možnosti využití navrhovaného systému.

Kapitola šestá je věnována vybraným podrobnostem implementace projektu, jsou v ní též zmíněny základní principy použité grafické knihovny Open Inventor. Jako poslední v řadě jsou uvedeny náměty na vylepšení projektu a případná pokračování vývoje v této oblasti.

2. Výchozí stav, projekty ve světě

S rozmachem virtuální reality bylo jistě jako jeden z prvních problémů třeba vyřešit zobrazování zemského povrchu. Ať už je účel výsledné aplikace jakýkoliv (hra, simulátor pro výcvik pilotů dopravních letadel, simulátor jízdy v automobilu pro autoškoly, vojenská simulace), je třeba se s krajinou nějakým způsobem vypořádat.

Při průzkumu této oblasti v rámci předchozí práce jsem dospěl k názoru, že se ve světě vývoj ubíral zhruba třemi směry.

První cestou, kterou reprezentuje například známý (a pro nekomerční účely volně dostupný) program Terragen, je renderování fotorealistických, krásně vypadajících a propracovaných, ovšem statických scenérií a animací.

Druhý v pořadí je směr využívající přesná geografická data a snímky z družic k vytvoření dokonalé kopie skutečného zemského povrchu. Tyto přístupy mají zcela nepochybně svoje opodstatnění například při zmiňovaném výcviku pilotů, nebo při plánování rozsáhlých inženýrských projektů, které mají ambice zásadně změnit tvář krajiny (vodní díla).

Třetí cestou se ubírají „méně vážné“ aplikace zejména z oblasti virtuální zábavy, počítačové hry. Charakteristickým rysem vývoje takových aplikací je hledání kompromisu mezi pokud možno realisticky vyhlížejícím terénem a rychlostí jeho generování a zobrazování.

V následujícím textu představím vybrané projekty, jejichž specifikace je dostupná na internetu. Není cílem podat vyčerpávající informace o produktech samotných, spíše jsem chtěl nastínit jejich deklarované schopnosti a porovnat jejich filosofii s mým vlastním výzkumem. V závěru této kapitoly budou totiž stanoveny cíle, které jsem se snažil v rámci vlastní práce splnit.

2.1 Terragen

Aplikaci Terragen vyvíjí britská firma Planetside Software. Program je pro nekomerční využití poskytován zdarma a lze ho získat z webových stránek produktu (viz [6]). Dostupné jsou verze pro operační systémy z rodiny MS Windows a pro Macintosh.

Samotní autoři popisují svůj produkt jako generátor krajiny vytvořený s cílem poskytovat fotorealistické obrázky a animace. Podporováno je několik metod generování terénu, výsledek lze ovšem také manuálně upravovat. Povrch je potažen texturami. Lze renderovat detailní vodní hladiny, včetně vln, odlesků slunce a odrazů okolí. Nebe zakrývají mraky, které vrhají stíny na povrch, lze definovat a pomocí mnoha parametrů upravovat mlhu, veškeré osvětlování je pojato realisticky pomocí simulace průchodu slunečních paprsků atmosférou.

Program byl dle údajů autorů využit ve filmu, v televizi, při výrobě videoklipů, počítačových her, knih, časopisů a reklam.

Použití popisovaného programu je přitom v základech velmi jednoduché. Ze všeho nejdříve

je třeba stanovit rozměry generované části krajiny. Poté si zvolíme metodu a necháme generátor vytvořit tzv. výškovou mapu. Poté určíme pozici a směr pohledu kamery, která bude scénu zobrazovat. Nastavíme pozici slunce na obloze, upravíme vzhled a hustotu mraků, nastavíme úroveň vodní hladiny a můžeme si nechat vyrenderovat první náhled na právě vytvořenou krajinu. Jsme-li s výsledkem spokojeni, zvolíme velikost výsledného obrázku, nastavíme kvalitu na maximum a necháme program počítat.

Obrázek 2.1 - Ukázka výstupu aplikace Terragen, vygenerování originálního obrázku o rozměrech 640x480 pixelů trvalo na počítači s procesorem AMD AthlonXP 1,4 GHz zhruba 5 minut.

Tvorbu animací jsem v praxi nezkoušel.

Terragen nemá ambice být nasazen v nějaké interaktivní aplikaci, ve které by byla kritická rychlost odezvy na podněty uživatele. Je zde uveden jako reprezentant prvního směru vývoje, směru, který klade důraz na vizuální dojem.

2.2 SIM Scenery a TGS TerrainViz

Uvedené produkty se zaměřují na real-time zobrazování rozsáhlých dat reprezentujících zemský povrch, které získávají ze specializovaných databázových struktur. Oba produkty jsou postaveny nad grafickou knihovnou Open Inventor, kterou využívá i tento projekt.

SIM Scenery je nadstavba nad inventorovskou implementací Coin3D a vyvíjí ji norská firma Systems in Motion (SIM). Ačkoliv Coin3D je volně přístupný software, tato nadstavba je komerčním produktem. Na projektu TerrainViz pracuje firma TGS, která taktéž vyvíjí komerční implementaci TGS Inventor. Kvůli skutečnosti, že oba produkty jsou komerční záležitostí, neexistují detailní informace o použitých algoritmech a k dispozici je pouze výčet jejich schopností. Z pochopitelných důvodů jsem taktéž neměl možnost tyto programy vyzkoušet v praxi.

Z informací dostupných na [7] a [8] lze ale odvodit základní principy jejich fungování. Jsou založeny na využívání rozsáhlých databázích, ve kterých jsou uloženy detailní mapy skutečného zemského povrchu. Kromě samotných map popisujících nadmořskou výšku terénu obsahují také textury, které po aplikaci poskytnou dokonalý obraz dané oblasti. Tyto produkty poskytují API pro snadnou obsluhu a získávání dat z databází a zajišťují renderování modelu krajiny při dosažení vysoké frekvence snímků. Tu zajišťují použitím LOD [= Level Of Detail] algoritmů tak, že před zobrazením každého snímku přepočítají trojúhelníkovou síť tvořící model krajiny, aby při zachování pevného počtu trojúhelníků byl výsledný model co nejlepší.

SIM Scenery je využívána několika norskými firmami zabývajícími se těžbou ropy a zemního plynu pro zobrazování mořského dna. TGS TerrainViz byl využit ve vojenských simulacích pro francouzské ministerstvo obrany.

2.3 FlightGear

FlightGear je letecký simulátor vyvíjený komunitou přispěvatelů z celého světa jako open source produkt distribuovaný dle licence GNU GPL. Podrobné informace včetně zdrojových kódů a dokumentace lze získat z webových stránek [9].

Jeho historie sahá do roku 1996, kdy hlavní podíl na práci měla univerzita v Berkeley.

Hlavní myšlenkou, která stála u zrodu tohoto projektu, byla touha vyvinout realistický civilní letecký simulátor, který by byl multiplatformní a snadno rozšiřovatelný. Simulátor by měl obsahoval propracovaný fyzikální model letu s ambicemi získat oficiální akreditaci jako trenažér použitelný při výcviku pilotů.

Z hlediska tohoto projektu je nejdůležitější, jakým způsobem se autoři vyrovnali se zemským povrchem a jeho zobrazováním. FlightGear je v tomto směru velmi důkladný a v současné době se chlubí kompletním pokrytím celé zeměkoule. Celý zemský povrch je rozdělen na oblasti vytyčené poledníky a rovnoběžkami, pouze data terénu v současné době dle informací na [9] zabírají prostor 3 DVD nosičů. Pro spuštění aplikace ovšem stačí mít jejich určitou část. Data jsou za běhu nahrávána z disku pomocí samostatného vlákna programu, tak aby byl minimalizován dopad těchto operací na plynulost renderování.

Textury povrchu odpovídají skutečnosti, nejvíce propracovány jsou okolí a samotné modely letišť. V touze po simulaci reality jde projekt tak daleko, že obsahuje dokonce realistické střídání dne a noci s tím, že poloha Slunce ve dne a pozice hvězd a fáze Měsíce na noční obloze odpovídají danému místu a času.

Vidíme, že tento simulátor klade důraz na fyzikální model letu a data povrchu získává z rozsáhlých, předem připravených struktur.

2.4 Stanovení základních cílů projektu

Z předchozího textu vyplývá několik skutečností, ze kterých při stanovení cílů nového projektu vycházíme. Byly již řešeny otázky zobrazování rozsáhlých území se zachováním solidní rychlosti dostatečné pro aplikace využívající vstupy od uživatele v reálném čase. Tyto aplikace ovšem ke svému běhu potřebují zdrojová data, která reprezentují terén, který samotný program již „pouze“ zobrazuje.

Taktéž již byly navrženy algoritmy, které dokáží pomocí speciálních algoritmů vygenerovat zemský povrch. Tyto algoritmy ovšem pracují pouze s omezenými rozměry mapy a kvůli složitosti a časovým nárokům nejsou vhodné pro real-time nasazení.

Ideální by bylo nalézt nějaký kompromis. Nalézt cestu, která umožní generovat terén bez nutnosti udržování složité a rozsáhlé statické databáze, která ovšem bude také schopna nezávisle vygenerované části krajiny spojovat do velkých celků. To vše dynamicky, v reálném čase, podle pohybu pozorovatele ve scéně, bez patrných prodlev způsobených výpočtem nové geometrie 3D modelu. Dalo by se říct, že výzkum se tedy ubíral směrem, který lze zařadit do třetí větve zmiňované v samotném úvodu této kapitoly. Nyní si však dovolím tvrdit, a z dalšího textu to vyplyne, že rozšíření navrženého enginu tak, aby se přibližoval i k ostatním dvěma proudům bude v budoucnu jistě možné.

Na závěr ještě krátká rekapitulace výchozích poznatků.

Stojíme-li před požadavkem zobrazit na obrazovce počítače trojrozměrný model zemského povrchu, musíme ihned zpočátku zvolit mezi dvěma diametrálně odlišnými přístupy.

Prvním řešením může být předem připravený 3D model virtuálního terénu, nebo dokonce celé scény, ve které se bude uživatel pohybovat. Výhodou tohoto přístupu může být dokonalá realističnost, závisející pouze na kvalitě a propracovanosti modelu. Mínusem jsou stále stejný vzhled scény a potřeba prostoru pro uložení takového modelu.

Opačným přístupem jsou dynamicky generované modely povrchu. Touto problematikou se tato práce zabývá. Pozitivem tohoto přístupu je variabilita terénu, které lze pouhou změnou několika parametrů generátoru dosáhnout.

Samotné řešení probíhalo v několika etapách. Bylo třeba vybrat a implementovat vhodnou metodu generování zemského povrchu a vymyslet způsob, jakým nezávisle získané mapy spojovat do větších celků. Bylo nutné navrhnout engine, tj. jakýsi mozek aplikace, který je zodpovědný za udržování konzistence mezi daty, dynamické generování map nových dle pohybu uživatele scénou a nakonec samotné vykreslování.

Jednotlivé etapy řešení jsou rozvedeny v následujícím textu.

3. Výškové mapy

3.1 Základní přiblížení

V této kapitole se budeme zabývat výškovými mapami jako prostředky k popisu zemského povrchu. Budou vysvětleny vybrané algoritmy operující nad daty výškových map a bude popsáno poslání knihovní třídy `TeHeightMap`, která je jedním z důležitých výsledků tohoto projektu. Obsahově tato kapitola čerpá z předchozího ročníkového projektu a projektu semestrálního.

Výškové mapy jsou důležité, protože jsou obvykle základem pro vytvoření 3D modelu zemského povrchu. Z výškové mapy lze totiž relativně jednoduše vytvořit model takový, jaký vyžaduje zvolená grafické knihovna (v našem případě Open Inventor). Takovým modelem je obvykle trojúhelníková síť.

Výškovou mapu si velmi jednoduše můžeme představit jako dvourozměrné pole čísel, jejichž významem je nadmořská výška povrchu v části prostoru, který je mapou reprezentován. Takto definovaná výšková mapa věrně odpovídá klasickým kartografickým mapám tak, jak je známe z běžného života.

Existuje mnoho různých algoritmů, které umožňují takovou mapu vygenerovat. Liší se navzájem nejen samotným postupem výpočtu, ale také charakterem krajiny, kterou jako výsledek poskytují. V následující podkapitole jsou popsány algoritmy, které jsem měl možnost vyzkoušet v praxi.

3.2 Algoritmy generování výškových map

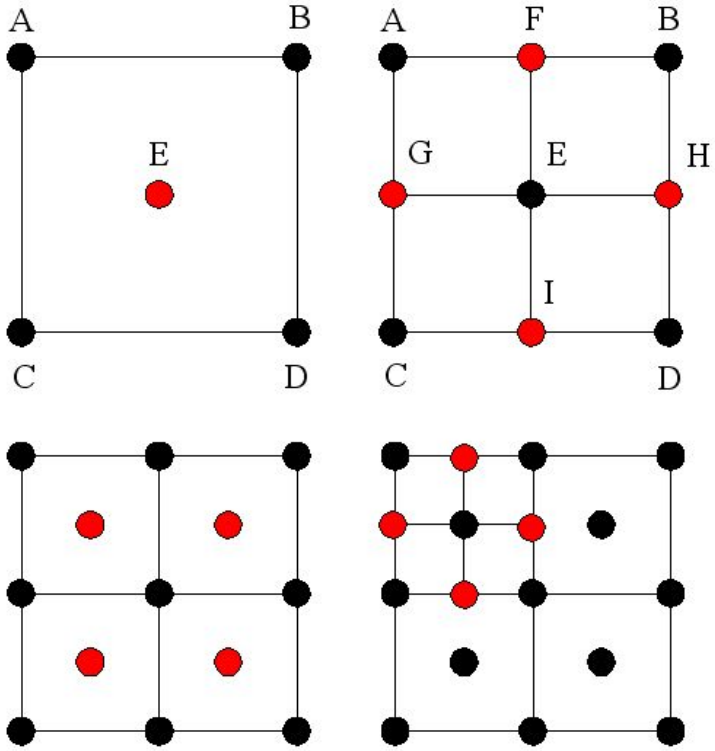
3.2.1 Midpoint Displacement

Midpoint Displacement (česky známý pod názvem "Přesouvání středního bodu") je elegantní, rekurzivní algoritmus. Funguje na principu dělení plochy výškové mapy na čtvrtiny. V každé iteraci náhodně přesune bod uprostřed aktuální plochy (tzn. zvýší nebo sníží jeho nadmořskou výšku) a body, které leží uprostřed jejich okrajů.

Princip je znázorněn na obrázku 3.1 na další stránce. Algoritmus začíná ve stavu, kdy jsou známy čtyři body v rozích tvořené plochy. Výšku bodu `E` uprostřed celé plochy získáme pomocí vzorce

```
E = (A+B+C+D)/4 + rand(d),
```
(1)

kde `A`, `B`, `C`, `D` jsou hodnoty v příslušných bodech a `d` je maximální povolený přesun pro danou iteraci, `rand(d)` je potom náhodné číslo ležící v intervalu <-`d`, `d`>.



Obrázek 3.1 - Princip činnosti algoritmu Midpoint Displacement.

Následuje výpočet bodů uprostřed okrajů plochy. Ty získáme pomocí rovnic

```
F = (A+B+E)/3 + rand(d)
G = (A+E+C)/3 + rand(d)
H = (B+E+D)/3 + rand(d)
I = (E+C+D)/3 + rand(d)
```
(2)

Uvedené dva kroky se rekurzivně opakují. Na obrázku 3.1 jsou černě označeny body, jejichž výšku v dané iteraci známe a využijeme tuto znalost pro výpočet bodů nových, označených červeně.

Algoritmus poskytuje dle [1] dobře vypadající chaotický terén, horské scenérie. Jeho nevýhodou jsou omezení na vyžadované rozměry generované mapy. Ty musí být kvůli rekurzivnímu charakteru výpočtu mocninami dvou.

Za zcela zásadní negativum jsem ovšem z hlediska tohoto projektu velmi dlouho považoval nemožnost jakkoliv rozumně popsat proces utváření mapy tak, aby z této informace mohla čerpat

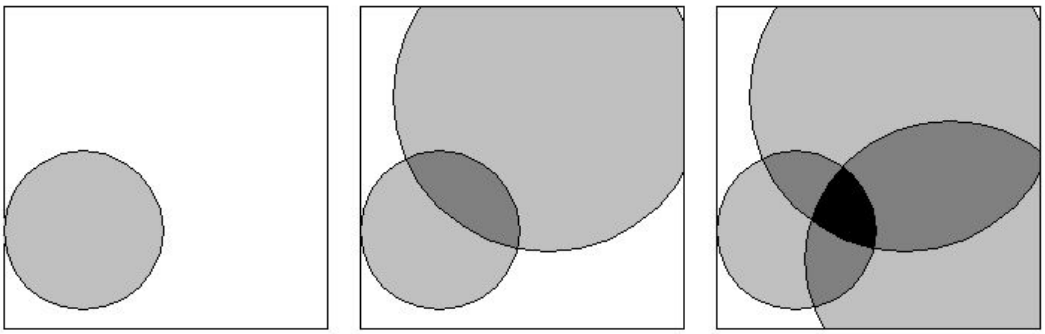
výšková mapa sousední. Tyto informace byly třeba v první části práce (v ročníkovém projektu), aby bylo možné zajistit hladké navázání jednotlivých map v prostoru. Této problematice se velmi podrobně věnuje kapitola 4, na tomto místě je pouze vhodné předem sdělit, že v rámci semestrálního projektu byl navržen a v praxi ověřen algoritmus, který žádné podobné informace nevyžaduje a může tedy využívat zcela libovolný postup výpočtu výškové mapy.

Jedinou nevýhodou algoritmu Midpoint Displacement tedy zůstala omezení na rozměry generované mapy.

3.2.2 Circles algoritmus

Na tento postup jsem narazil na internetu [3] a na první pohled se zdál sympaticky jednoduchý a terén, jež produkuje, dobře vyhlížející.

Iterace metody začínají náhodnou volbou bodu na mapě. Poté se zvolí číslo reprezentující poloměr kružnice se středem ve vybraném bodě, uvnitř které je povrch následně vyvýšen nebo snížen o nějakou hodnotu. První tři iterace znázorňuje obrázek 3.2. Úroveň šedé představuje nadmořskou výšku v daném bodě (vždy došlo k přičtení hodnoty).



Obrázek 3.2 - Princip činnosti Circles algoritmu.

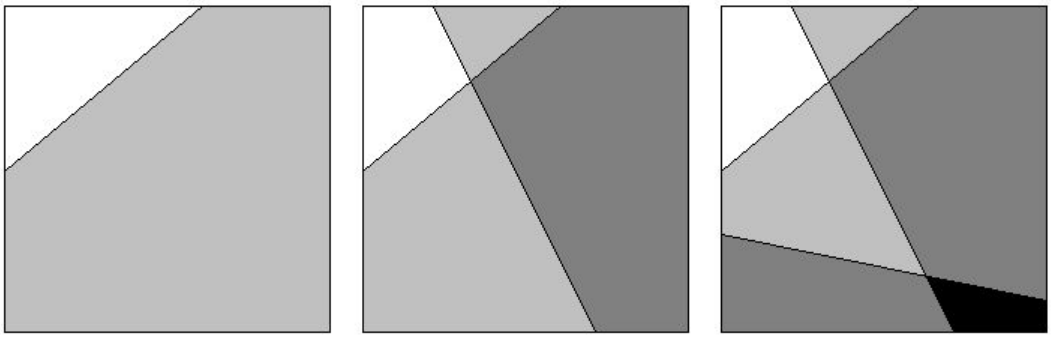
Výhod tohoto postupu je několik. Na rozdíl od Midpoint Displacement algoritmu nejsme nijak omezováni co do rozměrů požadované výškové mapy. Z hlediska navazování výškových map již v okamžiku jejich generování by se postup tvorby mapy navíc dal snadno parametrizovat (významné jsou informace o středu a poloměru kružnice, dále je důležité vědět o jakou hodnotu byla vnitřní oblast kružnice upravena).

Nevýhody této metody jsem odhalil až poté, co jsem ji implementoval. Pro to, aby povrch „vypadal hezky“ bylo třeba velké množství iterací (kolem 1000). Výpočet je velmi náročný na výkon procesoru (dle algoritmu popsaného ve [3] musíme v každé iteraci pro každý bod mapy vypočíst mj. jeho Eukleidovskou vzdálenost od středu kružnice a některou goniometrickou funkci kvůli zaoblení – místo vyvýšení s kolmými stěnami tvoří hladké „kopečky“). Spojení obou aspektů vyústilo v nepřijatelně dlouhou dobu výpočtu.

Z tohoto důvodu, a s ohledem na budoucí nutnost generovat stále nové a nové čtverce terénu ve snaze předstírat jeho nekonečnost, jsem nakonec jako základ pro implementaci zvolil Fault Formation algoritmus.

3.2.3 Fault Formation

Název metody by se dal do češtiny přeložit jako „formování pomocí zlomů“. Tento opis v zásadě nastiňuje princip algoritmu. Dalo by se říci, že tímto způsobem velmi zjednodušeně simuluje procesy při utváření některých reálných pohoří. Přitom algoritmus samotný není ve své základní variantě nikterak složitý. Protože je základem následujících úvah, rozeberu ho zde velmi podrobně. První tři iterace znázorňuje obrázek 3.3. Úroveň šedé představuje nadmořskou výšku v daném bodě.



Obrázek 3.3 - Princip činnosti algoritmu Fault Formation.

Výchozím bodem pro Fault Formation algoritmus je mapa obsahující v každém místě nulu. Následující úkony provádíme v cyklu. Vybereme zcela náhodně dva body na mapě. Tyto body vytyčují přímku, podél které dojte v terénu ke zlomu. Všechna místa na jedné straně od přímky (zlomu) vyvýšíme o stejnou hodnotu, ostatní body neměníme. Počet iterací tohoto cyklu pro získání realisticky vyhlížejícího kusu krajiny se pohybuje kolem čísla 150. V praxi se osvědčilo hodnotu o kterou zvyšujeme konkrétní část terénu svázat s pořadovým číslem iterace tak, aby zpočátku byla velká a postupně se zmenšovala. Tím způsobíme, že charakter terénu je rychle načrtnut v několika prvních iteracích a následující zlomy již dále vytvářejí drobnější povrchové detaily.

Fault Formation algoritmus v této základní verzi nám tedy dokáže vygenerovat výškovou mapu pro jednu oblast krajiny. Hlavním pozitivem této metody je ale fakt, že se jevila jako vhodná z hlediska řízení navazování jednotlivých map, což bylo v průběhu řešení ověřeno. Na rozdíl od předchozího algoritmu potřebuje k vytvoření „pěkně vypadajícího“ terénu řádově nižší počet iterací, což se příjemně odráží na rychlosti výpočtu. Neklade též žádná omezení na rozměr mapy.

Za nevýhodu by mohlo být pokládáno (podle [1] to nevýhoda je, podle mě to není na

závadu), že produkuje mírně zvlněný terén, nedají se od něho očekávat přirozeně vypadající divoká pohoří taková, jaká umí např. Midpoint Displacement.

3.3 Vyhlazování hodnot výškových map

Na povrch vzniklý pouze pomocí některého z uvedených algoritmů je vhodné před vykreslením aplikovat tzv. erozní filtr. Cílem filtrování je terén vyhladit – opět by se dalo říci, že se snažíme napodobit účinky povětrnostních vlivů (vítr, voda) na zemský povrch. Metody vyhlazování hodnot výškových map jsou obsahem této podkapitoly. Lze říci, že filtrování má na výsledek velký vliv a pomocí volby parametrů použitých filtrů lze tvořit různě zvlněný (různě vypadající) povrch.

3.3.1 Lineární filtrování (filtrování po pásech)

Tento algoritmus jsem převzal z [3]. Jedná se o základní postup, kterým lze uhladit vznikající terén.

Nadmořská výška každého bodu výškové mapy je při jeho aplikaci ovlivněna výškou bodu sousedního v závislosti na zvoleném parametru filtrace `k`. Tento parametr určuje míru vlivu sousedního bodu na bod právě počítaný. Je-li roven 1 nedochází k žádným změnám, čím více se blíží k nule, tím větší filtrování nastává.

Výpočet se řídí následujícím předpisem:

```
h[x,z] = h[x-1,z] * (1-k) + h[x,y] * k                    (3)
```

Vztah (3) platí pro řádky výškové mapy (souřadnice z) ve směru vzrůstajícího x. Aby bylo vše v pořádku, je nutné provést podobnou filtraci pro všechny řádky a sloupce výškové mapy a to v obou směrech. Pro dosažení lepšího efektu se navíc doporučuje celý proces filtrování několikrát zopakovat.

V průběhu řešení jsem dospěl k názoru, že je lepší raději několikrát přefiltrovat krajinu s vyšším koeficientem filtrace, než jedno „velké“ filtrování. V současné době je implicitně nastavena hodnota parametru `k` na 0.9 a filtrování provádíme pětkrát.

3.3.2 Maticové filtry

Maticové filtry jsou mocnějším prostředkem než výše uvedený filtr pásový. Ačkoliv jsem je

prakticky nevyzkoušel, myslím, že v tomto přehledu by neměly chybět.

Jsou reprezentovány čtvercovou maticí koeficientů, které určují vliv bodů v okolí na bod právě modifikovaný. Zatímco v předchozím případě byl aktuální bod ovlivněn pouze jediným sousedem, nyní přichází ke slovu celé okolí bodu. Rozměry matice jsou obvykle 3x3 a bývá dodrženo pravidlo, že součet prvků matice je roven 1. Pro matici 3x3 je tedy v každém místě matice uložena hodnota 1/9, pro matici 5x5 je to 1/25 atp.

Budeme-li uvažovat matici 3x3, pak ji lze obecně vyjádřit takto:

$$m = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{4}$$

Předchozí způsob filtrace lze vyjádřit maticí, jejíž prvek `m21=(1-k)`, `m22=k` a ostatní prvky jsou nulové.

Předpis pro výpočet hodnoty bodu vypadá následovně (převzato z [3]):

```
h[x,z] = m11*h[x-1,z-1] + m12*h[x,z-1] + m13*h[x+1,z-1] +
         m21*h[x-1,z]   + m22*h[x,z]   + m23*h[x+1,z]   +
         m31*h[x-1,z+1] + m32*h[x,z+1] + m33*h[x+1,z+1]      (5)
```

Jistá komplikace nastává pro body blízko okrajů mapy. Řešení spočívá v použití všech dostupných bodů a následném vydělení výsledku součtem použitých prvků matice.

3.4 Třída pro práci s výškovými mapami

Název celého projektu zní „Knihovna pro práci s výškovými mapami“, tudíž v něm má třída reprezentující výškovou mapu výsadní postavení. Nese označení `TeHeightMap` (veškeré třídy týkající se projektu mají v názvu prefix `Te` jako Terrain Engine). Její návrh byl také první věcí, na kterou přišla v letošním roce řada.

V současné době se, dle informací na [10], rozběhlo hned několik projektů souvisejících s virtuální krajinou. Bylo by jistě dobré, kdyby do budoucna existovalo jednotné rozhraní pro výpočet základní výškové mapy a základní operace s ní. V letošním roce se sice k tomuto ideálnímu stavu nepřiblížíme, věřím ovšem, že projekty, které odstartují v budoucnosti by mohly mých výsledků opravdu využít.

Již při předchozí práci bylo navrženo a posléze také implementováno několik algoritmů pracujících s daty výškové mapy. Jednalo se tehdy především o samotné generování krajiny. Cílem této třídy však bylo poskytnout programátorovi ucelenou nabídku operací, které by byly přístupné pomocí rozumně navrženého rozhraní. Nejde tedy pouze o výběr algoritmů užitečných pro další

práci, ale o celou množinu operací, které mohou mít nad výškovou mapou smysl.

Na tomto místě je vhodné podotknout, že se velmi pravděpodobně dříve či později objeví požadavek na nějakou novou funkci, která nás při návrhu třídy prostě nenapadla. To povede k přirozenému rozvoji a zvětšování možností třídy. Troufám si ale tvrdit, že základní množina operací je pokryta a jejich implementace je odladěna.

Zdrojové kódy jsou napsány v jazyce C++ a jsou volně dostupné pro veřejnost. Spolu s ostatními elektronicky odevzdávanými výstupy této práce jsou umístěny na stránkách „Inventor Projects 2005“ [10], které jsou věnovány všem projektům založeným na knihovně Open Inventor řešeným v letošním akademickém roce. Při praktickém programování jednotlivých metod jsem se snažil o maximální efektivitu kódu při zachování jeho jednoduchosti a čitelnosti. V současné době je třída využívána při tvorbě „nekonečného“ terénu a je nasazena v real-time 3D enginu.

Jednotlivé implementované operace by šlo velmi zhruba rozdělit do čtyř skupin.

První a pravděpodobně nejvyužitelnější skupinou jsou generátory terénu. Odpovídají jednotlivým algoritmům generování výškových map. V současnosti je dostupná sice pouze metoda formování pomocí zlomů (viz kapitola 3.2.3), avšak rozšíření o výsledky ostatních projektů nebude v budoucnu jistě nijak obtížné.

Druhou skupinou jsou filtry, které mají na výsledek taktéž velmi zásadní vliv. Filtrování povrchu je důležité pro vyhlazení krajiny a zlepšení výsledného vizuálního dojmu. K dispozici je lineární filtrování tak, jak bylo popsáno v kapitole 3.3.1.

Třetí skupinu tvoří operace kopírování hodnot a aritmetické operace působící nad daty zdrojové a cílové výškové mapy. U těchto funkcí lze přesně specifikovat zdrojové a cílové pravoúhlé oblasti v jednotlivých mapách.

Do poslední skupiny lze zařadit funkce využívající statistické údaje (minimální, maximální a průměrnou hodnotu) a vertikální posuvy (ve smyslu korekce všech hodnot na mapě stejnou hodnotou). Pomocí těchto operací lze řídit nadmořskou výšku výsledné krajiny.

4. Navazování výškových map

4.1 Porovnání možných přístupů

Stojíme-li před problémem navazování nezávisle generovaných výškových map do větších celků, máme v zásadě možnost zvolit mezi dvěma diametrálně odlišnými přístupy. Oba dva jsem si prakticky vyzkoušel a myslím, že je tedy mohu odpovědně porovnat a vysvětlit použité algoritmy.

Prvním přístupem, který jsem prosazoval v rámci ročníkového projektu bylo zajišťování hladkého navazování výškových map již v okamžiku jejich generování. Tento způsob řešení má svoje výhody i nevýhody. S odstupem času musím přiznat, že v porovnání s metodou popsanou v kapitole 4.3 převažují spíše negativní stránky a omezení.

Zcela zásadní nevýhodou zmiňovaného postupu je skutečnost, že funguje pouze pro krajinu generovanou algoritmem Fault Formation, a neumím si představit jeho zobecnění pro ostatní metody výpočtu výškových map. Abychom byli schopni navázat mapu na jejího souseda již při jejím vytváření, musíme být totiž schopni jednoznačně popsat proces utváření každé mapy. Právě Fault Formation algoritmus se z tohoto pohledu jeví jako vhodný kandidát a detailní popis tvorby mapy se v jeho případě v zásadě omezuje na souřadnice formujících zlomů v jednotlivých iteracích. Návrh tohoto algoritmu byl stěžejní částí předchozího ročníkového projektu a jeho základní myšlenky jsou uvedeny v následující podkapitole.

Na druhou stranu, tento postup nijak neovlivňuje samotné vykreslování trojrozměrného modelu z takovéto mapy vzniklého. Každá mapa popisuje svoji část povrchu a renderováním map s příslušným posunem vzniká dokonale hladká krajina. Tato skutečnost je, dle mého názoru, hlavním pozitivem zvoleného přístupu a ve svém důsledku, byl-li by uvedený způsob využit pro vytvoření „nekonečné“ krajiny, by znamenala značné zjednodušení enginu oproti verzi, která je nyní na světě. Nutno však podotknout, že navržený algoritmus je složitý a tudíž pomalý a pravděpodobně by v real-time generování krajiny neobstál.

Druhý přístup, který nahlíží na celou problematiku zcela jiným zrakem, se snaží zodpovědět otázku, zdali by bylo možné generovat výškové mapy absolutně nezávisle jednu na druhé a řešení jejich navazování ve výsledném kusu terénu řešit později. Ačkoliv se tato myšlenka zdá na první pohled nereálná, ukázalo se, že řešení existuje a za cenu vyšších paměťových nároků lze kýženého cíle dosáhnout.

Nyní pouze naznačím, že pro vytvoření části terénu potřebujeme celkem čtyři výškové mapy, které se překrývají a jejich průnikem vznikne právě výsledný kus krajiny s poloviční rozměry. Algoritmus je do detailu rozveden a vysvětlen v kapitole 4.3 této zprávy.

Hlavní výhodou této strategie je absolutní volnost při volbě metody vytváření povrchu. Není třeba nijak popisovat tvorbu krajiny tak, aby z popisu mohly čerpat mapy sousední. Zdrojové mapy lze různě filtrovat, pomocí funkcí třídy `TeHeightMap` měnit nadmořskou výšku a tím ovlivňovat ráz krajiny, lze dokonce kombinovat různé metody generování výškových map v jedné aplikaci.

Navíc, díky získané volnosti při tvorbě zdrojových map, můžeme využít základní verze algoritmů, které jsou dostatečně rychlé a osvědčily se při časově „kritických“ výpočtech základních stavebních kamenů „nekonečného“ zemského povrchu.

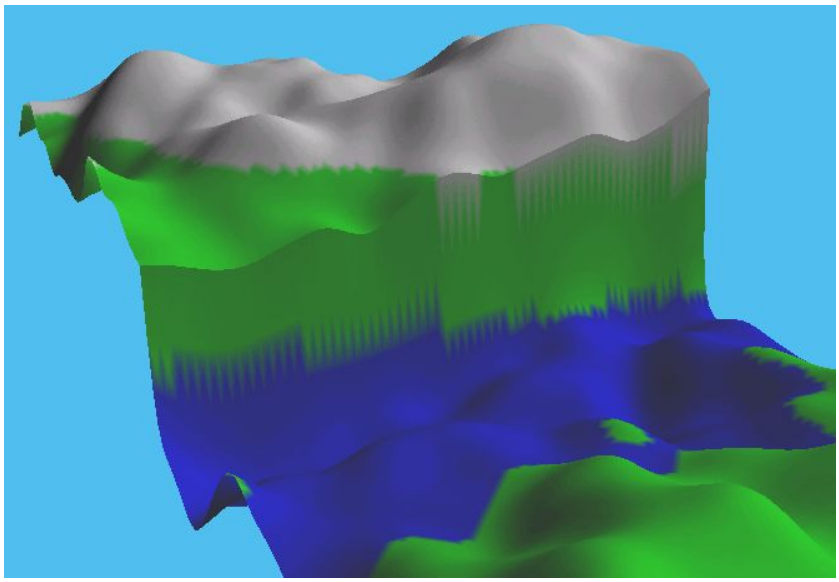
Naproti tomu se kvůli naznačenému postupu komplikuje řídící engine výsledné aplikace a hrozilo nebezpečí, že rychlost generování jednotlivých map bude ve finále degradována pomalostí složitého enginu udržujícího konzistenci v prostoru. Tomuto nebezpečí bylo nutno již při návrhu předcházet.

4.2 Navazování v okamžiku generování mapy

4.2.1 Průběh vývoje algoritmu

Odložit řešení návaznosti jednotlivých čtverců terénu na okamžik filtrování výškové mapy, byl první nápad. Bohužel se ukázalo, že k dosažení uspokojivých výsledků zdaleka nedostačuje.

Navržený algoritmus fungoval na principu uschovávání hodnot na hranách mapy pro pozdější využití mapou sousední. V okamžiku vyhlazování povrchu se tato informace použila a sjednotila výšku na společné hranici. Tím bylo zajištěno, že čtverce na sebe přesně navazovaly na hraniční linii. Metoda ale nijak nezaručovala podobný charakter terénu na obou stranách hranice. Velmi často se, například, stávalo, že na jedné straně hranice bylo pohoří, na druhé mořské dno. Výsledek byl, jak je patrno z obr.4.1, nepoužitelný.



Obrázek 4.1 - Navazování při filtraci (vnucování okraje).

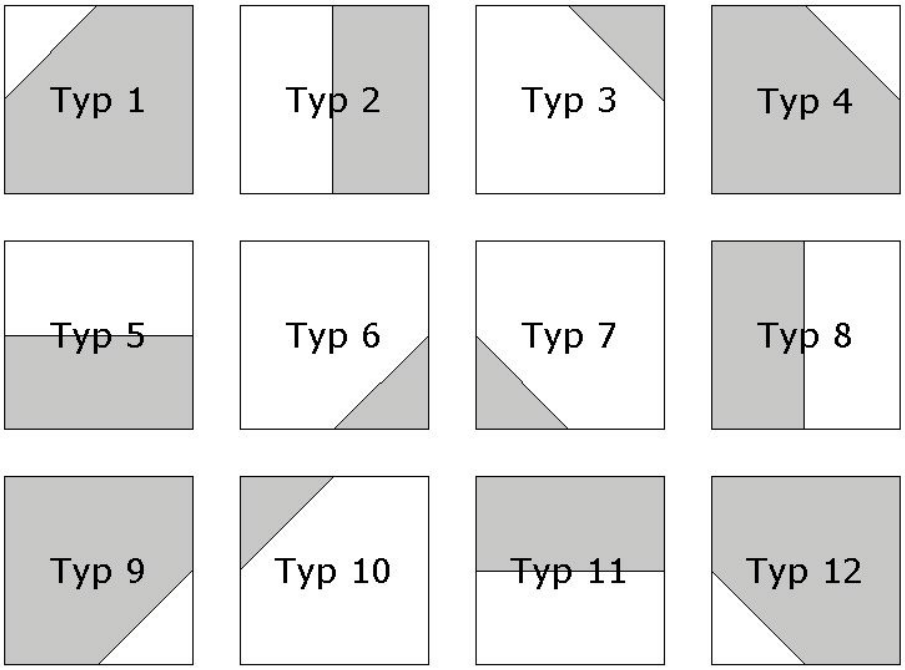
Jak prokázala předchozí metoda, bylo nutno zařídit stejný charakter povrchu na obou stranách společné hranice samostatných dílů terénu. K tomu mělo přispět zapamatování zlomů procházejících dotyčným okrajem mapy.

Toho využívá další verze algoritmu, které stačí zjistit, zda zlom prochází okrajem, o který se zajímáme, zapamatovat si souřadnici a číslo iterace, ve které ke němu došlo. Sousední mapa tyto informace využívá k tomu, aby zlom v dané iteraci plynule pokračoval i na ní.

Dále využijeme postupu popsaného výše (vnucení okraje), abychom zajistili dokonalé sesazení čtverců terénu k sobě.

Úskalí této metody tkví ve skutečnosti, že iteracím, ve kterých hranicí obou map neprochází žádný terénní zlom, je ponechána volnost. Úspěšnost metody tedy závisí na náhodě a nedá se předem odhadnout. Občas poskytla výsledky dobré, většinou byly ale podobné předchozí variantě. Bylo tedy nutné dál hledat metodu, která by popsala také iterace dosud nekontrolovatelné.

Při hledání takové metody jsem vyšel ze skutečnosti, že zlom lze na výškové mapě vygenerovat pouze dvanácti způsoby tak, jak ukazuje obr.4.2.



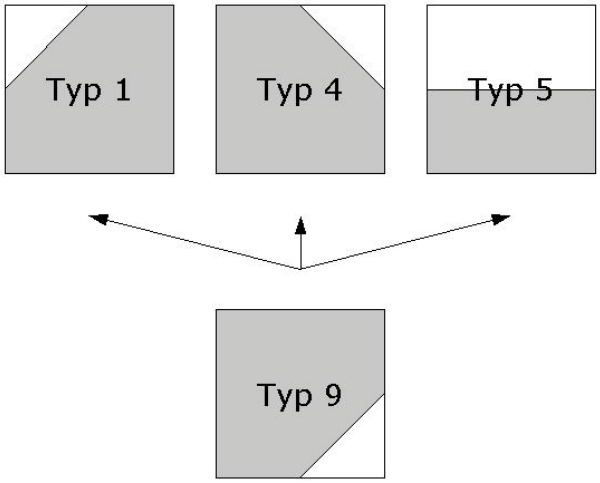
Obrázek 4.2 - 12 možných typů zlomu (šedá oblast vyvýšena).

Tato informace je výhodná, neboť dokáže postihnout chování terénu na rozhraní samostatně získávaných čtverců také pro iterace, ve kterých námi sledovaným okrajem neprochází žádný zlom.

Nejlépe se situace ozřejmí na konkrétním příkladu (viz obr.4.3 na následující straně).

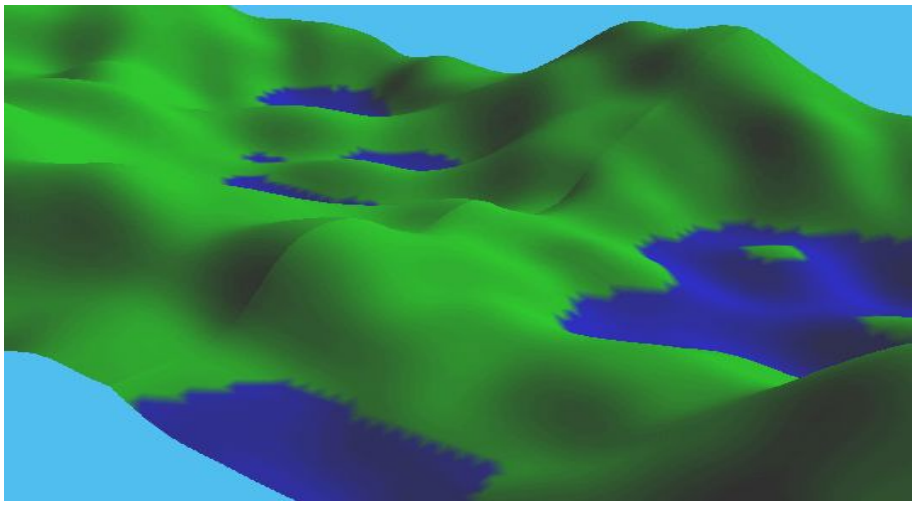
Předpokládejme, že máme vytvořenou mapu a nyní generujeme další, která k původní přiléhá shora. Dále předpokládejme, že pro k-tou iteraci byl v již hotové mapě použit zlom typu 9. V případě předchozí metody bychom neměli žádné omezení pro k-tou iteraci tvorby nové mapy.

Mohli bychom tedy zlom vytvořit zcela libovolně a to by mohlo vyústit ve značný výškový rozdíl na hranici obou map, který by byl vidět a působil by nepřirozeně a rušivě. Využijeme-li poznatku o typu zlomu, přijdeme na to, že zlom v nové mapě lze generovat třemi způsoby: jako typy 1, 4 a 5. Mezi nimi si již můžeme bezpečně zvolit náhodně. Tím je zajištěna konzistence v místě spojení obou map.



Obrázek 4.3 - Příklad volby typu zlomu.

Uvedeným postupem jsme schopni eliminovat problémy vyskytující se u předchozích dvou metod. Kvůli filtraci povrchu se ovšem mezi spojovanými kusy terénu vyskytovaly škvíry. Tento neduh lze snadno odstranit vnucením přesných hodnot hrany souseda nově vytvářené mapě. Na zobrazeném terénu se však stále v místě napojení objevovala rušivá (tmavší) čára. Problémem byly špatně vypočtené normály na hranicích obou map.



Obrázek 4.4 - Důsledek špatně vypočtených normál.

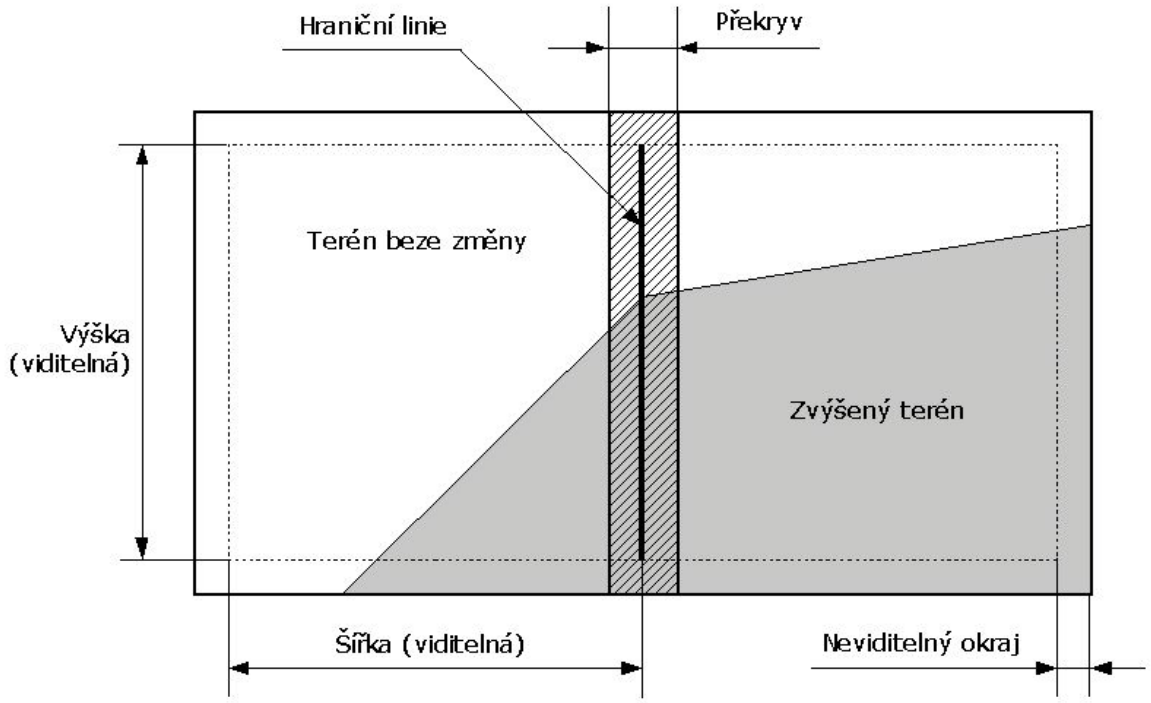
4.2.2 Mapy s překryvy - konečné řešení

Finální metoda navazování výškových map fungující, bohužel, pouze pro Fault Formation algoritmus je založena na dvou principech. Prvním je poznatek, že musíme být schopni co nejúspornějším způsobem popsat průběh každé iterace vnitřní smyčky algoritmu, neboť sousední mapa tyto informace nutně potřebuje pro vytvoření sebe sama. Druhým principem je částečné překrývání map kvůli potřebě vypočíst správně normály k povrchu i v hraničních bodech jednotlivých map.

Při řešení byly využity všechny postupy uvedené v předchozím textu.

Každou iteraci popíšeme třemi čísly – typem zlomu, a dvěma významnými souřadnicemi vystihujícími směr zlomu (zbylé 2 souřadnice jsou buď nulové nebo rovny šířce mapy a lze je z informace o typu zlomu odvodit).

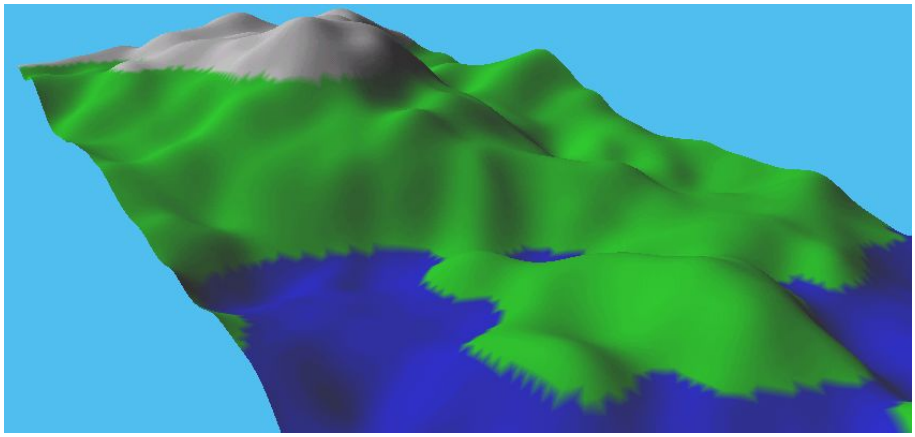
Potíže s normálami na okrajích map, které nás budou potkávat i v nové metodě, byly vyřešeny pomocí částečného překrývání sousedících čtverců. Celá potíž spočívá ve skutečnosti, že mapa, kterou právě generujeme, by pro výpočet správných normál na svém okraji potřebovala data původem od mapy sousední. Sousední mapa však nemusí být k dispozici. Proto každá mapa generuje „o kousek víc“, než později zobrazí. Začínáme tedy rozlišovat fyzickou a viditelnou velikost pokryté oblasti. Situace je naznačena v obrázku 4.5.



Obrázek 4.5 - Ilustrace překrývání výškových map.

Generujeme-li mapu, která již má existujícího souseda, musíme data vytvořená sousedem

respektovat a právě vytvářenou krajinu jimi korigovat. Hodnoty, které byly vypočteny aktuální mapou musí být v překrývající se části ignorovány. Případný engine vybudovaný nad takto fungujícími algoritmy by musel pouze udržovat informace o sousedských vztazích a zajišťovat postupné generování nových map ve vhodném pořadí.



Obrázek 4.6 - Místo spojení dvou čtverců terénu, správně navázáno.

4.3 Navazování nezávisle generovaných map

Pro další použití při tvorbě programu zobrazujícího „nekonečný“ terén byla ovšem zvolena následující strategie zajištění navazování výškových map.

Důvody k tomuto rozhodnutí jsem již naznačil výše, přesto připomenu zejména očekávanou rychlost, s jakou budou nové základní mapy generovány, jelikož nebude třeba využívat komplikovaných algoritmů nastíněných v předchozí kapitole, ale postačí základní verze jednotlivých generátorů. Dalším pozitivem je možnost veskrze libovolné kombinace metod generování a filtrování povrchu a tím nabytá schopnost, alespoň do určité míry, ovlivňovat a řídit výsledný charakter produkované krajiny.

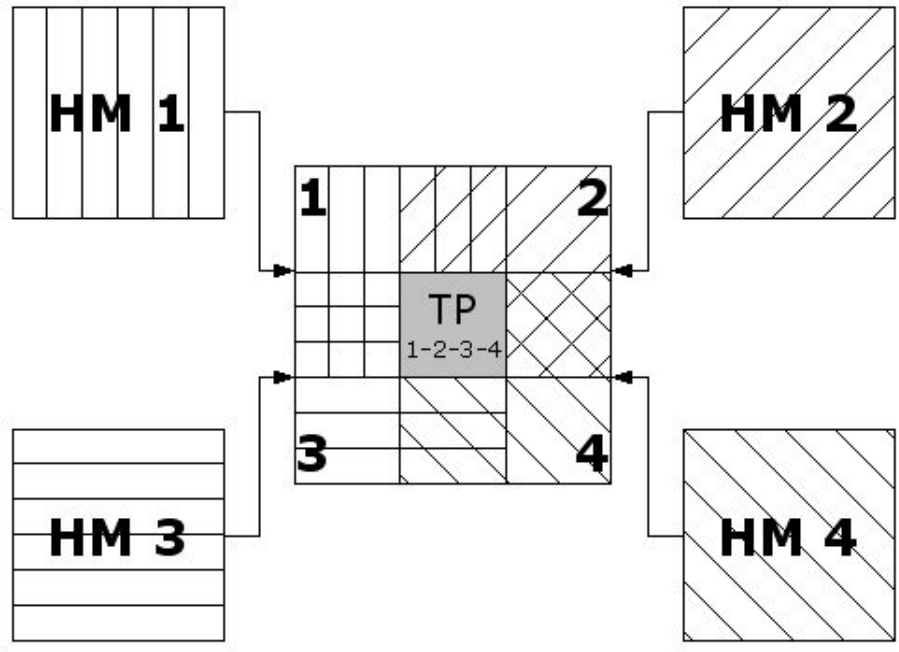
Daní za tyto nové možnosti jsou vyšší paměťové nároky a také složitější návrh a implementace enginu, který generování řídí.

4.3.1 Principy algoritmu

Jak bylo již uvedeno, pro vytvoření části terénu budeme nyní potřebovat celkem čtyři výškové mapy. Zde je právě původ oněch zvýšených paměťových nároků. Naše čtyři mapy budu v dalším textu označovat jako „základní“ či „zdrojové“ a mohou být získány jakoukoliv dostupnou

technikou bez omezení. Předchozí věta platí na čistě teoretické úrovni pohledu. V praxi se ovšem ukazuje, že velmi výhodné z hlediska složitosti implementace bude vyžadovat čtvercový tvar, kvůli použitému způsobu renderování musí být jeho rozměr navíc liché číslo. Troufnu si tvrdit, že v drtivé většině případů toto omezení nebude na závadu, naopak přispěje ke zpřehlednění celé situace.

Mějme tedy připravené čtyři nezávisle vytvořené výškové mapy. Princip metody spočívá v tom, že základní mapy na sebe nenavazují, nýbrž se překrývají přesně v polovině svého rozměru. Situace je neznačena na obrázku 4.7. Označme si rozměr mapy W (jako width, tj. šířka). První mapu umístíme do pomyslných souřadnic ve 2D prostoru tak, že její levý horní roh bude na souřadnici [0,0]. Druhá mapa má své místo na souřadnici [W/2,0], třetí na pozici [0,W/2] a konečně poslední na [W/2,W/2].



Obrázek 4.7 - Skládání terénu ze čtyř výškových map.

Na obrázku vidíme, že uprostřed vzniká oblast o rozměrech W/2, která je pokryta všemi mapami. Zároveň je patrné, že případné další základní mapy již na tuto oblast nebudou mít vliv, tudíž je připravena k zobrazení. Vlastní data vznikajícího plátu krajiny získáme jako součet hodnot na příslušném místě v základních mapách. Hodnoty ovšem před sečtením násobíme vahami, které vyjadřují vliv konkrétní základní mapy na výsledek.

V rozích výsledné oblasti je vždy 100% vliv té základní mapy, uprostřed které se roh nachází. Ve středu je rovnoměrný příspěvek od všech čtyř map (25% podíl). Na hranicích oblasti se uplatňují vždy dvě základní mapy. Pro ostatní body se váhy interpolují tak, aby se uplatnily všechny 4 základní mapy. Více k tématu získávání váhových koeficientů je uvedeno v následující podkapitole, ve které je podán neformální důkaz korektnosti uvedeného postupu.

Na závěr ještě krátký příklad. Pro „severní“ okraj znázorněné oblasti platí: V levém horním rohu 100% vliv mapy č.1, žádný příspěvek od č.2, následuje rovnoměrný pokles vlivu č.1 a vzrůst vlivu č.2 tak, že v polovině cesty od jednoho rohu k druhému se vlivy obou map rovnají 50%. Poté pokračuje oslabování vlivu č.1 tak, že v pravém horním rohu nastává absolutní dominance mapy č.2 a výška pro tento bod se tedy ze zdrojové mapy prostě překopíruje.

Pokud bychom chtěli nyní zobrazit další část krajiny, např. vpravo od stávajícího kousku, musíme vygenerovat 2 nové základní mapy, „umístit je správně do prostoru“ a obdobným postupem získat průnik nových map a stávajících map č.2 a 4. Představme si, že nyní počítáme hraniční linii mezi novým a starým čtvercem. Na této hranici jsou váhy příspěvků od nově vygenerovaných map nulové, takže musíme dojít k přesně stejným výsledkům, jako v případě čtverce předchozího. Tím je spolehlivě automaticky zajištěno přesné navázání nového kousku krajiny na kousek starý.

Navíc, díky pozvolnému přidávání významu novým mapám, nemůže se stát, že by se v krajině vyskytly náhlé změny jejího charakteru. V dřívějších pokusech se mnohdy stalo, že se vedle sebe vyskytly vysoké hory a mořské pobřeží. Staré metody sice dokázaly takto rozdílné povrchy „spojit“, ovšem vizuální dojem byl neuspokojivý a iluze reality prakticky žádná. Tato metoda zajišťuje, že i když se snažíme vygenerovat kus krajiny „z hor a moře“ současně, dojde ke „zprůměrování“ hodnot a pozvolnému, realisticky vyhlížejícímu přechodu od jednoho extrému ke druhému.

Engine „nekonečné“ krajiny se komplikuje jednak nutností udržovat konzistenci v prostoru základních map, které musí pohotově generovat, jednak udržováním informací o již vytvořených plátech krajiny ve vyšší vrstvě. Jak uvidíme dále, přibudou ještě komplikace způsobené normálami na hranicích vykreslovaných plátů terénu.

4.3.2 Matice vah

Nyní již víme, že kvůli navazování jednotlivých plátů krajiny musíme při sčítání násobit hodnoty obsažené v základních výškových mapách vahami, které vyjadřují podíl vlivu konkrétní mapy na výsledek.

Základní úvaha vychází ze skutečnosti, že každá zdrojová mapa ovlivňuje čtyři mapy výsledné. Jsou to přesně ty, které se v prostoru nacházejí na souřadnicích pokrytých touto mapou. Váhy, kterými ovlivňuje čtyři zmiňované výsledky si můžeme představit v matici o rozměrech shodných s rozměry vlastní základní mapy. Významnými hodnotami jsou jednička uprostřed matice (odpovídá 100% vlivu v rozích výsledků ovlivněných touto mapou), nuly po obvodu a hodnoty 0,25 uprostřed jednotlivých kvadrantů (to jsou místa, která jsou ve výsledku ovlivněná rovnoměrně všemi zdrojovými mapami).

Hodnoty v osách matice procházejících středem (středový kříž) jsou lineární interpolací od nuly k jedničce a zase zpět do nuly. Hodnoty v ostatních bodech matice vypočítáme vynásobením hodnot středového kříže na odpovídajících souřadnicích. Výsledky jsou středově souměrné, takže

je stačí počítat v jednom kvadrantu a rozmisťovat paralelně do všech čtyř.

0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,00	0,06	0,13	0,19	0,25	0,19	0,13	0,06	0,00
0,00	0,13	0,25	0,38	0,50	0,38	0,25	0,13	0,00
0,00	0,19	0,38	0,56	0,75	0,56	0,38	0,19	0,00
0,00	0,25	0,50	0,75	1,00	0,75	0,50	0,25	0,00
0,00	0,19	0,38	0,56	0,75	0,56	0,38	0,19	0,00
0,00	0,13	0,25	0,38	0,50	0,38	0,25	0,13	0,00
0,00	0,06	0,13	0,19	0,25	0,19	0,13	0,06	0,00
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Obrázek 4.8 - Příklad matice vah 9x9, zvýrazněný první kvadrant a významné hodnoty.

Máme-li takto vytvořenu matici vah, můžeme jí ihned po vygenerování násobit hodnoty základní výškové mapy a při skládání výsledného plátu krajiny jednoduše sčítat hodnoty na překrývajících se souřadnicích.

Alternativně lze koeficienty získávat při každém výpočtu výsledného čtverce krajiny, násobit jimi hodnoty zdrojových map a ty sčítat. Tento postup je nyní implementován a výpočet koeficientů neovlivňuje nijak zásadním způsobem rychlost tvorby nových map. Navíc je tato alternativa vhodnější z hlediska budoucího rozšíření o techniky LOD [=Level Of Detail], které mohou vyžadovat původní data. Těmito záležitostmi jsem se nezabýval, každopádně jsou zmíněné postupy ekvivalentní a vedou ke stejným výsledkům.

Zbývá ukázat, že takto vypočtená matice je korektní a zaručuje, že výsledná krajina je rovnoměrně ovlivněna všemi zainteresovanými zdrojovými mapami. Důkaz je velmi jednoduchý. Na obrázku 4.8 je matice 9x9 se zvýrazněnými významnými hodnotami, které byly popsány výše v textu a barevně odlišeným prvním kvadrantem. Hranice kvadrantů jsou vyznačeny tlustší čarou, hodnoty ve středovém kříži jsou sdílené. Představme si, že počítáme výsledný plát krajiny ze čtyř základních výškových map. Takový plát spočítáme jako součet hodnot na překrývajících se plochách map, které byly předem vynásobeny stejnou maticí vah (nebo alternativní cestou uvedenou výše). Pro každý plát se uplatní část zdrojové mapy násobená jiným kvadrantem matice. Překryjeme-li navzájem hodnoty jednotlivých kvadrantů a sečteme je, dostaneme matici (v tomto případě 5x5, obecně W/2 zaokrouhleno nahoru), jejímiž prvky jsou samé jedničky.

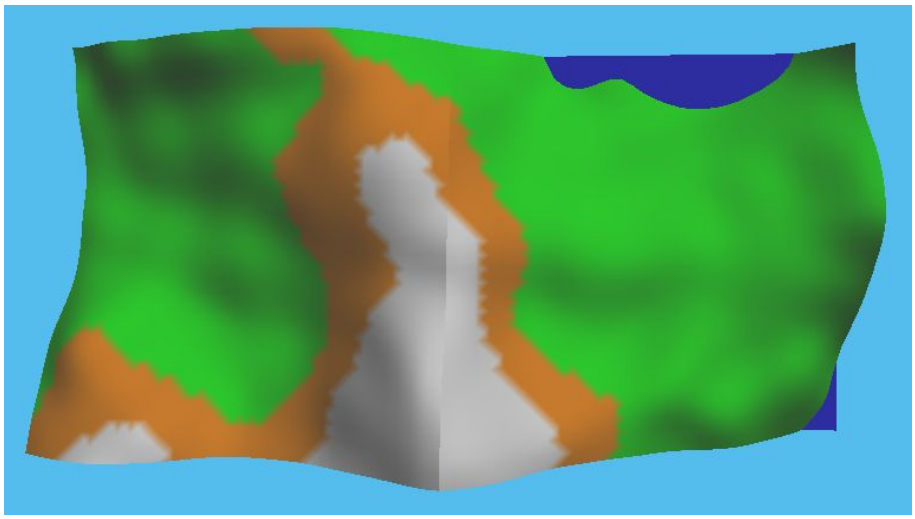
Tím je dokázána spravedlivost vůči základním mapám co do možnosti ovlivnit výsledný ráz vytvářené krajiny.

4.3.3 Normály na okrajích čtverců

V předchozím textu jsem popsal způsob, jakým lze z naprosto nezávisle generovaných výškových map vytvářet navazující pláty krajiny. Chceme-li získanou krajinu vykreslit na obrazovku počítače, je třeba vytvořit trojrozměrný model složený z trojúhelníků se správně vypočtenými normálami v každém vrcholu (vertexu). Právě s výpočtem normál souvisí poslední úskalí navržené metody.

Není totiž dost dobře možné vypočítat normály pro body nacházející se na okrajích plátu krajiny, který chceme renderovat. Pro výpočet těchto normál bychom totiž potřebovali údaje z mapy sousední, která nemusí být k dispozici. Opakuje se tedy stejná situace, jako v případě předchozí metody. Důsledkem špatně vypočtených normál na hranicích jednotlivých čtverců jsou viditelné čáry přes terén právě podél hranice čtverců, které působí velmi rušivě.

Problém byl v předchozím případě řešen zavedením překrývání. Každá mapa vygenerovala více, než později vykreslila a „nadbytečné“ údaje použila právě k výpočtu normál. Při použití metody skládání ovšem máme k dispozici „skoro dobré“ údaje i za hranicí aktuálního čtverce v podobě dat ze dvou map v nižší vrstvě. Myslel jsem, že se hodnoty nebudou od správných příliš lišit a k výpočtu normál budou postačovat. Výsledek, jak ukazuje obrázek 4.9, byl ovšem neuspokojivý.



Obrázek 4.9 - Důsledek špatně vypočtených normál.

Bylo tudíž třeba najít jiný způsob řešení. Zavedení dalšího překrývání podobným způsobem, jako v předchozím případě by bylo nešikovné, jednak z hlediska přehlednosti, jednak kvůli

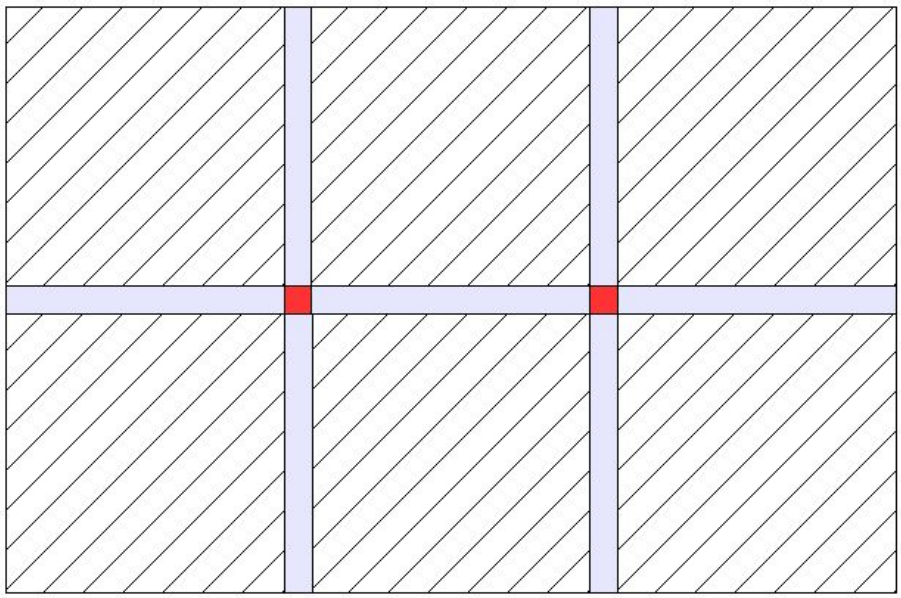
nutnosti generování alespoň části sousedů aktuálního čtverce, které by se jistě podepsalo na rychlosti.

Řešení, které jsem zvolil, se možná na první pohled zdá zbytečně složité, ovšem poskytuje dobré výsledky a již při jeho návrhu jsem předpokládal existenci grafického enginu, který se o vše potřebné automaticky postará sám.

Každý čtverec terénu zná totiž dobře normály ve všech svých bodech, kromě právě bodů hraničních. Nechme tedy čtverec renderovat to, co zná dobře (tj. čtverec o rozměru W/2-2) a kritickou oblast spoje vykreslujme až v okamžiku, kdy máme vypočteného souseda a chceme ho též zobrazit.

Celá krajina se tedy bude skládat z velkých „hlavních“ čtverců, mezi kterými budou vertikální a horizontální spojky (o šířce dvou pruhů trojúhelníků). Mezi každou čtveřicí hlavních čtverců bude navíc malinký čtvereček, který bude vyplňovat vzniklou díru. Tyto spojky budeme dále nazývat „švy“ (anglicky seam).

Schema stavby krajiny z vyjmenovaných komponent je znázorněno na obrázku 4.10, kde jsou jednotlivé součásti barevně rozlišeny. Jedná se zde pouze o zobrazení principu a velikosti stavebních bloků terénu nejsou v měřítku, šířka švů je na něm přehnaně zvýrazněna.



Obrázek 4.10 - Schema stavby krajiny.

5. Engine pseudonekonečné krajiny

5.1 Schema enginu

Grafický engine tvoří jakýsi mozek aplikace. Takový mozek je zodpovědný za hladký běh programu a kvalita jeho návrhu předurčuje kvalitu celého výsledného díla.

Navržený engine si jako hlavní cíl klade schopnost dynamicky generovat nové pláty krajiny, jejichž skládáním pomocí techniky popsané v předchozí kapitole (švy) navodí iluzi absolutní volnosti uživatele pohybujícího se ve virtuálním prostoru.

Z předchozího odstavce plyne několik úkolů, které musíme engine naučit řešit. Engine musí udržovat přehled o již vygenerovaných mapách a to jak o tzv. mapách základních, tak o již hotových plátech krajiny vzniklých složením čtyř základních map. Dále musí přesně znát pozici uživatele ve scéně a na jeho pohyb reagovat. Prochází-li uživatel po již „známém“ terénu, musíme data nalézt v paměti a vykreslit. V opačném případě musíme ještě před tím zajistit jejich vygenerování. To celé by mělo probíhat „paralelně“ se samotným renderováním v režii grafické knihovny.

Ve splnění tohoto požadavku osobně spatřuji hlavní přínos navržených algoritmů. Pomocí koncepce tzv. generátorů (viz kapitola 5.3) jsme schopni distribuovat procesorový čas a dosáhnout kýžené iluze nekonečnosti bez nežádoucích efektů v podobě trhání obrazu. I když samotný model zemského povrchu je nyní vcelku prostý (není implementováno např. jeho texturování), díky návrhu kooperujících generátorů je do budoucna otevřena cesta k jeho podstatnému rozšíření.

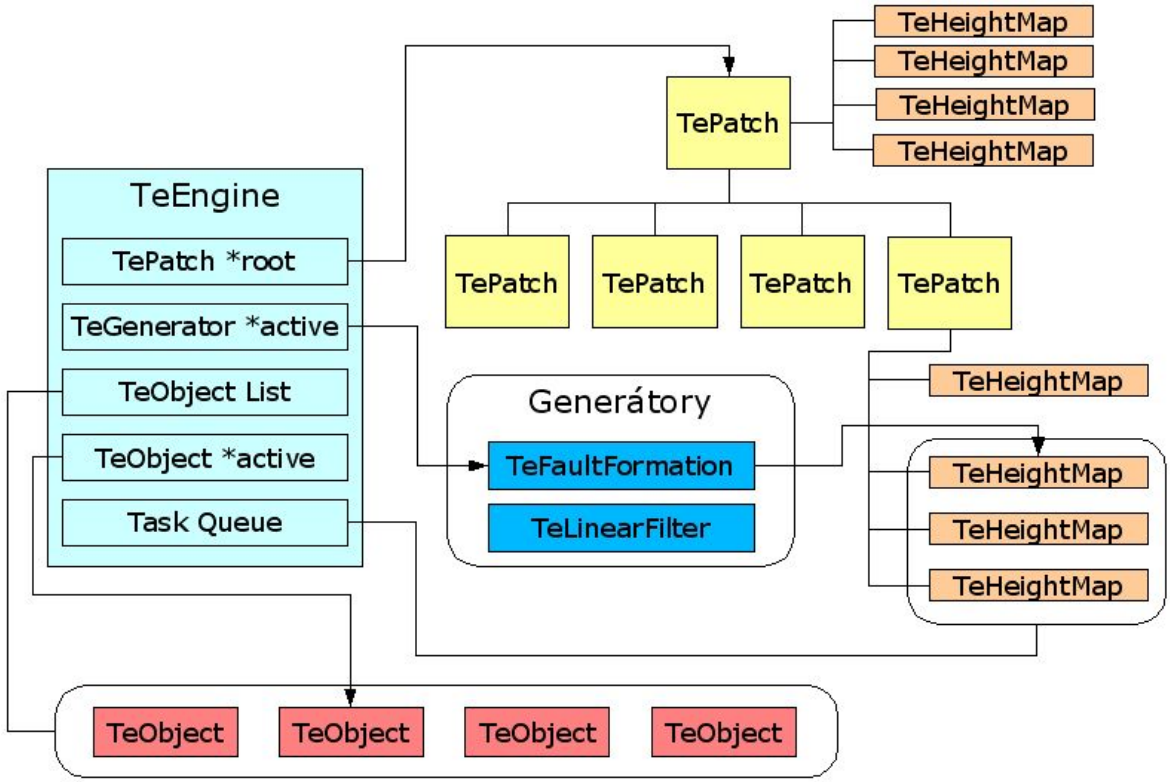
Informace o výškových mapách udržuje stromová struktura a její prohledávání se i pro rozsáhlou databázi map osvědčilo jako dostatečně rychlé.

Engine udržuje seznam objektů, které se ve virtuálním prostoru vyskytují. Každý z těchto objektů způsobuje vyvolávání požadavků na generátory terénu, jeden z nich má ale výsadní postavení. Je v něm totiž umístěna kamera a představuje tedy uživatele.

Na obrázku 5.1, který je z prostorových důvodů, bohužel, umístěn až na další straně jsou znázorněny vazby mezi jednotlivými součástmi enginu.

Světle modrou barvu má vlastní objekt reprezentující engine. Žlutě jsou reprezentovány objekty třídy `TePatch`, které představují hotové pláty krajiny. Oranžovou barvou jsou označeny základní výškové mapy potřebné k vytvoření plátů. Jedná se o objekty již popsané třídy `TeHeightMap`. Tmavě modrou barvu mají generátory operující nad daty výškových map a konečně červené jsou objekty umístěné do virtuálního prostoru.

Na obrázku vidíme, že engine má k dispozici ukazatel na kořen stromové struktury plátů krajiny, ukazatel na právě aktivní generátor (to je ten, který aktuálně dostává příděly procesorového času), seznam objektů a ukazatel na aktivní objekt (objekt, ve kterém „sedí“ uživatel). Dosud nebyla řeč o frontě úloh, do které jsou umisťovány rozpracované výškové mapy a generátory tak vědí, jakou práci vykonávat.



Obrázek 5.1 - Schema enginu.

Všechny zmíněné komponenty jsou detailněji popsány v následujících podkapitolách.

5.2 Základní schema aplikace

V minulosti bylo běžné, že běh aplikace byl řízen přímo kódem programu. Byla přesně definována místa, kde program čekal na vstupy od uživatele a naopak, prováděl-li program výpočet, uživatel neměl možnost do procesu zasahovat. Na druhé straně každá moderní aplikace s grafickým uživatelským rozhraním využívá principu událostmi řízeného běhu.

Základním stavebním kamenem každé takové aplikace je nekonečný cyklus, který neustále testuje, zdali nebyla správcem oken aplikaci zaslána nějaká událost, na kterou by bylo třeba zareagovat. Existenci takového nekonečného cyklu využívá beze zbytku i implementovaný engine. I když je smyčka zpráv zapouzdřena uvnitř grafické knihovny, nejde o nic jiného.

Hlavní a zcela zásadní myšlenka spočívá v tom, že v každém cyklu této smyčky dáme enginu šanci provést nějakou užitečnou operaci. Takovou operací jsou např. změny pozic objektů v prostoru, reakce na uživatelskou interakci pomocí klávesnice či myši, kontrola, zdali je

renderováno vše, co je vidět kamerou uživatele, nějaké rozhodnutí umělé inteligence, detekce kolizí atp. Veledůležitou operací je přidělení určitého časového kvanta generátoru, který pracuje na vytvoření nějaké výškové mapy.

Celou smyčku pak můžeme pomocí pseudokódu zapsat takto:

```
for (;;) {
    dt = compute_elapsed_time();
    if (unhandled_event) active_object->handle_events();
    for each object {
        object->update(dt);
        object->detect_collisions();
    }
    check_terrain_under(each object);
    active_generator->tick();
    render_scene();
}
```

Po každém průchodu touto smyčkou dojde k vykreslení jednoho snímku na obrazovku a celý proces se opakuje. Jsou-li všechny operace uvnitř smyčky provedeny „dostatečně“ rychle, pozorovateli se jeví případný pohyb jako plynulý. Obecně se za minimální postačující frekvenci udává 25 snímků za sekundu (zkratka FPS = Frames Per Second).

Vše tedy záleží na tom, jak rychle lze operace, pro které se engine rozhodne v jednom cyklu, provést. V našem případě, kdy klademe důraz na generování nových výškových map, ovšem nelze z důvodu časové složitosti takového úkolu provést celou operaci najednou. I když díky výše diskutovaným technikám používáme základní verze algoritmů, hrubý hardwarový výkon je pro takový úkol stále nedostatečný. Zde přicházejí ke slovu vyvinuté generátory, které pomocí fronty rozpracovaných úloh proces vytvoření výškové mapy rozdělí na kratší úseky, které lze již bez problémů uskutečnit naráz.

5.3 Generátory, fronta úloh

Již v předchozím textu bylo naznačeno základní poslání třídy generátorů. Slouží k rozdělení potenciálně dlouhých a složitých operací na úseky proveditelné v dostatečně krátkém čase.

Každý generátor je specializovaný na určitý úkol, pro jehož splnění vyžaduje určité množství běhů své výkonné rutiny. Úkolem enginu je sdělit aktivnímu generátoru nad jakými daty má pracovat a volat opakovaně tuto rutinu. Tato funkce je implementována s návratovým typem `bool`, pomocí kterého sděluje enginu, zda práci dokončila, nebo je ji příště třeba volat znovu.

Engine udržuje frontu rozpracovaných úloh. Přímo v enginu je v tuto chvíli zabudován řídící mechanismus, který během několika volání příslušné funkce zajistí splnění úlohy, která prozatím

vždy spočívá ve vytvoření plátu krajiny. Jednotlivé etapy plnění této úlohy vyžadují nalezení, případně vygenerování čtyř výškových map, výpočet výsledné mapy plátu a její převod na 3D model požadovaný grafickou knihovnou.

Škála úloh, které mohou být pomocí generátorů řešeny, je velmi široká. Dokáži si představit generátor, který bude data nahrávat po částech z disku, jiný může počítat procedurální textury povrchu, další vytvoří reálně vypadající vodní hladinu, třetí do krajiny umístí řeky, silnice, budovy, stromy, atp. Zde se otevírá prostor pro pokračování projektu, neboť všechna uvedená témata byla či jsou na naší fakultě aktuálně řešena a rozšíření enginu o jejich schopnosti je do budoucna jistě lákavé.

Nyní hotové generátory pracují s daty výškové mapy a představují dva dříve popsané algoritmy. Jsou to Fault Formation algoritmus pro generování hodnot výškových map a lineární filtr pro jejich vyhlazení. Generování hodnot je nastaveno tak, že v každém volání funkce vytvoří desetinu požadovaných zlomů, filtrování provede za jeden běh jeden kompletní průchod polem po všech řádcích a sloupcích v obou směrech.

Je nutné přiznat, že takto pevně nastavené hodnoty jsou nešikovné. Neberou totiž v úvahu skutečný výkon stroje, na kterém engine běží. Vše je experimentálně nastaveno tak, aby program běžel plynule na počítači, na kterém byl vyvíjen (AMD AthlonXP 1600+ pracující na frekvenci 1,4 GHz, grafická karta nVidia GeForce2 MX 400, 384 MB RAM). Bylo by lepší, aby generátory nějakým způsobem dynamicky měnily své nastavení a přizpůsobovaly se možnostem hardwaru. Bylo by například možné určovat čas, který smí generátor zabrat tak, aby bylo dosaženo přednastavené hodnoty FPS. To je další výzva pro případný navazující výzkum.

5.4 Databáze výškových map

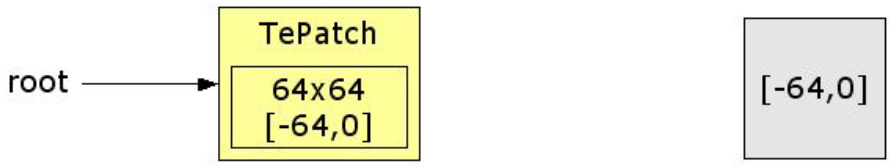
Veškeré informace nutné pro udržení konzistence terénu v prostoru jsou udržovány pomocí třídy `TePatch`. Ta představuje hotový plát krajiny přichystaný k renderování a jednotlivé její instance se propojují a vytvářejí stromovou strukturu.

Engine zná ukazatel na kořen celé hierarchie. Každý objekt této třídy nese informaci o ploše, kterou pokrývá. Neodpovídá-li požadovaná plocha přesně jeho vlastní, avšak je její podmnožinou, poskytne ukazatel na jednoho ze svých čtyř následníků (detailnějších). O tom, který ze čtyř možných kandidátů je použit rozhoduje relativní směr k požadované oblasti. Není-li požadovaný uzel ve stromu nalezen, je provedena některá z akcí v závislosti na zvolené strategii. Buď je vytvořen uzel nový (nový kořen, pokrývající větší plochu a uzel přesně odpovídající požadavkům), nebo je vrácen uzel obsahující menší úroveň detailů. Poslední variantou je navrácení nulového ukazatele na znamení, že se uzel ve stromu nenachází.

Myslím, že princip půjde lépe vysvětlit na příkladu. Budeme uvažovat situaci, kdy požadujeme generování plátů krajiny uspořádaných do čtverce 3x3 se středem v bodě [-64,0]. Tento požadavek vznikne automaticky, umístíme-li na souřadnici [-64,0] (nebo do prostoru

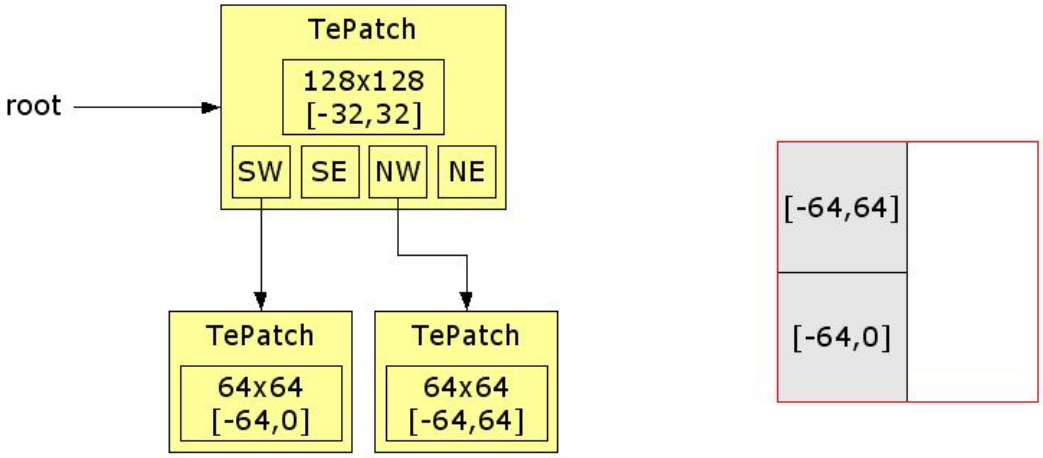
pokrytého tímto plátem) aktivní objekt. Předpokládejme, že implicitní rozměr jednoho plátu krajiny je 64x64 jednotek. Požadovaných devět plátů bude generováno postupně, několik počátečních kroků znázorněme schematicky.

Nejdříve se generuje plát uprostřed požadované plochy (tzn. pod pozicí daného objektu).



Obrázek 5.2 - Příklad výstavby databázového stromu (krok 1).

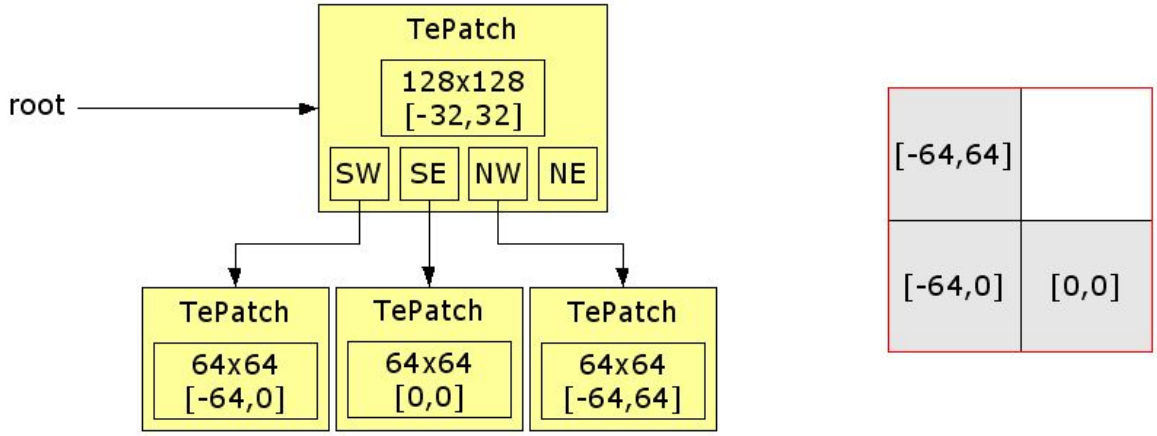
Na obrázku 5.2 je znázorněna situace po provedení prvního kroku algoritmu. Strom nyní obsahuje jediný uzel, který je jeho kořenem. Je v něm uložen plát krajiny o rozměrech 64x64 a se středem v bodě [-64,0]. V pravé části bude postupně budována mapa pokrytí prostoru. Nyní bude požadován čtverec „severně“ od aktuálního (souřadnice jeho středu je [-64,64]).



Obrázek 5.3 - Příklad výstavby databázového stromu (krok 2).

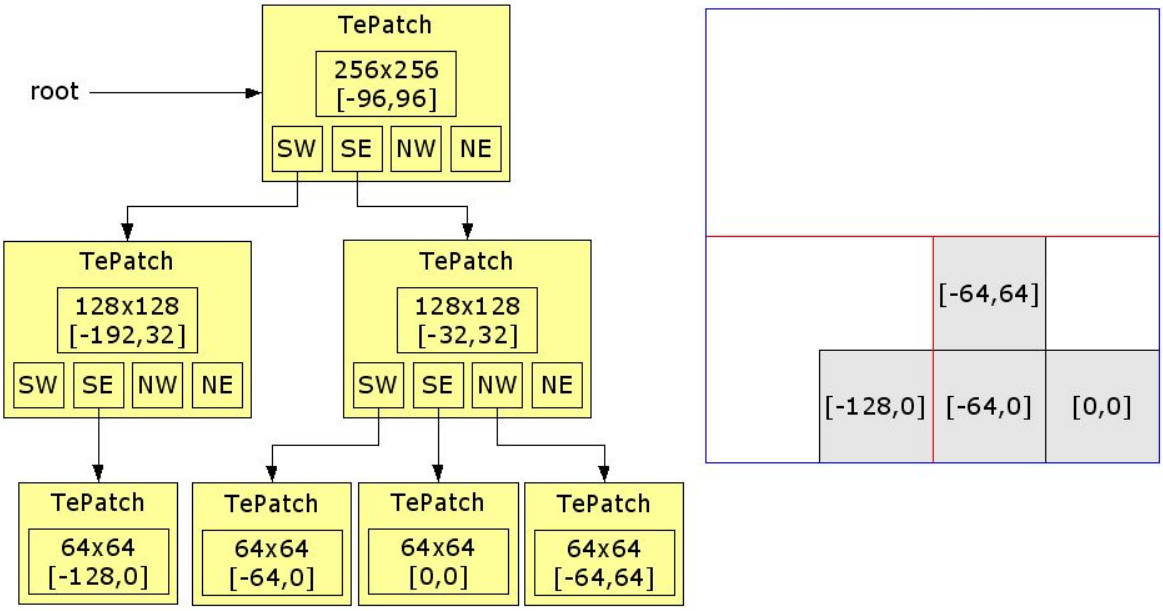
Algoritmus zkontroluje údaje v kořeni stromu a zjistí, že požadovaný čtverec není v jeho kompetenci. Kromě samotného uzlu s potřebným plátem vytvoří tedy navíc uzel, který se stane novým kořenem stromu. Tento uzel pokrývá dvojnásobně velké území a na mapě vpravo je ohraničen červenou čarou. Původní uzel a uzel s novým kouskem terénu budou dostupní pomocí ukazatelů (ty jsou označeny anglickými zkratkami světových stran).

V dalším kroku budeme požadovat terén se středem o souřadnicích [0,0]. Nastane tedy situace, kdy požadovaný úsek krajiny je pokryt současným kořenem stromu. Program se pokusí nalézt uzel na příslušném ukazateli, ten je však nulový. Provedenou akcí bude tedy vytvoření požadovaného uzlu a jeho navázání na správný ukazatel. Situaci znázorňuje obrázek 5.4.



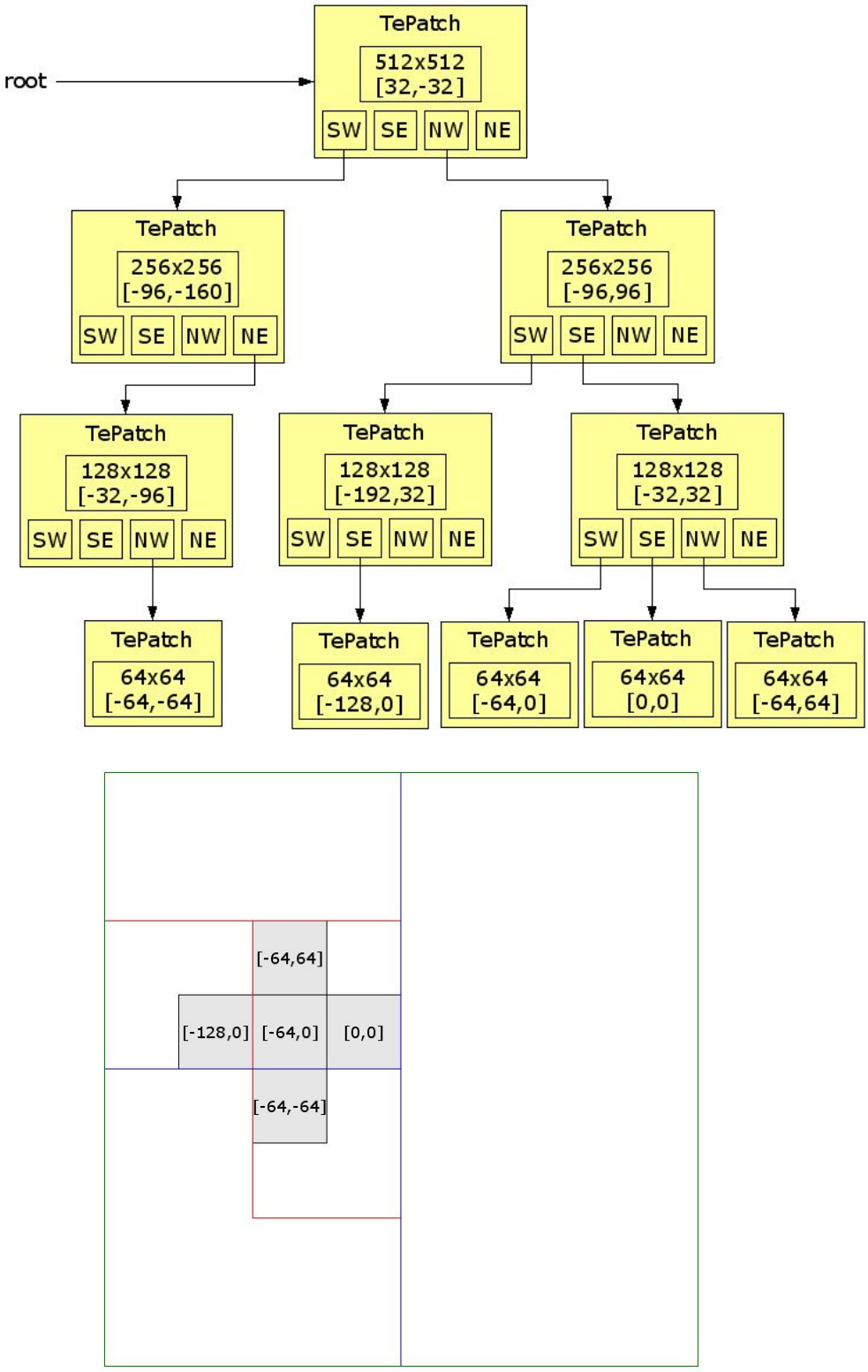
Obrázek 5.4 - Příklad výstavby databázového stromu (krok 3).

Následující kroky probíhají podobným způsobem. Požadujeme-li krajinu na souřadnici [-128,0] dojde mechanismus opět k závěru, že požadovaný uzel se ve stromu nenachází. Vytvoří tedy nový kořen stromu, další uzel na úrovni uzlu předchozího a samotný list s užitečnými daty. Plocha pokrytá novým kořenem je nyní ohraničena modře.



Obrázek 5.5 - Příklad výstavby databázového stromu (krok 4).

Posledním uvažovaným krokem tohoto příkladu bude plát na souřadnicích [-64,-64]. Ten opět není pokryt aktuálním kořenem stromu. Je tudíž znovu vytvořen kořen nový, pokrývající dvojnásobnou plochu, všechny uzly na úrovních srovnatelných s uzly již existujícími a vlastní list udržující požadované informace.



Obrázek 5.6 - Příklad výstavby databázového stromu (krok 5).

Na uvedeném příkladu vidíme, že veškerá užitečná data jsou uložena v listech stromu, vnitřní uzly slouží k jejich vyhledávání. Každý list dále obsahuje ukazatele na čtyři výškové mapy, ze kterých byl výsledný plát vytvořen, ukazatele na svých osm sousedů v prostoru (existují-li) a ukazatele na osm švů, kterými je obklopen. Tyto údaje jsou využívány při sdílení dat mezi jednotlivými pláty tak, že veškeré informace jsou fyzicky v paměti pouze jednou.

Navíc obsahuje každý plát uložený 3D model, který je v případě potřeby velmi rychle k dispozici pro vykreslení. Není tudíž třeba takový model vytvářet pokaždé, když se uživatel pohybuje po již známé krajině. To vše samozřejmě za cenu dalších paměťových nároků.

Na tomto místě je vhodné zmínit se o skutečnosti, že jsem neřešil situaci, ke které by došlo v případě kompletního zaplnění dostupné paměti. V současné době stále ještě převládající 32bitové architektury umožňují sice adresovat až 4 GB paměťového prostoru, ale po zaplnění fyzické paměti operační systém začíná odkládat data na disk, což se projeví celkovým snížením výkonu.

Tuto situaci by bylo možné řešit lépe ve vlastní režii. Bylo bylo možné vystavět generátor (i když v tomto významu je označení generátor zavádějící, chci tím pouze vyjádřit koncepci „rozsekání“ úkolu), který by dlouho nepoužitá data odstraňoval. V první fázi by stačilo zbavit se uloženého 3D modelu a ponechat data výškových map. Dále by jistě šlo data postupně ukládat na disk, případně sáhnout k nějaké metodě komprese přímo v paměti. Nejdůraznějším řešením by bylo odstranění dat samotných zdrojových map s tím, že by byly ponechány informace o parametrech použitých generátorů a použitém semínku pro generátor náhodných čísel. Z těchto informací by totiž šlo celé mapy vygenerovat znovu.

Skutečnost, že v každém okamžiku máme přístup k „surovým“ datům výškových map bude do budoucna jistě výhodná i pro metody LOD [=Level Of Detail]. Pro tyto metody obecně platí, že se snaží minimalizovat počet ve scéně použitých trojúhelníků. Jedny neustále přepočítávají trojúhelníkovou sít tvořící model viditelného terénu (např. ROAM), jiné vykreslují vzdálenější oblasti s menším množstvím detailů (Chunked-LOD). Zejména Chunked-LOD metoda má k navrženému způsobu databáze plátů krajiny velmi blízko a projekt jistě půjde o techniky LOD obohatit.

Detailněji jsem se však ve své práci problematikou LOD nezabýval.

5.5 Objekty, zpracování událostí

Třída `TeObject` představuje abstrakci pro všechny předměty, které lze umístit do scény. Definuje rozhraní společné všem takovým předmětům, specializované objekty mohou být reprezentovány třídami odvozenými. Engine udržuje seznam vložených objektů a v každém průchodu hlavním cyklem programu dává každému objektu šanci změnit svůj stav. Takovou změnou je většinou úprava polohy v prostoru dle nastaveného vektoru rychlosti. V budoucnu by se v tomto okamžiku měla ke slovu dostat umělá inteligence, která by řídila chování objektů nekontrolovaných uživatelem.

Objekt kontrolovaný uživatelem má výsadní postavení. Je označen jako aktivní a jeho hlavními povinnostmi jsou ovládání kamery, vykreslování HUD [=Heads-Up Display] a zpracování událostí od uživatele.

Pozice kamery, pomocí níž pozorujeme virtuální scénu, je svázána s pozicí aktivního objektu. Definujeme ji vzdáleností od něho. Umístíme-li kameru „dovnitř“ objektu, vytvoříme iluzi uživatele sedícího např. v kokpitu letounu (tzv. 1st person pohled – odpovídá pohledu očima našeho avatara). Umístíme-li kameru „mimo“ objekt, bude se jednat o tzv. 3rd person pohled na scénu, kdy vidíme vlastní postavu jakoby očima někoho dalšího.

Heads-Up Display (HUD) slouží k vypisování různých informací pro uživatele (aktuální pozice v prostoru, vektor rychlosti, nápověda ovládání apod.). Je realizován pomocí prvků vykreslovaných odlišnou kamerou až po renderování vlastní geometrie scény. Tím je zajištěna jejich viditelnost.

V současné době je odpovědnost za aktualizaci tohoto displeje ponechána na aktivním objektu, i když spousta potenciálně užitečných věcí je na objektu nezávislá a měla by je tudíž mít na starosti spíše nějaká další komponenta enginu. Až budoucnost ukáže, nakolik se jedná o rozumné řešení. Hlavním faktorem pro tuto volbu byla touha, aby uživatel podle vzhledu HUD poznal, v jakém objektu se nachází. Použijeme-li příkladu letadel, zajistíme tak, že každý typ letounu bude mít vlastní kokpit, což jistě přispěje ke zkvalitnění celkového dojmu a posílí realističnost aplikace. Na druhé straně, i toto by šlo jistě vyřešit pomocí specializované komponenty enginu, jak již bylo naznačeno. V tomto případě se tedy jedná o stále otevřené téma a pro potenciální pokračování projektu se bude třeba rozhodnout pro jednu z uvedených variant. Osobně se stále více přikláním k variantě řízení displeje přímo enginem, i když ji již z časových důvodů nemám šanci zrealizovat.

Naopak zodpovědnost za zpracování uživatelských vstupů s aktivním objektem přímo souvisí. Každý typ objektu může reagovat na stejné vstupy různě. Vyjdeme-li z obvyklých konvencí pro ovládání různých objektů v počítačových hrách vidíme, že stisk kurzorové šipky „nahoru“ může znamenat různé věci. V případě automobilu je ekvivalentní se sešlápnutím plynového pedálu, v případě letadla znamená natočení výškového kormidla. Reakce na povely od uživatele musí mít tedy do budoucna souvislost s fyzikálním modelem objektu. Zde je vidět výhoda jejich implementace pro každou třídu objektů zvlášť.

Objekty mají ještě jednu významnou zodpovědnost – pomocí nich definujeme modely reprezentující jednotlivé předměty. Rozeznáváme modely dvou typů. Jednak modely, které se budou účastnit procesu renderování scény a budou tedy tvořit vizuální podobu daného objektu, jednak modely určené k výpočtům detekujícím kolize ve scéně. Modely pro detekci kolizí jsou kvůli složitosti výpočtů mnohem jednodušší, než propracované modely určené k vykreslování. Kolize mohou nastat jak mezi jednotlivými objekty navzájem, tak mezi objekty a terénem. Detekce kolizí není v současnosti implementována.

Koncepci modelů jsem převzal z projektu SpaceGame. Všem autorům, podílejícím se na tomto projektu, patří touto cestou moje poděkování.

Podle polohy objektů ve scéně engine vykresluje viditelné pláty terénu. Více k tomuto tématu je uvedeno v kapitole 6.2, neboť navržené algoritmy využívají přímo možností

poskytovaných použitou grafickou knihovnou.

5.6 Možnosti využití enginu

V kapitole 2 jsem se pokoušel zasadit tuto práci do kontextu jiných projektů, které jsou ve světě komerčně využívány. I když jsem se přihlásil k větvi vývoje směřující k virtuální zábavě, v následujících odstavcích chci demonstrovat možnosti budoucího využití enginu i pro zbylé oblasti. Nejprve ale zmíním dvě varianty nasazení v počítačových hrách.

První variantou je využití enginu pro prosté získání modelu rozsáhlé, avšak ohraničené virtuální krajiny. Tato varianta je již nyní kompletně připravena k použití, neboť vše, co vyžaduje je implementováno a odladěno. Hlavní využití zde bude mít databáze výškových map a mechanismus tvorby finálních plátů krajiny. Potřebujeme-li pouze vygenerovat rozsáhlý terén, budeme postupně požadovat vytvoření příslušného počtu objektů třídy `TePatch`. Již manuálně potom budeme data těchto objektů a švy mezi nimi renderovat.

Dokonalejší variantou, pro kterou je vyvíjený systém primárně určen je udržování scény dynamicky se rozšiřující. V tomto případě využijeme koncepce objektů a necháme enginu volnost při distribuci procesorového času mezi jednotlivé výpočty realizované pomocí generátorů.

Ukázkové aplikace obou variant jsou součástí této práce a nacházejí se na přiloženém CD.

Budeme-li se chtít přiblížit funkčnosti aplikací zmiňovaných v kapitole 2, půjde v budoucnu s výhodou využít koncepce generátorů. Pro získávání modelu podle skutečnosti poslouží generátory nahrávající data z databází (v tomto případě nebude nutné používat techniky generování plátu krajiny ze čtyř základních map, neboť data na sebe budou přirozeně navazovat). Pro zlepšení vizuálního vjemu lze např. vytvořit generátory poskytující textury a jimi potáhnout stávající model.

Myslím, že v případě pokračování projektu existuje šance dosáhnout velmi kvalitních výsledků. Další náměty pro navazující práci jsou uvedeny v kapitole 6.4.

6. Implementace

6.1 Open Inventor

Open Inventor vyvinula v 90.letech minulého století firma SGI (Silicon Graphics), pro tento projekt je použita freeware implementace, kompatibilní s Open Inventor API, jménem Coin od norské firmy Systems In Motion.

Open Inventor je napsán v jazyku C++ a je objektovou nadstavbou nad grafickou knihovnou OpenGL. Umožňuje programátorovi využívat při návrhu aplikace vyššího stupně abstrakce (ve srovnání s čistým OpenGL) a tím jednodušeji a rychleji dosáhnout kýženého cíle.

3D scénu v Open Inventoru reprezentuje strom, který se může skládat z mnoha typů uzlů (používá se anglický termín nodes). Základním typem jsou uzly popisující geometrii objektu, dále uzly definující vlastnosti objektu (např. materiál). Specializované uzly obsahují prostorové transformace (posunutí, rotace), senzory, časovače, kamery atd. V neposlední řadě je třeba zmínit uzly, které obsahují seznam dalších uzlů a tím de facto umožňují vybudovat stromovou strukturu grafu scény. Vlastní vykreslování probíhá při průchodu vytvořeným stromem a je zcela v režii grafické knihovny.

K dispozici jsou kromě klasického okna pro rendering také různé specializované prohlížeče vybudované scény. Ty poskytuje knihovna SoWin (nebo SoQt, dle operačního systému). Pomocí prohlížečů lze pohodlně ověřovat správnost vytvořeného modelu a lze je s výhodou využít pro rychlý náhled získaných výsledků.

6.2 Správa grafu scény

Důležitou úlohou enginu je správa grafu scény. Je uvedena až zde, když byly již popsány nejzákladnější principy použité grafické knihovny.

Modely jednotlivých plátů krajiny jsou uloženy v objektech třídy `TePatch`. Tyto modely obsahují všechny uzly stromu reprezentujícího příslušný čtverec krajiny. Zejména se jedná o uzly obsahující souřadnice jednotlivých vertexů, normály, definice materiálů určujících vzhled povrchu a uzel obsahující posunutí (translaci) na příslušné místo v prostoru. Má-li být čtverec vyrenderován, engine umístí příslušný podstrom do grafu scény a tím zajistí, že při příštím průchodu stromem bude potřebná oblast vykreslena.

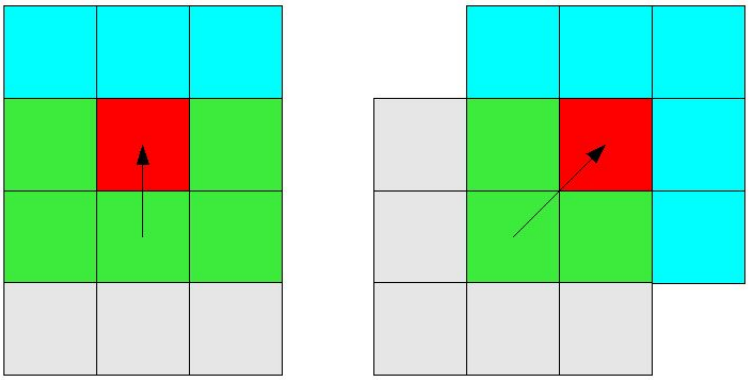
Pokud bychom pouze přidávali nové a nové pláty do grafu scény, narazili bychom velmi brzo na limity dané hardwarovým výkonem použitého počítače. Je tedy důležité zvolit vhodnou strategii a nepotřebné části krajiny, tzn. ty, které již nejsou v zorném poli uživatele, z grafu scény odstraňovat. K odstraňování jsou použity standardní prostředky poskytované knihovnou. Jejich

použitím dochází k odstranění příslušného podstromu z grafu.

Implementovaná strategie správy grafu scény udržuje oblast terénu tvořenou devíti pláty krajiny uspořádanými do čtverce 3x3. Umístění této oblasti je dáno aktuální polohou aktivního objektu, se kterým je svázána kamera pozorovatele. Aktivní objekt se vždy nachází nad plátem přímo uprostřed této oblasti. Podle pohybu objektu scénou dochází v určitém okamžiku k naplánování nových úloh pro generátory. Takový okamžik nastává, přibližuje-li se objekt k některému z okrajů prostředního čtverce.

Pohybuje-li se objekt rovnoběžně se souřadnými osami, jsou požadovány tři nové pláty terénu, pohybuje-li se objekt spíše „po úhlopříčce“, jedná se dokonce o 5 nových stavebních bloků krajiny.

Jakmile objekt překročí hranice jednotlivých čtverců, stane se čtverec pod ním novým středem udržované oblasti a všechny pláty, které byly ve scéně dříve a nezůstaly pokryty nově vytyčenou oblastí, jsou ze scény odstraněny. Situace je znázorněna na obrázku 6.1.



Obrázek 6.1 - Správa plátů při pohybu objektu.

Světle modrou barvou jsou vyznačeny pláty nově generované, červený je nový střed pokryté oblasti, zelenou barvu mají staré pláty, které zůstanou ve scéně a konečně šedě jsou označeny staré pláty, které budou ze scény odstraněny. Šipka znázorňuje provedený pohyb objektu.

Kromě této devítice plátů pod aktivním objektem je ve scéně udržován také vždy jeden plát pod každým z ostatních objektů. Bude totiž v budoucnu potřeba pro detekci kolizí těchto objektů s terénem.

6.3 Terrain Engine API

Pro případné pokračování projektu je jistě jednou z nejdůležitějších informací detailní popis navrženého a implementovaného rozhraní. Vzhledem k rozsahu zdrojových kódů není tento popis uveden přímo v této zprávě, je však součástí elektronicky odevzdávaných výstupů projektu.

Pro generování dokumentace byl využit volně šiřitelný nástroj Doxygen, který poskytuje

přehledný výstup ve formátu HTML. Tento nástroj k vygenerování stránek s dokumentací využívá poznámek přímo ve zdrojovém kódu projektu.

Vygenerovaná dokumentace Terrain Engine API je k dispozici na již zmiňovaných stránkách [10] a taktéž na přiloženém CD.

V přílohách A a B jsou nicméně uvedeny některé, z mého pohledu užitečné, rady, které by snad mohl případný zájemce o pokračování práce využít.

6.4 Náměty na pokračování projektu

Již v předchozím textu byla na vhodných místech zmíněna některá rozšíření, která se nabízejí v případě pokračování tohoto projektu. Tato podkapitola slouží k jejich shrnutí a představení dalších.

Bude-li hlavní motivací v další práci touha po zlepšení kvality 3D modelu krajiny, tzn. touha po zlepšení vizuálního dojmu, půjde s výhodou využít konceptu generátorů.

Model krajiny v současné době tvoří pouze množina vertexů (bodů v prostoru) s přiřazenými materiály a normálami. Materiály v našem případě slouží pro definici barvy povrchu v daném bodě. Podle nadmořské výšky, což je jediné kritérium výběru, jsou rozlišeny tři oblasti. Nejvyšší vrcholky hor jsou bílé, jakoby pokryty sněhem. Pod nimi je pásmo skal a zbytek povrchu pokrývá tráva reprezentovaná zelenou plochou. Krajina je navíc v předem stanovené úrovni proložena rovinou, která má představovat vodní hladinu.

Tento model by šel obohatit o procedurálně generované textury, které by jistě pozitivně ovlivnily celkový dojem. Na podobném projektu pracoval v tomto semestru Jiří Janoušek (viz [10]) a snad by šlo výsledky obou prací zkombinovat. Kromě samotného povrchu lze dále upravovat model vodní hladiny, ať už pomocí textur, nebo nějakým vysoce realistickým animovaným 3D modelem.

Oblohu by měl pokrýt tzv. sky-box, neboli otexturovaná krychle dostatečně velká na to, aby vznikl výsledný dojem realistické oblohy.

Dalších vizuálních efektů je nepřeberné množství. Jedná se např. o různé výbuchy, deformace objektů při kolizích, efekty simulující povětrnostní podmínky (mlha, déšť, mraky, kterými by se dalo proletět), kondenzační čáry vzniklé činností motorů letadel, kouř valící se z poškozeného stroje atp.

Další kapitolou mohou být statické objekty ve scéně. Těmi mohou být např. stromy či budovy. Na takové objekty je engine připraven a jejich zasazení do scény je záležitostí vytvoření odpovídajících modelů.

Opustíme-li vylepšování vizuálního dojmu, vyvstávají další témata, jejichž řešení může pronikavě zlepšit výkon aplikace. Velmi zajímavou oblastí je například autoadaptace generátorů podle skutečných hardwarových možností stroje, na kterém engine běží. Tak lze dosáhnout optimálního výkonu.

Velkou oblastí, která se otevírá jsou techniky LOD. I v této části problematiky se již dle

výsledků prezentovaných na [10] rozběhl výzkum.

Pro nasazení v reálné interaktivní aplikaci bude třeba vytvořit fyzikální modely chování objektů a jim přizpůsobit jejich ovládání. Jedním z prvních úkolů bude detekce kolizí objektů ve scéně. Na těchto úkolech pracuje tým SpaceGame a vzhledem k velmi podobnému rozhraní v této oblasti bude snad možné převzít celý systém bez větších úprav.

Dalšími náměty mohou být algoritmy umělé inteligence. Umělá inteligence musí řídit objekty, které nepodléhají uživateli, ať už se jedná o kooperující spojence, nebo protihráče. Dokáži si představit komponentu enginu, která bude dostávat příděly procesorového času a v co nejkratší době bude ovlivňovat chování ostatních objektů.

Pro zobrazování skutečného povrchu, tedy vlastně jakési plastické mapy, by bylo třeba zabývat se propojením enginu s databází.

7. Závěr

V průběhu řešení se, dle mého názoru, podařilo dojít k dobrým výsledkům ve všech hlavních oblastech výzkumu.

Byla navržena a implementována třída zastřešující operace pracující s výškovými mapami. Tato třída je k dispozici široké veřejnosti a bude se s přibývajícím časem postupně jistě rozrůstat o další možnosti.

V praxi byl ověřen způsob, který zajišťuje hladké navazování samostatně generovaných výškových map. Vývoji tohoto algoritmu předcházela snaha zajistit navazování map již v okamžiku jejich vzniku. Přestože se podařilo tento postup implementovat, bylo od něj v zájmu rychlosti a výkonu aplikace upuštěno a přednost dostal postup, který je popsán v kapitole 4.3.

Engine nekonečné krajiny byl navržen a zrealizován. Podařilo se ověřit správnost implementace databáze udržující informace o již vytvořených výškových mapách a výsledných plátech krajiny. Jednou z hlavních myšlenek celé práce je koncepce generátorů. Generátory dostávají od enginu příděly procesorového času a dokáží si požadovanou práci rozdělit na úseky proveditelné dostatečně rychle na to, aby nebyla ohrožena plynulost běhu celé aplikace.

V předchozí kapitole 6.4 byly naznačeny některé náměty na další pokračování výzkumu v podobné oblasti a nápady na případná rozšíření schopností enginu.

Vidíme, že oblast virtuální reality obecně je tak rozsáhlá, že rozhodně není v silách jednoho člověka postihnout všechny cesty, kterými se lze ubírat. Mým cílem bylo vytvořit jakousi kostru enginu, který sice neoslňuje krásou poskytovaného modelu, ale pracuje plynule v reálném čase a bude do budoucna snadno rozšiřovatelný.

8. Literatura

[1] Polack, Trent: Focus on 3D Terrain Programming.
Premier Press 2003, ISBN: 1592000282

[2] GameDev.net
http://www.gamedev.net/reference/list.asp?categoryid=45#88 (květen 2005)

[3] Lighthouse 3D
http://www.lighthouse3d.com/opengl/terrain/ (květen 2005)

[4] Open Inventor tutoriál na ROOT.CZ
http://www.root.cz (květen 2005)

[5] Wernecke, Josie: The Inventor Mentor.
Addison-Wesley Professional 1994, ISBN: 0201624958
Dokument dostupný na URL http://www-evasion.imag.fr/Membres/
Francois.Faure/doc/inventorMentor/sgi_html/ (květen 2005)

[6] Planetside Software: Terragen HomePage
http://www.planetside.co.uk/terragen/ (květen 2005)

[7] Systems in Motion: SIM Scenery HomePage
http://www.sim.no/products/Scenery/ (květen 2005)

[8] TGS TerrainViz HomePage
http://www.tgs.com/pro_div/terrain_viz_main.htm (květen 2005)

[9] FlightGear HomePage
http://www.flightgear.org (květen 2005)

[10] Inventor Projects 2005
http://merlin.fit.vutbr.cz/wiki/index.php?title=Inventor_Projects_2005
(květen 2005)

A. Jak napsat aplikaci využívající engine?

Především je třeba zvážit, k jakému účelu chceme engine využít. Nastávají v zásadě dvě varianty.

První z nich je prosté vygenerování prostorově omezeného modelu krajiny, nebo dokonce pouze čistých dat použitých výškových map s tím, že model budeme již tvořit ve vlastní režii. Tato možnost ale dává smysl pouze v případě, že takových dat bude hodně a nebudou renderovány současně. V opačném případě by totiž bylo mnohem jednodušší vytvořit dostatečně rozměrný jediný objekt třídy `TeHeightMap`. Druhou variantou je aplikace předstírající nekonečnost virtuálního terénu.

Příklady obou aplikací jsou umístěny na CD, které je dodáváno společně s touto zprávou. Na tomto místě pouze shrnu základní postupy vedoucí k získání statického modelu omezené krajiny.

Ze všeho nejdříve je nutné vytvořit samotný objekt třídy `TeEngine`. Již v jeho konstruktoru musíme zvolit rozměry zdrojových výškových map a rozměry oblasti, která je následně jednou takovou mapou pokryta. Zatímco rozměry výškové mapy udávají de facto velikost matice, ve které budou uložena data, rozměry pokryté oblasti odpovídají číselně rozměrům na souřadných osách v grafické knihovně.

```
TeEngine engine = new TeEngine( SbVec2s(65, 65),
                                SbVec2f(64.0f, 64.0f) );
```

Nyní již můžeme požadovat vytvoření plátů krajiny. Ty specifikujeme pomocí souřadnice středu plochy, kterou pokrývají. Poté, vyhovuje-li nám implicitní nastavení generátorů, můžeme již vytvořit 4 zdrojové výškové mapy a z nich data výsledného plátu krajiny. Například takto:

```
TePatch *tmp = NULL;

tmp = engine->getPatch(SbVec2f(0.0f, 0.0f));
tmp->prepareBuildMaps();
tmp->buildHMapFromBuildMaps();
```

Předpokládejme, že již dříve jsme založili kořen grafu scény a pojmenovali ho `root`. Právě získaný model jednoho kusu krajiny do něj zařadíme jednoduše:

```
root->addChild( tmp->getPatchGraph() );
```

Podobně bychom do scény zařazovali švy spojující několik takových plátů krajiny. K jejich získání slouží metoda `getSeamGraph`, jejímž parametrem je určení směru, ve kterém šev leží.

Její použití pak může vypadat např. takto:

```
root->addChild( tmp->getSeamGraph(TePatch::EAST) );
```

Samozřejmě, že v tomto případě musí nutně existovat východní soused plátu, na který ukazuje `tmp`. Jinak by totiž data švu nešlo získat a volání funkce by skončilo neúspěchem.

Aplikace předstírající nekonečnost terénu je taktéž na přiloženém CD. Na tomto místě nebudu již vypisovat fragmenty zdrojového kódu, uvedu pouze logický sled úkolů, které musíme pro aktivaci enginu provést.

Díky tomu, že nyní necháváme zodpovědnost za správu grafu scény enginu, musíme nastavit jeho parametr ukazující na kořen naší scény. Více práce máme nyní s kamerami, jednu potřebujeme pro zobrazení samotné scény, jednu vyžaduje engine pro vykreslování HUD.

Důležitým krokem je vytvoření aktivního objektu a jeho umístění do seznamu objektů v enginu. Poloha tohoto objektu je rozhodující pro vygenerování viditelného úseku krajiny a je jím řízena kamera pozorovatele.

Počáteční inicializaci enginu provedeme zavoláním funkce `initialize()`. Pokud program vypíše chybové hlášení, lze v kódu této funkce dohledat, na co jsme v předešlých krocích zapomněli. Tato funkce totiž kontroluje základní předpoklady nutné pro správný běh aplikace (definice kamer, určení kořenů scény atp.).

Poslední věcí, kterou je třeba zajistit je opakované volání funkce `timeTick()` při každém průchodu hlavním cyklem aplikace. Metod, jak tohoto dosáhnou jistě existuje více, ověřeno je využití senzorů knihovny Open Inventor (konkrétně `SoOneShotSensor`), které umožňují nastavit callback funkci, která je volána vždy, když dojde k aktivaci senzoru. V této callback funkci zjistíme dobu, která uplynula od posledního volání této funkce a zavoláme metodu `timeTick()`. Jejím argumentem je právě vypočtená doba.

B. Jak napsat nový generátor?

Každý generátor by měl být odvozen od abstraktní třídy `TeGenerator`. Tato třída předepisuje rozhraní, ve kterém jsou důležité čtyři metody:

```
protected: virtual void setDefaults();
   public: virtual void updateParams();
           virtual void generate();
           virtual SbBool genStep();
```

Metoda `setDefaults()` nastavuje implicitní parametry ovlivňující funkci generátoru. Tyto parametry jsou uloženy v atributech objektu a to dokonce ve dvou provedeních. První jsou public atributy, které má šanci měnit uživatel, druhé jim odpovídající atributy privátní, které využívá generátor při práci. Výhodou tohoto návrhu je skutečnost, že uživatel může kdykoliv měnit parametry generátoru, aniž by se ohlížel na stav právě probíhající úlohy. Před začátkem práce na nové úloze jsou privátní hodnoty aktualizovány, aby korespondovaly s nastaveními provedenými uživatelem. K tomu slouží funkce `updateParams()`.

Zbývající dvě metody reprezentují samotný výpočet.

Metoda `generate()` nebere ohled na časovou náročnost výpočtu a provede celou práci v jednom kroku. V současné době není v enginu využita, záměrem bylo poskytnout způsob, jak kritické výpočty provést v nouzi bez ohledu na plynulost běhu.

Vlastní výpočetní jádro reprezentuje funkce `genStep()`. Jedná se o jeden krok generátoru, který by měl být časově nenáročný a nemusí vést k dokončení celého objemu práce. Funkce vrací booleovskou hodnotu, která určuje, zda práci dokončila, či zda ji je třeba volat v příštím cyklu opět.

K tomu, aby engine začal nově vytvořený generátor využívat je třeba vytvořit jeho instanci uvnitř objektu enginu. Dále je třeba na vhodné místo (nejčastěji uvnitř funkce `TeEngine::genTick()`) umístit volání metody `genStep()` nového generátoru.

C. Ovládání ukázkových aplikací

Static Terrain Demo

Aplikace slouží k demonstraci správnosti použitého algoritmu navazování výškových map. Výsledek tvoří 9 plátů krajiny propojených pomocí švů.

Pro vizualizaci je použit standardní prohlížeč modelů poskytovaných knihovnou SoWin. Renderování lze ovlivňovat výběrem položek z kontextového menu, které se zobrazí po stisknutí pravého tlačítka myši.

Pohled na model lze měnit použitím ovládacích prvků na okrajích okna a pomocí myši. Levé tlačítko myši slouží k rotaci modelem, prostřední tlačítko k jeho posouvání. Kombinace levého a prostředního tlačítka slouží k přibližování a oddalování.

Ukončení aplikace lze provést standardním způsobem, nebo stiskem klávesy Q.

Engine

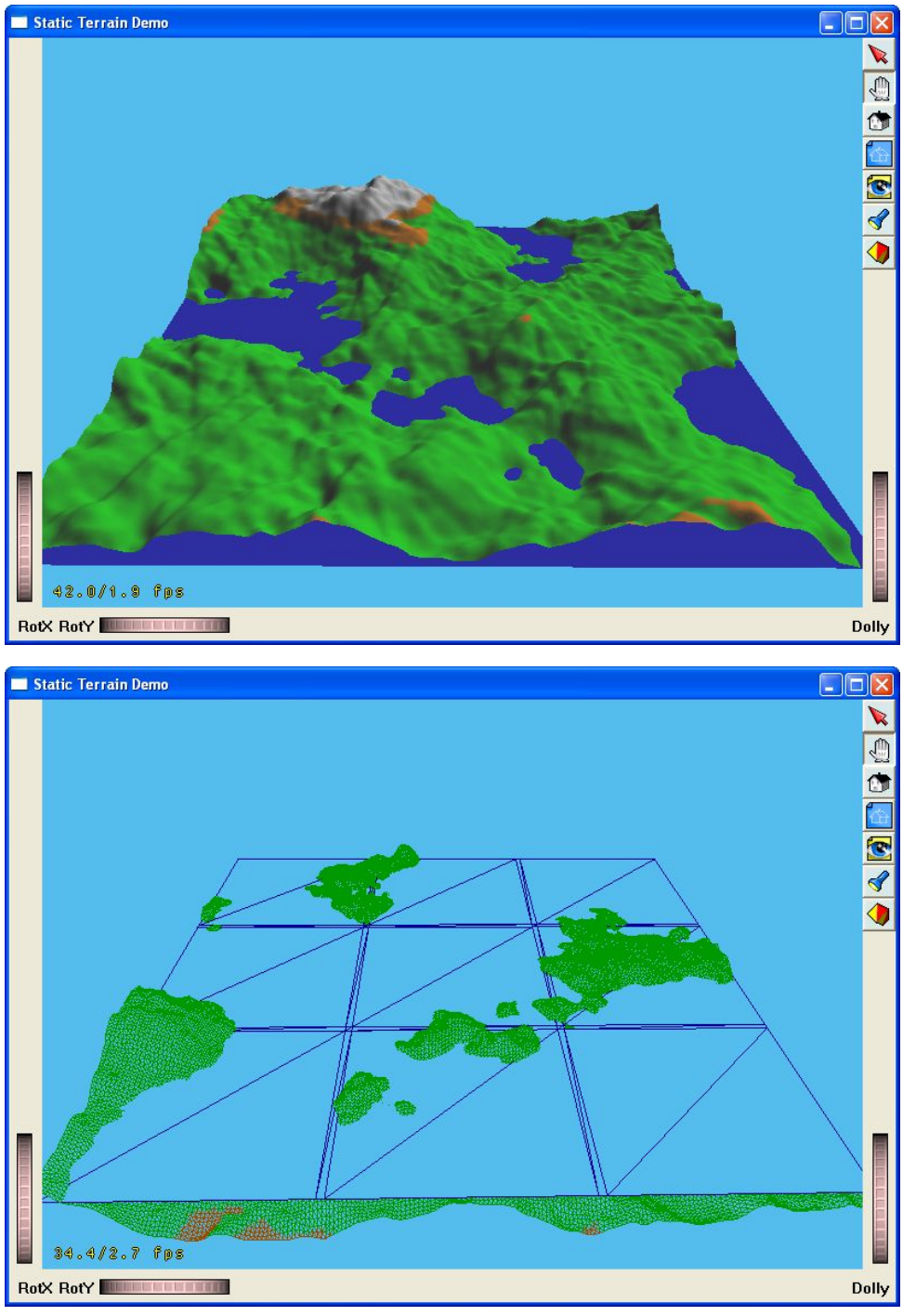
Aplikace demonstruje všechny schopnosti navrženého enginu. Ovládání je možné pouze použitím klávesnice.

Klávesa	Funkce
Kurzorové šipky	Úpravy vektoru rychlosti aktivního objektu.
Num8, Num2	Posun pozice kamery vůči objektu – vpřed, vzad.
Num4, Num6	Posun pozice kamery vůči objektu – vlevo, vpravo.
PageUp, PageDown	Posun pozice kamery vůči objektu – nahoru, dolů.
Num5	Posun pozice kamery vůči objektu zpět do výchozí polohy.
Escape	Ukončení aplikace.

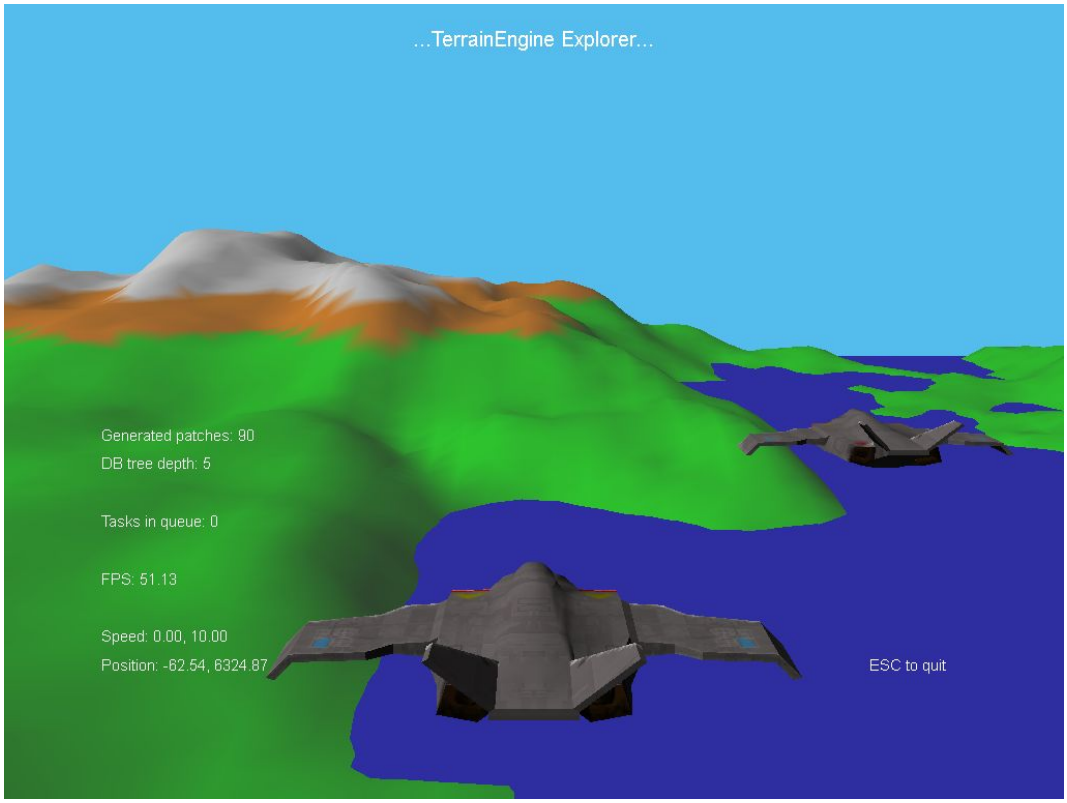
Tabulka 1 - Ovládání ukázkové aplikace.

Ve scéně je kromě uživatele, který se objeví na souřadnici [0,0], také další letoun, který se pohybuje vpřed na souřadnici -60 ve směru osy +y (po spuštění je tedy „vlevo“ od kamery).

D. Obrazová příloha



Obrázek D.1 - Výstup ukázkové aplikace. Stejný model. Na drátěném modelu „vzhůru nohama“ je patrné složení krajiny z jednotlivých plátů a švů mezi nimi.



Obrázek D.2 - Engine nekonečné krajiny za běhu.