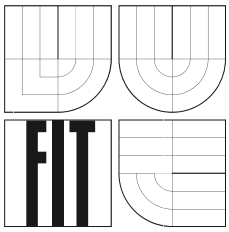
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# Realistické zobrazování krajiny

Bakalářský projekt

**2005** **Jiří Janoušek**

# Realistické zobrazování krajiny

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně dne 3. května 2005



## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .

*Jiří Janoušek*

3. května 2005

## Abstrakt

Tato práce pojednává o realistickém zobrazení krajiny (výškové mapy) pomocí generování textury. Je vytvořena knihovna funkcí a algoritmů pro zpracování matice hodnot (výškových bodů), která je po analýze schopna vytvořit texturu pro tyto hodnoty. Je taktéž nastíněna možnost filtrování dat pro hladší vzhled výsledného povrchu. Knihovna je psána s využitím vývojového balíku COIN3D (Open Inventor) a je přeložitelná pod Windows i pod Linuxem.



## Klíčová slova

3D model krajiny, realistická textura, generování textury, Open Inventor



## Poděkování

Rád bych poděkoval panu Ing. Pečivovi, který mi vždy ochotně vysvětlil, co jsem potřeboval a měl se mnou trpělivost při tvorbě této práce.

## Abstract

This thesis deals with the realistic depiction of the surface (altitudinal map) using texture generation. In order to process the value matrix (height points), library of functions and algorithms has been created, which, after analysis, is able to create the texture for these values. Also a possibility to filter data for the further appearance of the final surface has been outlined. The library is written using development package COIN3D (Open Inventor) and it can be used on Windows and also on Linux platforms.

## Keywords

3D model of terrain, realistic texture, generation of texture, Open Inventor

# Obsah

# Kapitola 1

# Úvod

Právě jste otevřeli práci zaměřenou na zobrazování výškových dat jako realisticky vypadající krajiny. Práce je koncipována jako materiál, který se zpočátku zabývá poměrně obecnými, známými věcmi a pozvolna přechází k hlavnímu cíli práce. Od způsobu nakládání a získávání výškových údajů se přes 3D model krajiny, jeho osvětlení, namapování textury dostaneme postupně až po generování realisticky vypadající textury.

Je zde naznačeno pouze jedno z možných řešení, existuje však mnoho cest. Při rozhodování, které z nabízených řešení vyzkoušet jsem vycházel z toho, že i když se mi některé řešení bude zdát dobré a vyberu si ho pro implementaci, může se nakonec ukázat jako reálně nepoužitelné. Je to však pro ostatní zájemce a pokračovatele taktéž cenná informace, protože „. . . je důležité vědět, kudy ne“, jak jednou prohlásil klasik.

První kapitola (kapitola 2) je pojata velmi obecně a má za úkol nastínit danou problematiku z širšího pohledu. Schválně jsou vzpomenuty i věci zdánlivě nesouvisející s tímto tématem. V praxi však bohužel většinou souvisí všechno se vším, a proto si myslím, že tato kapitola si zde své místo rozhodně zaslouží.

V dalších dvou kapitolách (kapitoly 3 a 4) se budeme zabývat již konkrétnějšími problémy, které ač přímo souvisejí s cílem této práce (podkapitola 2.3), jsou to problémy, jejichž řešení jsou známá (takřka standardizovaná). Jedná se o zobrazování výškových dat jako 3D modelu včetně výpočtu normál povrchu a nanášení textury na tento povrch. V souvislosti s nanášením textury byly udělány některé experimenty s nanášením nascanované turistické mapy na reálná data pořízená z téže mapy (viz. obrázek 4.3).

Poslední, nejrozsáhlejší kapitola, se věnuje generování textury na základě analýzy vlastností výškových údajů (zejména nadmořské výšky a sklonu svahu – gradientu). Myslím, že nemá velký smysl uvádět nějaké exaktní rovnice a výpočty. Spíš je zde snaha o vysvětlení principu a postupu, kterým se k těm principům dá dojít. Pokud bude chtít někdo naznačený postup použít, stejně si jej musí přizpůsobit svým potřebám a k tomu potřebuje, aby principu bezpodmínečně rozuměl.

Na závěr práce je formou přílohy (příloha A) umístěna uživatelská dokumentace k implementované knihovně. Pro lepší názornost je uveden závěrem této dokumentace komplexní příklad použití. Dalším příkladem použití pak jsou zdrojové kódy velmi jednoduché aplikace, která ukazuje možnosti implementované knihovny.

Jako celek má práce za cíl přispět k celku algoritmů a knihoven pro zpracování krajiny na fakultě. Jak úspěšně, posoudí až její nasazení a použití v praxi . . .

# Kapitola 2

# Výškové údaje

## 2.1 Zpracování a získávání výškových údajů

Pro zpracovávání výškových údajů ve formě rastru (tzv. *výškové mapy*) existuje mnoho komerčních nástrojů – většinou ve formě GIS (*Geografické Informační Systémy*). Výškové údaje, které jsou uložené ve formě rastru se již dále poměrně dobře zpracovávají a zobrazují. Příkladem takové GIS aplikace může být ArcGis od firmy ESRI nebo volně šířitelný GRASS, který provozuje naše fakulta. Tyto systémy jsou schopné výškovou mapu zobrazit různým způsobem – ať už formou rastru (s různými barevnými odstíny dle nadmořské výšky), nebo pomocí 3D nástaveb jako model krajiny. Oba systémy jsou schopny takto zobrazený 3D model krajiny pokrýt texturou např. leteckého snímku. Jsou schopny všemožně analyzovat vlastnosti krajiny – např. zda-li na sebe dva body vidí atd.

Získávání těchto údajů je zcela jiný problém – v drtivé většině případů se musejí data získat z terénu ručním měřením a následným přenosem do počítače. Tento způsob získávání dat tato data velmi prodražuje a geografická data mají na dnešním trhu informací poměrně vysokou hodnotu. Dalším způsobem získávání výškových údajů o krajině je odečet z mapy dle vrstevnic. Je to však pro člověka činnost také velmi zdlouhavá a pro software velmi náročná na analýzu rastrového obrázku. Nejsou-li vrstevnice dostatečně odlišitelné od okolí, stává se téměř nemožnou. Touto problematikou se na fakultě zabývá taktéž jeden z projektů.

Data pro tento projekt byla získána ručním odečtem z turistických map v měřítku $1:50\,000$. Bylo možno sice použít nějaká nagenerovaná data, ale byla snaha vyzkoušet, jak si knihovna poradí s reálnými daty. Je totiž velmi obtížné najít generátor krajiny, který by krajinu nageneroval opravdu reálně se vším všudy.

## 2.2 Zobrazování povrchu výškové mapy

V předchozí kapitolce je popsáno, jak můžou s výškovými mapami pracovat GIS systémy. Ty jsou však zaměřeny na práci se skutečnými údaji – leteckými snímky, rastrovými mapami (turistické mapy) apod. Ale existují i programy pro tzv. fotorealistické generování krajiny, které nejenže vygenerují výškovou mapu, ale také ji reálně zobrazí. Bohužel ji zobrazí pouze z nějakého úhlu jako jeden pohled – obrázek. Nejsou většinou schopny vytvořit nějaký použitelný prostorový model. Příkladem takového softwaru může být třeba Terragen.

Smozřejmě existuje software pro profesionální využití, který dokáže na daný povrch umístit objekty a vymodelovat krajinu včetně povrchu a vše zobrazit jako 3D model. Jedná se však v drtivé většině o velmi složité a drahé komerční nástroje používané krajinnými architekty apod. Většinou

je tento software propojený s GIS systémem a výsledek je pak téměř dokonalý.

Dále existuje skupina softwaru, který se zabývá simulacemi a zobrazováním jejích výsledků – např. jak bude vypadat krajina za několik let při postupující erozi, nebo po povodni atd. Všechny zde uvedené softwarové systémy nějak souvisejí se zobrazování povrchu ve formě výškové mapy, proto se zde jimi zabýváme.

## 2.3 Cíl práce

Vzhledem ke skutečnosti, že na drtivé většině reálně řešených projektů pracuje více lidí a jejich řešení musejí být univerzální a pokud možno znovupoužitelná, má tato práce za cíl přispět k celku programů, knihoven a algoritmů pro zpracování a generování krajiny vyvíjených na této fakultě studenty. Většina prací je zaměřená na tvorbu či generování výškové mapy pro různé potřeby. Také jsou již hotovy práce se zaměřením na vkládání objektů do takto vygenerované krajiny. Například generování městské zástavby, silnice apod. Zatím je však minimum projektů, které se zabývají vzhledem krajiny – např. pro potřeby leteckých simulátorů. Proto je tato práce zaměřena na zpracování knihovny, která bude schopná generovat povrch krajiny formou textury na základě vlastností této krajiny (sklon svahu, nadmořská výška, atd.).

Neboť většina dalších řešených projektů je vyvíjena pro volně šířitelnou implementaci Open Inventoru – Coin3D, bylo i zde na základě doporučení vedoucího práce rozhodnuto pro zpracování práce pro toto vývojové prostředí v zájmu použitelnosti tohoto kódu a algoritmů v dalších projektech.

# Kapitola 3

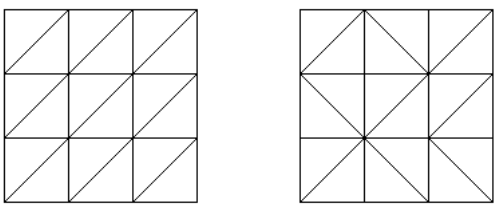
# Výškový model terénu

## 3.1 Popis dat

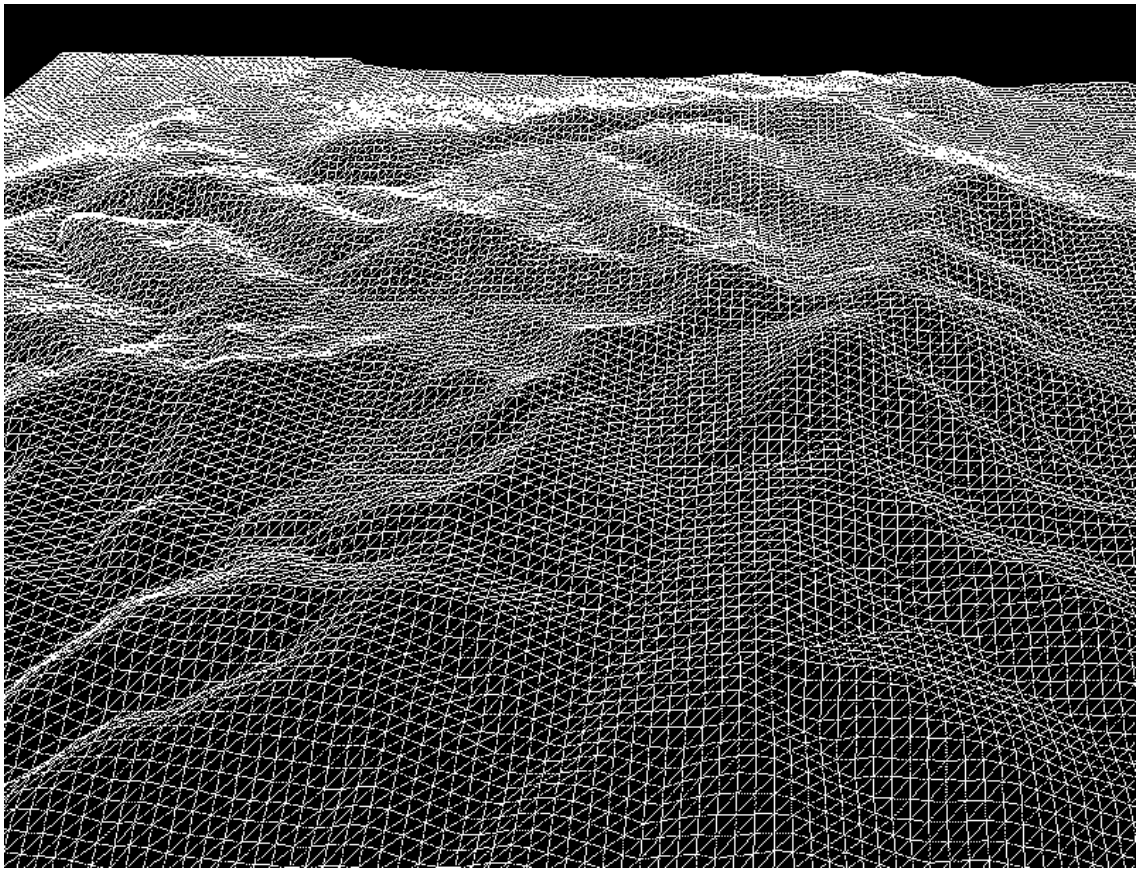
Data pro výškový model terénu bývají uložena buď v databázi GIS systému nebo ve formě nějaké tabulky v souborech. Původní data využitá pro tuto práci byla pořízena do programu MS Excel a jejich použití umožňuje export do textového souboru ve formátu CSV (*Comma Separated Values*). Tento formát byl pro svou jednoduchost zvolen za vstupní formát dat do knihovny. Knihovna načte tato data do pole. Do tohoto pole je možné samozřejmě načíst již hotová data v jiném poli a načítání ze souboru si zařídit jinde v programu, popř. si data generovat. Podrobněji viz. příloha A – Uživatelská dokumentace.

## 3.2 Způsob zobrazování výškových dat

Výškový model terénu je v zásadě možno zobrazovat několika různými způsoby založenými na stejném principu – zobrazení pomocí trojúhelníkové sítě. Záleží na způsobu pořízení dat – zda je to matice hodnot (rastr) nebo jsou data vztažena pouze k bodům bez pravidelného uspořádání. Vzhledem k tomu, že poslední uvedená možnost (data bez pravidelného uspořádání) se většinou vztahuje ke GIS systémům a vyžaduje poměrně složitou analýzu před zobrazením, nebudeme se jí zde podrobněji zabývat. Máme-li matici hodnot (rastr), máme v zásadě dvě možnosti, jak tato data seskupit do trojúhelníků – pravidelně nebo nepravidelně (v závislosti na vlastnostech terénu). Výsledná skupina trojúhelníků se nazývá většinou *strip*. Příkladem pravidelného a nepravidelného stripu – hlediskem pro orientaci trojúhelníků může být například hrana ve směru většího gradientu – může být obrázek 3.1. Jak situovat trojúhelníky, aby byla výsledná zobrazená krajina co nejhladší je taktéž součástí řešení některé z prací na fakultě. Pro tento projekt bylo zvoleno zobrazení pravidelným stri-



Obrázek 3.1: pravidelný a nepravidelný strip



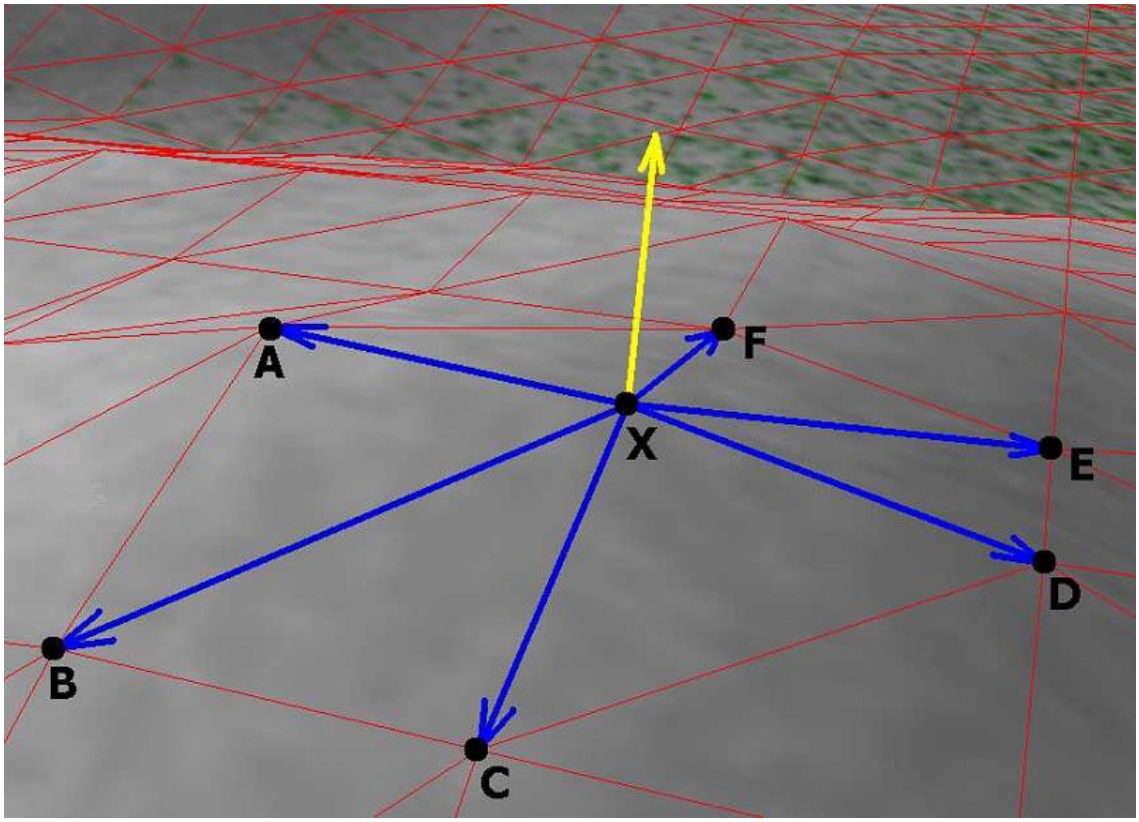
Obrázek 3.2: zobrazení konkrétních dat pravidelnou trojúhelníkovou sítí

pem. Jednoduchým algoritmem se projde pole a body se propojí pravidelnou trojúhelníkovou sítí. Využívá se přitom tříd Coinu (viz. obrázek 3.2).

## 3.3 Povrch a jeho osvětlení

Dalším problémem, který se při zobrazování výškových dat řeší, je osvětlení povrchu (scény) modelu. Když nebude světlo, nebude nic vidět. Protože však náš model nemá plynulé přechody, ale je poskládán z trojúhelníků, je nutné pro každý vrchol (bod) modelu vypočítat normálu. Tato normála bude určovat, jak se bude bod (a potažmo jeho okolí) chovat k dopadajícímu světlu. Jak vznikne normála (pro bod X) je dobře patrné z obrázku 3.3.

Zde je žlutou barvou znázorněna normála, která byla vypočítána z vektorů z bodu X do všech bodů A-F. Postup je následující – vypočítáme vektorový součin dvojic normál ve stejné rovině (patřící jednomu trojúhelníku). V našem případě to bude $\overrightarrow{XA} \times \overrightarrow{XF}$, $\overrightarrow{XB} \times \overrightarrow{XA}$, $\overrightarrow{XC} \times \overrightarrow{XB}$ atd. až $\overrightarrow{XF} \times \overrightarrow{XE}$. Vznikne nám šest vektorů, které stačí vektorově sečíst a dostáváme žlutou normálu. Systém OpenGL vyžaduje, aby všechny normály byly v normovaném tvaru (tzn. měly délku rovnu jedné). Proto musí být ještě po výpočtu každá normála přepočítána tak aby tuto podmínku splňovala. Je sice možné využít funkcí pro automatickou normalizaci za běhu, které poskytuje nastavení OpenGL, výrazně to však zpomaluje výsledný program.



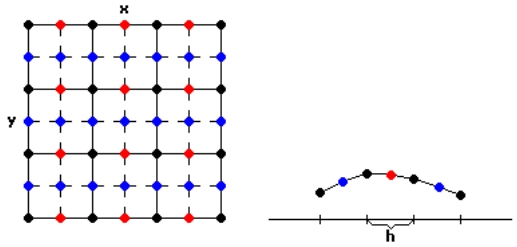

Obrázek 3.3: výpočet normály povrchu

## 3.4 Vyhlazování povrchu dopočítáním bodů

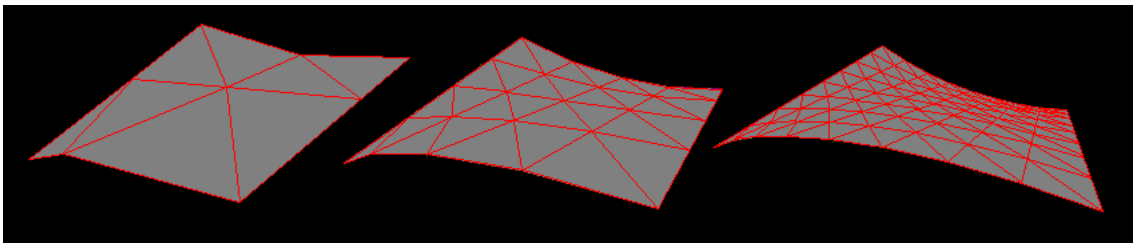
Snaha o dokonalejší vzhled povrchu vedla k hledání jednoduchého a rychlého způsobu, jak zvýšit počet bodů modelu a tím dosáhnout hladšího povrchu. Z různých způsobů, které se nabízely byla nakonec vybrána aproximace Lagrangeovým polynomem a dopočet hodnoty bodu dle tohoto polynomu. Princip metody spočívá v procházení pole hodnot a mezi každé dvě hodnoty se dopočítá hodnota třetí, která je získána pomocí aproximace 4 okolních bodů Lagrangeovým polynomem 3. stupně. Vše je patrné z obrázku 3.4. Představíme-li si pole jako matici hodnot, pak procházíme tuto matici nejprve ve směru x-ové osy, dopočítáme body a následně ve směru y-nové osy a dopočítáme nové body jednak z bodů starých a jednak z bodů vypočítaných již při minulém průchodu.

V levé části obrázku jsou vyznačeny červeně body, které se dopočítávají při prvním průchodu a modře body, které se dopočítávají při druhém průchodu. V pravé části je pak naznačena myšlenka aproximace pomocí polynomu 3. stupně. Skutečnost, že body mají od sebe stejnou vzdálenost *h* je pro nás velkou výhodou stejně jako to, že známe přesně polohu dopočítávaného bodu. Využijeme-li všech těchto skutečností, stačí si pak obecně vyjádřit vzorec Lagrangeova polynomu 3. stupně, z něj odvodit vztah pro náš konkrétní bod a při výpočtech do něj již jen dosazovat hodnoty okolních bodů. Vznikne tím poměrně rychlá metoda, která přes drobnou nepřesnost vykazuje velmi zajímavé výsledky.

Je samozřejmě nutné speciálně ošetřit situace na krajích matice – a to tím, že počítaný bod není uprostřed (mezi 2. a 3. bodem), ale na kraji (např. mezi 1. a 2. nebo 3. a 4. bodem). Na obrázku 3.4 v jeho pravé části je červeně znázorněný bod dopočítávaný uprostřed matice a modře znázorněny




Obrázek 3.4: dopočet bodů pomocí Lagrangeova polynomu



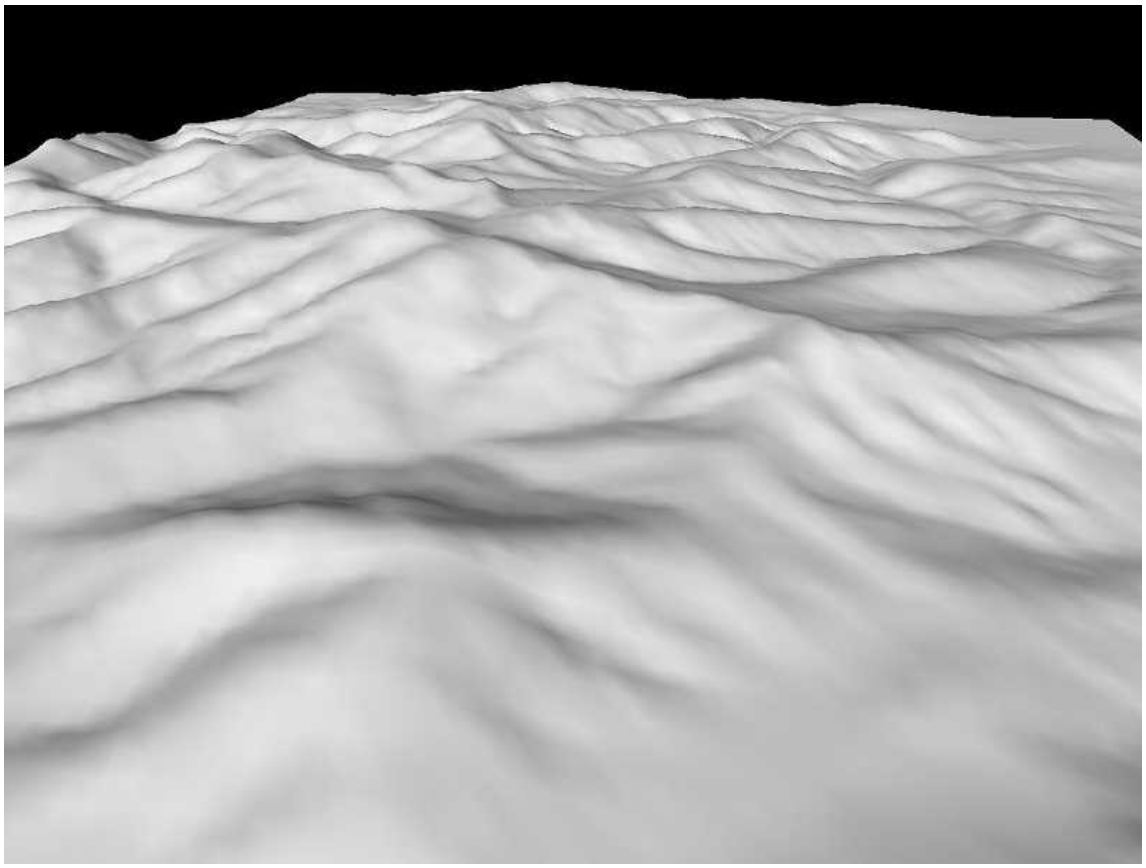
Obrázek 3.5: výsledky po prvním a druhém průchodu filtrem

body dopočítávané na okrajích matice. Výsledky aplikace metody jsou znázorněny na obrázku 3.5. První část znázorňuje původní matici bodů, druhá část tutéž matici po prvním průchodu filtrem a poslední část je dvakrát vyfiltrovaná matice bodů.

Jedna nepříjemnost tento jinak celkem dobrý algoritmus však přece jen provází. Tou nepříjemností je poměrně velké přibývání bodů a tím pádem také trojúhelníků ve scéně. Máme-li totiž na počátku $x \cdot y$ bodů, po průchodu filtrem jich máme $4xy - 2x - 2y + 1$, což je velký nárust.

## 3.5 Výsledný model

V této fázi máme již hotový 3D model terénu, dle libosti vyfiltrovaný a můžeme ho zobrazit. Model by mohl vypadat např. jako na obrázku 3.6 – zde je model zobrazen bez jakékoli textury (bude náplní dalších kapitol), pouze osvětlený. Barva modelu je dána barvou jednotlivých bodů. Barvu jednotlivých trojúhelníků OpenGL automaticky dopočítá dle barev jeho vrcholů. Mezi různými barvami pak utvoří plynulé přechody (záleží na nastaveném režimu). Dalším faktorem ovlivňujícím barvu výsledného modelu je intenzita dopadajícího světla – zde (obrázek 3.6) je barva bodů bílá a intenzita dopadajícího světla je rovna 1 (nejvyšší).



Obrázek 3.6: výsledný osvětlený model terénu s propočítanými normálami

# Kapitola 4

# Pokrytí texturou

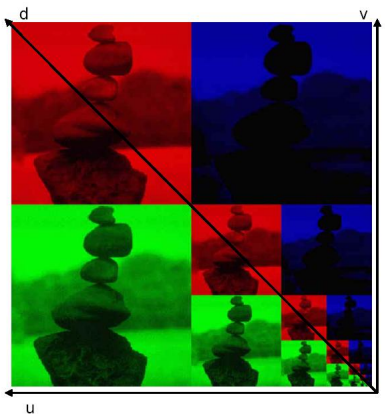
## 4.1 2D Textura

2D Textura je většinou nějaký obrázek, který chceme dát na povrch nějakého polygonu ve scéně. Nejjednodušší textura je prostě pole hodnot seskupené po třech nebo čtyřech bytech, které udávají hodnotu v kanálech RGB nebo RGBA pro každý bod. Taková matice hodnot má nějaké rozměry $x$ a $y$. Tedy velikost textury v nekomprimovaném tvaru je $x \cdot y \cdot 3$ případně $x \cdot y \cdot 4$.

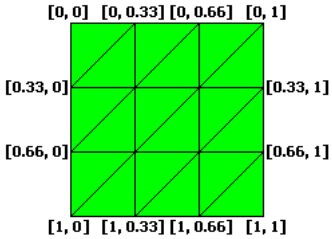
Typů textur je velké množství a každý typ se hodí pro jinou aplikaci. Existuje například tzv. *mip-mapping* (viz. obrázek 4.1), kdy je textura uložena ve speciálním formátu, který obsahuje tutéž texturu v různých velikostech. Taková textura se například hodí pro použití při proměnné úrovni detailu (jsme-li od objektu dál, vidíme méně detailů – užita menší textura, jsme-li naopak blíž, vidíme více detailů – užita textura o větším rozlišení). Také je vhodná při zobrazování velkých textur na dálku – její použití odstraňuje *aliasing*. Dalším typem je textura, která vytváří efekt hrbolatého povrchu – této technologii se říká *bump-mapping*.

## 4.2 Mapování textury na objekt

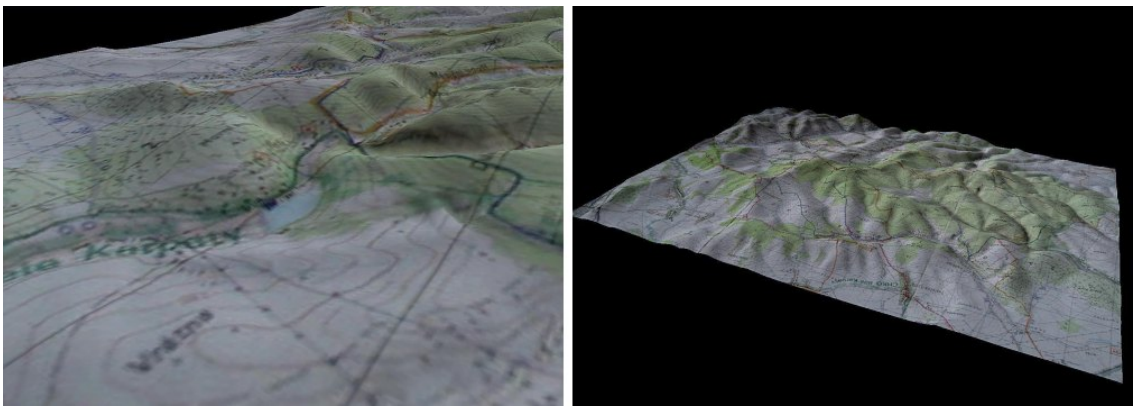
V našem případě se jedná o relativně primitivní problém, protože budeme mapovat 2D texturu na povrch o tvaru obdélníku. Jediný problém, který je nutné vyřešit, jsou texturovací souřadnice.




Obrázek 4.1: příklad textury pro mip-mapping (převzato z [6])




Obrázek 4.2: texturovací souřadnice



Obrázek 4.3: mapování scanované textury na odpovídající výškovou mapu

OpenGL sice umožňuje automatické generování texturovacích souřadnic, avšak rychlejší je souřadnice dopředu předpočítat a navíc OpenGL si texturu otočí vždy delší stranou k počáteční hraně polygonu, což ne vždy vyhovuje.

Mapování textury (výpočet texturovacích souřadnic) probíhá tak, že každému bodu v polygonu, na který chceme texturu umístit, přiřadíme souřadnici v textuře. V textuře se souřadnice určují relativně a to od 0 do 1 v obou rozměrech. Vše je dobře patrné z obrázku 4.2.

Texturovací souřadnice se pochopitelně generují pro všech 16 bodů polygonů, zde jsou označeny pro názornost jenom některé body. V průběhu práce – po přechodnou dobu, než byla textura generována – bylo implementováno nanášení obrázku jako textury na zobrazený povrch. Byla nascanována turistická mapa odpovídající skutečné krajině a po vzoru GIS systémů zobrazena na modelu krajiny. Výsledky můžete posoudit na obrázku 4.3.

# Kapitola 5

# Generování textury

## 5.1 Volba metody

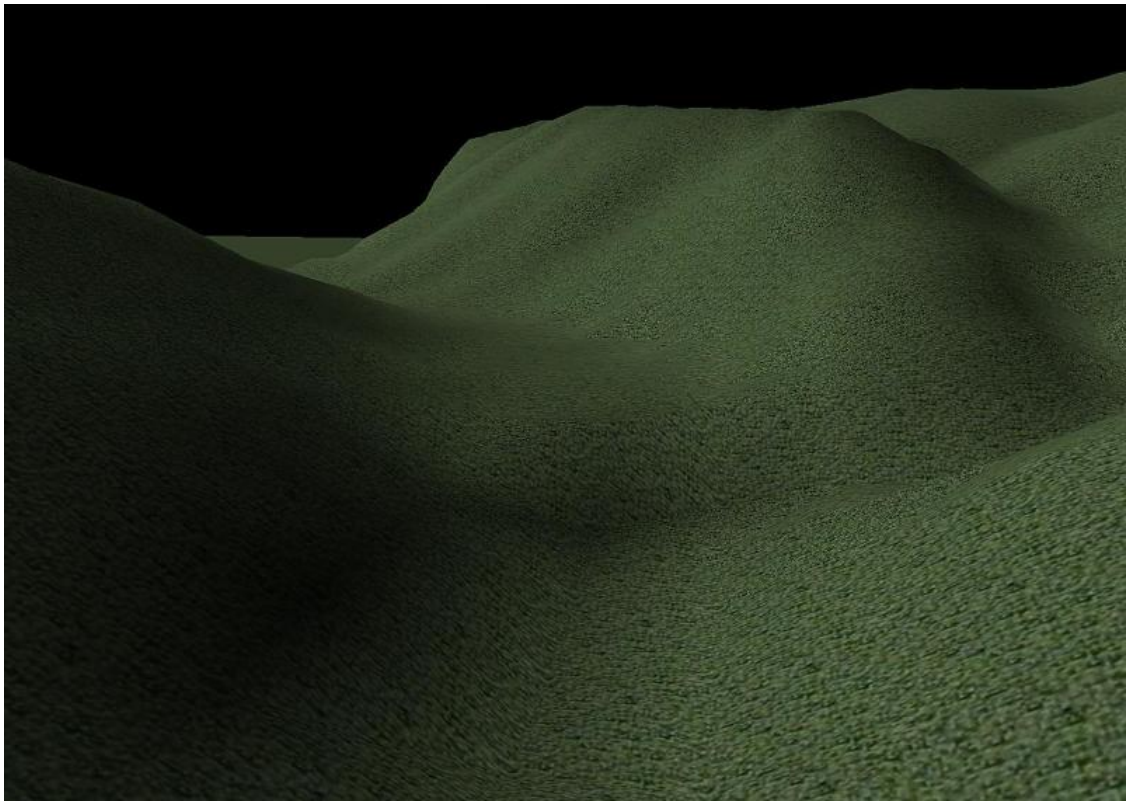
Až doposud jsme se zde zabývali relativně obecnými věcmi a řešili jsme problémy (až na pár výjimek), jejichž řešení bylo již několika způsoby vyřešeno. Nyní však nastává okamžik, kdy bylo nutné začít vymýšlet a zvažovat další postup, který by mohl vést k vytyčenému cíli – co nejrealističtěji zobrazit krajinu bez její předchozí znalosti (tzn., že data jsou dodána až za běhu programu). Z různých úvah vyplynulo několik metod, jak cíle dosáhnout:

- Rozdělit model na jednotlivé úseky a ty pokrýt kousky předpřipravených textur
- Multitexturing
- Generovat texturu za běhu a namapovat ji na povrch modelu

Zabývejme se nyní první metodou. Tento přístup má však hned několik nevýhod. První nevýhodou je, že analýza a rozdělení modelu na jednotlivé úseky by trvala programu příliš dlouho, což není použitelné např. v leteckém simulátoru, kde se za běhu krajina teprve generuje. Další nevýhodou jsou samotné textury. Je velký problém najít (nebo vytvořit) kousky textury tak, aby se daly libovolně kombinovat a připomínaly alespoň vzdáleně to, co připomínat mají. Takže tento způsob dalšího pokračování práce byl nakonec zavrhnut.

Další metodou, jak vygenerovat povrch krajiny by byl multitexturing. Ten se z počátku zdál být ideálním řešením, ale narazili bychom zde na jednu dosti podstatnou překážku – a tou je výkon. Při multitexturingu klesá téměř lineárně výkon s počtem textur. Vzhledem k tomu, že bychom potřebovali vytvořit alespoň lesy, louky, skály, sníh a řeky, potřebovali bychom 5 textur, což znamená asi 5-ti násobné snížení výkonu při renderování scény – a to je nevyhovující. Než byl tento způsob zavrhnut, byl učiněn pokus pokrýt krajinu texturou o velikosti $64 \times 64$ texelů, která má připomínat les a bude na ní co nejméně vidět skutečnost, že se opakuje. Výsledek je vidět na obrázku 5.1. Tato metoda se zpočátku také zdála velmi vhodnou ještě z jiného důvodu (a ten naopak trochu nevýhodou poslední, nakonec vybrané, metody) – rozlišení textury (zde například lesa) může být poměrně vysoké (třeba $256 \times 256$ texelů) a v paměti je uložena jen jednou a přitom nezáleží na tom, kolikrát je použita. Povrch je tím pádem zobrazen velmi detailně.

Poslední metodou je vytvoření jedné textury přímo za běhu programu a její namapování na celý povrch modelu. Tato metoda byla nakonec zvolena, neboť má nejméně protiargumentů. Je ideální v tom, že se textura tvoří přímo na daný povrch, a proto může být „šitá přesně na míru“. Další výhodou je, že téměř nezáleží na počtu detailů, které se na povrchu budou generovat (lesy, řeky, atd.). Říkám téměř, neboť neovlivňuje to sice velikost textury, ale ovlivňuje to trochu čas potřebný



Obrázek 5.1: opakující se textura lesa

k jejímu vygenerování. Jediným omezením této metody je však právě velikost textury – paměť grafického adaptéru není neomezená a OpenGL stanovuje pro daný stroj maximální možný rozměr textury. Na testovacím stroji bylo například omezení stanoveno na $2048 \times 2048$ texelů. Navíc správa tak velkých textur je sama o sobě celkem náročná. Je však docela překvapivé, jakých výsledků bylo pomocí této metody dosaženo.

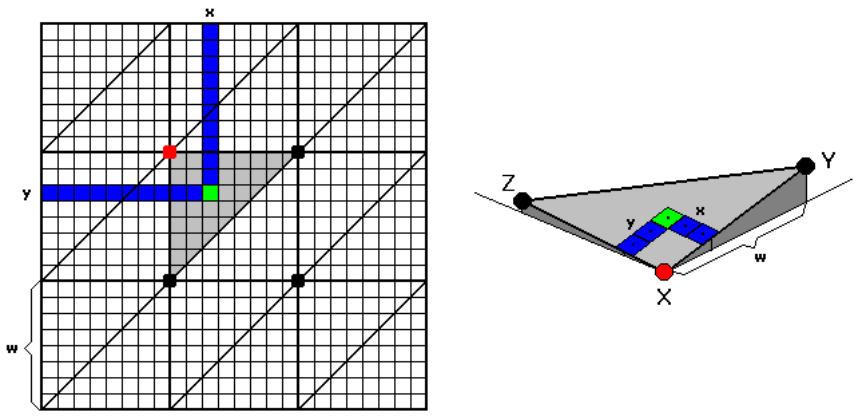
## 5.2 Vlastní tvorba textury

### 5.2.1 Uložení textury

Textura je uložená v poli, kde každá po sobě jdoucí trojice bytů určuje jeden texel v textuře – kanály RGB. Je možno zvolit i kanály RGBA (pak by to byly čtveřice) nebo jenom odstíny šedé (1 byte) atd. Pro jednoduchost bylo zvoleno RGB – tedy pole pro uložení textury má $3xy$ bytů, kde $x$ a $y$ udává šířku a výšku textury.

### 5.2.2 Analýza povrchu

Abychom mohli začít pole textury plnit hodnotami, musíme znát nějaké informace o každém texelu textury. Řekněme, že se budeme rozhodovat dle nadmořské výšky a sklonu svahu.




Obrázek 5.2: výpočet bodů trojúhelníku připadajícího texelu textury

### Nadmořská výška

Nadmořská výška se dá lehce pro každý texel spočítat z trojúhelníku, ve kterém se nachází. Nejprve je tedy nutné určit trojúhelník, ve kterém se texel bude nacházet. Respektive je třeba určit 3 body, které určují ten trojúhelník. Zde se právě s výhodou využije vlastnosti stripu, který má všechny trojúhelníky orientovány jedním směrem. Kdyby se jednalo o nepravidelný strip, byla by situace mnohem složitější. Stačí odvodit z indexu do pole textury souřadnice texelu v textuře a tyto souřadnice podělit počtem texelů mezi dvěma body modelu povrchu. Jasnější to bude z obrázku 5.2.

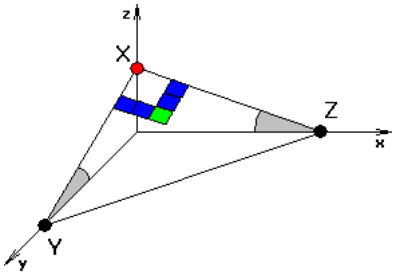
Na obrázku $w$ udává počet texelů mezi dvěma body pole a $x$ a $y$ souřadnice zeleně zvýrazněného texelu v textuře. Podělíme-li $x \operatorname{div} w$ a $y \operatorname{div} w$, dostaneme souřadnice červeně zvýrazněného bodu povrchu. Nyní již není problém vztáhnout souřadnice $x$ a $y$ nikoli k počátku textury, ale k tomuto bodu a to podělením souřadnic $x \bmod w$ a $y \bmod w$. Zbývá určit do kterého ze dvou trojúhelníků rozdělujících zvýrazněné 4 body náleží náš zelený texel. Zde právě využijeme znalosti orientace trojúhelníků a řekneme, že je-li polovina součtu souřadnic texelu vztažených k červenému bodu menší než $w$, je texel v prvním (šedě zvýrazněném) trojúhelníku, jinak je ve druhém.

Nyní již známe trojúhelník, body a přesnou polohu texelu. Na druhé části obrázku 5.2 je znázorněn výpočet nadmořské výšky texelu. Zelený texel má výšku jako červený bod $X$ plus nárůst výšky směrem k bodu $Y$ plus nárůst výšky směrem k bodu $Z$. Je-li některý z bodů $Y$ nebo $Z$ níž než $X$, je nárůst výšky k němu prostě záporný. Tímto na první pohled složitým algoritmem, který se však dá formulovat jedním matematickým vzorcem, lze získat nadmořskou výšku každého texelu textury.

### Sklon svahu – gradient

Potřebujeme-li zjistit gradient nějakého texelu, musíme si o něm zjistit stejné informace, které byly popsány v předchozí části. Dále využijeme toho, že náš trojúhelník je pravoúhlý a tím pádem velmi snadno určíme parciální derivace podle jednotlivých os. Naznačeno je vše na obrázku 5.3.

Značení bodů odpovídá minulému obrázku 5.2. Uvědomíme-li si, že parciální derivace určuje přírustek funkce (zde tangentu úhlu s rovinou $x$,$y$) ve směru příslušné osy a dále si uvědomíme, že tangens úhlu je protilehlá strana ku přilehlé, zjistíme, že stačí vzít absolutní hodnotu rozdílu mezi body a podělit jejich vzdáleností. Tímto způsobem dostaneme tangenty obou šedě naznačených úhlů.




Obrázek 5.3: výpočet gradientu

Dále si musíme uvědomit, že gradient v daném bodě je vektorovým součtem směrových gradientů ve směru jednotlivých os, přičemž hodnota takového gradientu je dána velikostí jeho vektoru. Naše parciální derivace svírají mezi sebou pravý úhel a tudíž je nasnadě dle Pythagorovy věty vypočítat gradient celého trojúhelníku. Tento gradient nám určuje tangentu úhlu svíraného trojúhelníkem s rovinou, což znamená, že chceme-li skály umístit tam, kde je sklon svahu větší než $30°$, stačí porovnat hodnotu gradientu a hodnotu $tan(30°)$.

Nyní již známe vše potřebné k rozhodování (nadmořskou výšku i gradient) a můžeme přistoupit k problematice vlastního generování jednotlivých texelů textury.

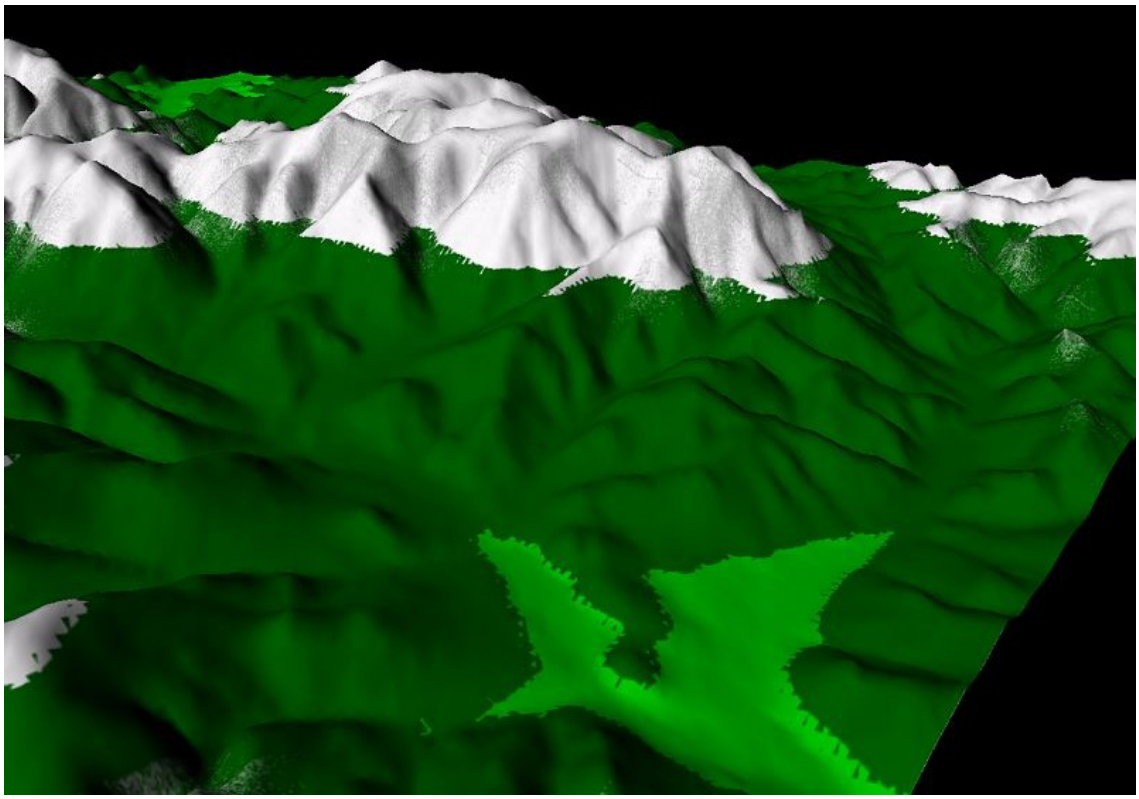
### 5.2.3 Generování jednotlivých texelů

#### Základní povrch – lesy, louky, vrcholky hor, skály

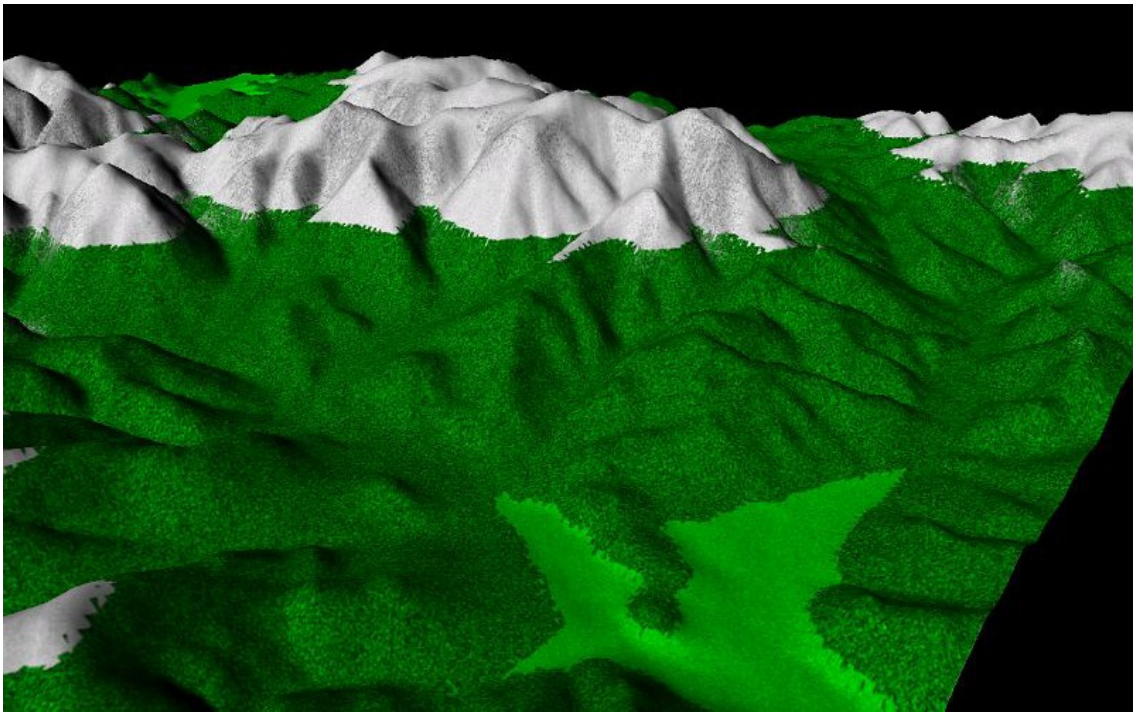
Textura je generována po jednom texelu, přičemž u každého texelu se rozhoduje o jeho barvě na základě nadmořské výšky a gradientu. Pro nadmořskou výšku jsou stanoveny implicitní limity, které jsou využity, pokud si uživatel nezvolí jinak (podrobnosti viz. příloha A – Uživatelská dokumentace). Limity jsou nížina, lesní pásmo a vrcholky hor. Gradient rozhoduje o umístění skal. Přičemž, opět v závislosti na nastavení, je rozdíl, ve kterém výškovém pásmu se texel nachází. Je větší pravděpodobnost, že se skála bude vyskytovat ve vyšší poloze (vrcholky hor), proto jsou implicitně rozdílné limitní gradienty pro nížiny a lesní pásmo a pro vrcholky hor. Např. pro lesní pásmo a níž je stanoven limitní gradient na $45°$ a cokoli bude mít větší gradient, bude skála, kdežto na vrcholcích hor bude skálou vše, co bude mít gradient větší než $30°$.

Generování skal není však tak jednoduché, jak by se mohlo zdát. Protože, pokud bychom se snažili udělat skálu jen tam, kde gradient skutečně překročí limit, nepůsobilo by to dostatečně realisticky. Proto byl vymyšlena rekursivní metoda, která jakoby „rozeseje“ dalších pár texelů jako skály kolem aktuálně zpracovávaného texelu označeného za skálu. Na první pohled se může tento přístup zdát jako divný, ale po několika pokusech s jinými přístupy působil celek nejpřirozeněji. Vygenerovaná pásma s naznačenými skálami můžeme vidět na obrázku 5.4.

Při pohledu na obrázek 5.4 je však jasné, že jednotlivá pásma nejsou nikdy ve skutečnosti tak ostře ohraničena a navíc les, louka ani vrcholek hory nikdy nemá tak jasnou a plnou barvu. Na vyřešení barvy (vzoru, který by připomínal les, louku nebo vrcholek hory) použijeme náhodné generování barevné složky určující majoritní barvu v daném pásmu. Například u lesa převládá tmavší zelená, což se dá vyjádřit například RGB kombinací 0,150,0. Vygenerujeme-li ale místo čísla 150 náhodné číslo od 100 do 200, bude výsledek mnohem reálnější, než jednolitá barva. Tento postup je využit u všech 4 základních složek – lesy, louky, skály a vrcholky hor. Výsledek takto upraveného generování je vidět na obrázku 5.5.



Obrázek 5.4: základní vygenerovaná pásma s naznačenými skálami



Obrázek 5.5: vylepšená vygenerovaná pásma s naznačenými skálami

Realističnosti již oproti obrázku 5.4 rozhodně přibylo. Zbývá nám však vyřešit problém s ostrými přechody mezi jednotlivými pásmy. U minulého problému jsme využili generování náhodných čísel. Taktéž tento problém budeme řešit pomocí generování náhodných čísel – tentokrát však nebudeme upravovat přímo barvevné složky texelu, ale ovlivníme celé rozhodnování mnohem dříve, a to již při výpočtu nadmořské výšky. Každou výšku, kterou vypočteme změníme o libovolně (tedy vhodně) náhodně zvolené číslo. Tím dosáhneme efektu, že jednotlivá pásma, se mezi sebou jakoby promísí, neboť např. některé texely v pásmu lesa se budou chovat, jako by byly ještě v pásmu nížin. Takovýchto texelů samozřejmě ubývá směrem od hranice přechodů. Označuje-li např. proměnná `height` výšku, může být taková vhodná změna zapsána např. takto:

```
height = height + (int)(((rand() - RAND_MAX)/(float)RAND_MAX)*150);
```

kde konstanta 150 je právě tím libovolně, tedy vhodně, zvoleným číslem rozptylu. Konečný výsledek vygenerované textury si můžeme prohlédnout na obrázku 5.6.

**Další objekty – řeky**

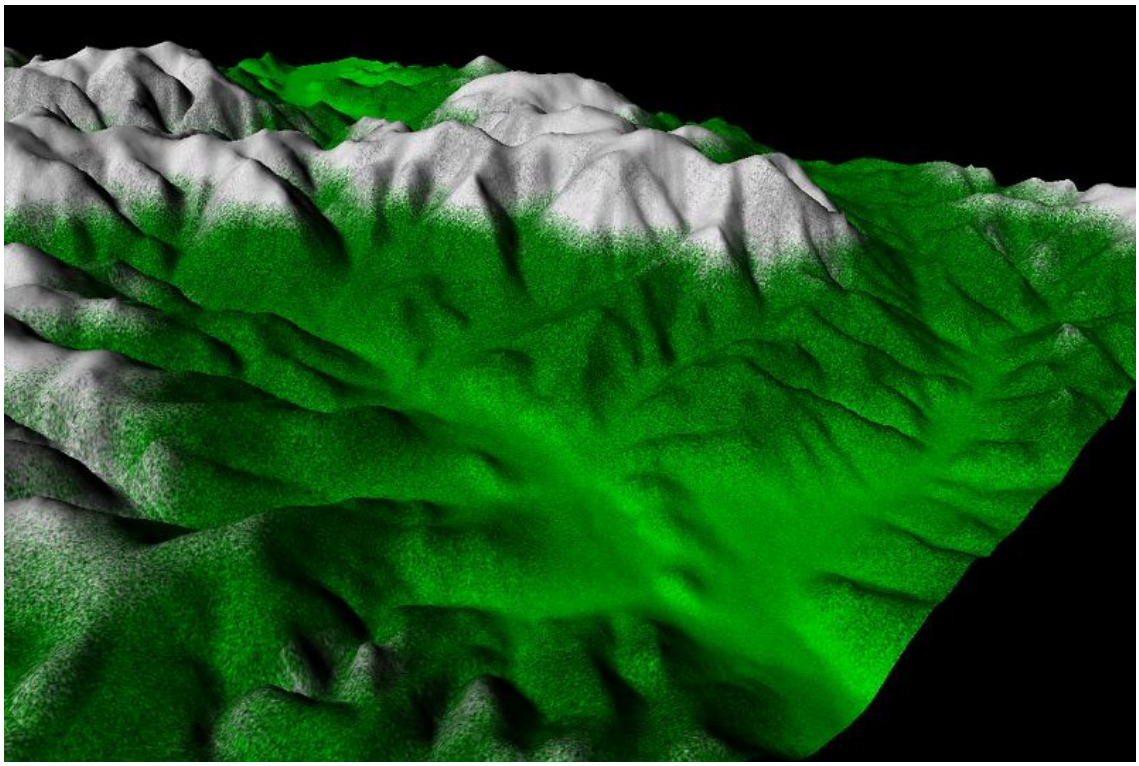
Chceme-li vkládat do textury další objekty, které potřebují další analýzu terénu, musíme procházet znovu matici hodnot a přepisovat hodnoty jednotlivých texelů v již hotové textuře a měnit tak jejich význam. Zde byla snaha vygenerovat řeky. Tento problém není tak jednoduchý, jak by se na první pohled mohlo zdát a taktéž jedna z prací se jím na fakultě zabývá detailněji. I v GIS systémech, kde je podobná analýza podporována, je výpočet říčních toků poměrně výpočetně náročná záležitost. Největším problémem jsou pro nás data – ta jsou navzorkovaná, a proto tvoří třeba dolík i tam, kde ve skutečnosti žádný není. To je jeden z hlavních problémů. Proto nemůžeme využít pouze klasické metody proudění, která pokračuje z aktuálního místa do místa s nejmenší nadmořskou výškou. Musíme ji mírně poupravit – a to tak, že v „dolíku“ budeme vodu „hromadit“.

Na implementaci již zmíněného postupu byl použit buffer o velikosti matice dat, který sloužil k „hromadění“ vody – na políčku, kde měla hladina stoupat, se přičítaly postupně po jednom metry nadmořské výšky a při rozhodování se tyto dvě matice (data a buffer) sčítaly. Protože takových nerovností vzniklých navzorkováním dat je poměrně hodně, nemá smysl v každém dolíku dělat jezero, proto bylo od tvorby jezer upuštěno (je to také tak trošku další problém).

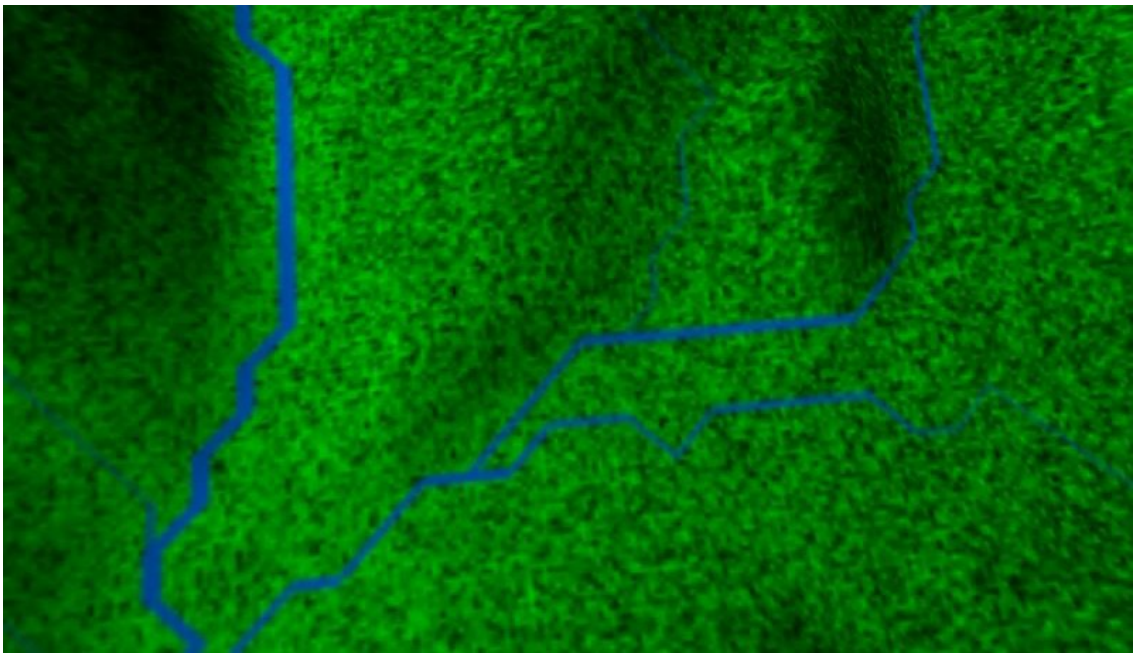
Tímto bychom vyřešili problém s trasou řeky. Dalším problémem zůstává tlušťa řeky a její ohyby. Tloušťka řeky se dá poměrně snadno vyřešit – vzhledem ke skutečnosti, že se řeky vlévají jedna do druhé, mohou v našem algoritmu také jedna druhou přepisovat. Tedy – určíme si, že například po 20 políčcích toku řeky má řeka nárok stát se silnější. Setká-li se pak při generování řek řeka, která teče už 50 políček, s řekou která teče teprve 8 políček a byla tam první, tak nová řeka tu starou prostě přepíše. Obráceně se pochopitelně nestane nic. S ohyby je to už horší – bylo vyzkoušeno několik pokusů, ale žádný nedával uspokojivé výsledky a už vůbec ne rychle. Vzhledem k časové tísni a skutečnosti, že na toto téma jsou řešeny další projekty, které se zabývají detailním generováním řek včetně jejích okrajů a ohybů, zůstal tento problém nedořešen.

Poslední otázkou zůstává, kde vezmeme počátky (prameniště) řek. Odpověď je jednoduchá – tak jako ve skutečnosti, pramení i v našem modelu řeky v lesním pásmu a jejich rozložení je generováno náhodně. Uživatel knihovny má možnost si jejich počet zadat konkrétně. Nezadá-li, knihovna dopočítá vhodné množství dle rozměrů matice dat.

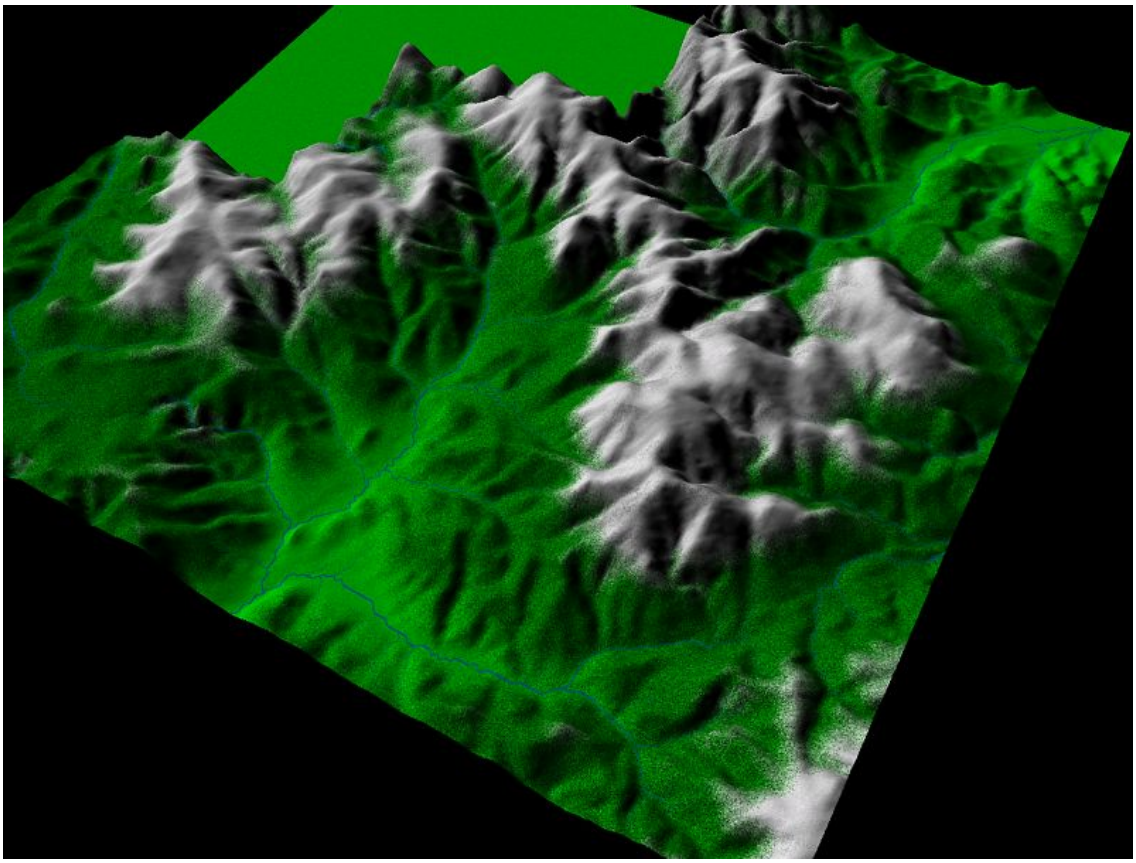
Na závěr pár obrázků – na obr. 5.7 je jasně vidět, jak řeka tvoří ostré záhyby, které při takto blízkém pohledu ruší a působí nepřirozeně. Na obrázku 5.8 je pak záběr na hotovou texturu, která na druhou stranu (bude-li použita např. v nějakém leteckém simulátoru) nepůsobí jako celek zase tak nepřirozeně.



Obrázek 5.6: hotová základní textura



Obrázek 5.7: detail řeky



Obrázek 5.8: hotová textura i s řekami

# Kapitola 6

# Závěr

Projektu se podařilo splnit cíl zobrazení 3D modelu a vygenerování textury s přirozeným vzhledem pro tento model. Zvolená metoda není možná nejšťastnější, ale je třeba vyzkoušet další postupy a následně porovnat reálné výsledky. Pro použití v praxi má tento projekt poměrně vysoké hardwarové nároky, což může být velkou překážkou. Částečně by se daly tyto problémy odstranit, podstata metody však výrazné zlepšení nepředpokládá. Zlepšení by se dalo dosáhnout v kombinaci s jinými metodami – např. dělení modelu na různě podrobné polygony a pro ty generovat různé textury. Docela zajímavé srovnání by bylo zde použité metody s mutlitexturingem.

Co se týká kvalitativního zlepšení, je zde široké pole působnosti, zejména v nedostačujícím řešení vodních toků. Krajina by vypadala podstatně lépe, kdyby se do ní přidaly kromě řek také přehrady a jezera. Vylepšit by se dozajista daly i algoritmy pro generování základního vzhledu textury – jde však o to, co by to provedlo s výslednou rychlostí generování. Je třeba zvolit kompromis mezi vzhledem, rychlostí a účelem, na který bude textura použita. U leteckého simulátoru není třeba tak kvalitní textury jako například u pozemní hry.

Tudíž tento projekt si přímo říká o další vylepšování, zdokonalování (hlavně urychlení) použitých algoritmů a také o začlenění do skupiny dalších, podobně zaměřených projektů. Pro takovéto přímé začlenění by samozřejmě potřeboval možná ještě některé další drobné úpravy, které se však netýkají použitých algoritmů, zmíněných výše, ale spíše obslužných funkcí. Projekt je vhodný k začlenění do skupiny projektů, které se zabývají vkládáním detailů do krajiny (silnice, detailní řeky, budovy, atd.). Byla snaha dát některé lidi dohromady a zkusit větší projekt, bohužel časové možnosti většiny z nás nám neumožnily spolupráci, které by k dosažení výsledků bylo třeba.

# Literatura


[1] Open Inventor tutoriál na ROOT.CZ

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

[3] GameDev.net: http://www.gamedev.net/reference/list.asp?categoryid=45#88

[4] Lighthouse 3D: http://www.lighthouse3d.com/opengl/terrain/

[5] Josie Wernecke: *The Inventor Mentor*, Addison-Wesley Professional, 1994, ISBN: 0201624958

[6] Přemysl Kršek: *Skriptum pro předmět Základy počítačové grafiky*, FIT VUT, 2004

[7] Server nehe.opengl.cz

# Dodatek A

# Uživatelská dokumentace

## A.1 Soubory a souvislosti

Pro použití knihovny jsou nutné dvě věci – mít nainstalovaný vývojový balík Coin3D a vložit hlavičkový soubor `RealisticSurface.h` do zdorojového souboru, kde chcete knihovnu využívat. Vlastní implementace knihovny je v souboru `RealisticSurface.cpp`. V hlavičkovém souboru je pouze deklarace třídy RealisticSurface a definice konstant. Pro ukázku, jak pracovat s knihovnou může posloužit přiložený soubor `Application.cpp`, který obsahuje pouze vytvoření okna, ve kterém se bude scéna zobrazovat, vytvoření samotné scény a ukázkové použití knihovny. Vhodným a snadno nastavitelným překladačem je třeba MS Visual C++ 6.0. Knihovna je psána v jazyce C/C++ a je přeložitelná jak pod Windows, tak pod Linuxem včetně ukázkové aplikace.

## A.2 Použití

Knihovna je implementována jako jedna třída, která obsahuje objekt scény a tento objekt (třídy RealisticSurface) je možné přidat kamokoli do scény – třeba hned do rootu. Vytvoření objektu a jeho přidání do scény může tedy vypadat následovně:

```
RealisticSurface * terrain = new RealisticSurface();
root->addChild(terrain->createTerrain());
```

`root` je kořen scény a funkce `createTerrain()` vrací ukazatel na objekt scény – zde nazývaný `SoSeparator`. Mezi tyto dva příkazy je možné dát volání dalších funkcí, které nastavují vlastnosti modelu, nebo načítají data apod. Následuje přehled všech dostupných funkcí s popisem:

```
SoSeparator * createTerrain( void );
SoSeparator * createTerrain( float displacement_x, float displacement_y );
```

Tato funkce vytvoří objekt (model) výškové mapy a dle nastavení jednotlivých proměnných vygeneruje texturu o 24-bitové barevné hloubce, kterou pokryje výškovou mapu. Nebyli-li hodnoty některých proměnných zadány uživatelem, jsou použity hodnoty implicitní. Proměnné `displacement_x` a `displacement_y` slouží k posunutí celého modelu ve scéně na jinou pozici.

```
DATA* readData(void);
DATA* readData(const char* filename);
```

Tyto dvě modifikace téže funkce slouží pro načtení dat (výškové mapy) ze souboru ve formátu CSV. Pokud se načtení z jakéhokoli důvodu nepovedlo, vrátí funkce `NULL`, jinak vrací ukazatel na načtenou datovou strukturu. Není-li zadán vstupní soubor `filename`, použije se implicitní jméno souboru `data.csv`.

```
void loadData(float* hmap, unsigned long width, unsigned long height);
```

Funkce `loadData()` slouží pro načtení dat z pole – `hmap` je ukazatel na pole (matici) čísel, `width` a `height` jsou šířka a výška matice hodnot, kterou má funkce z toho pole načíst.

```
bool saveData(void);
bool saveData(const char* filename);
```

Zde jsou dvě funkce jako protiklad k funkcím pro načítání dat ze souboru – umí data ukládat z vnitřní datové struktury do souboru ve formátu CSV. Pokud není zadáno jméno `filename` výstupního textového souboru, je vytvořen implicitně soubor pojmenovaný `saveddata.csv`. Funkce vracejí `true`, jestliže se zápis povedl, jinak `false`.

```
void filterData(void);
void filterData(int numberOfPass);
```

Funkce pro vzhlazování dat (filtraci). V proměnné `numberOfPass` je možné zadat počet průchodů. Není-li počet průchodů zadán, je implicitně proveden jeden.

```
int getScale(void);
void setScale(int newScale);
```

Nastavování měřítka se děje pomocí funkce `setScale()`, kde proměnná `newScale` určuje vzdálenost mezi dvěma body ve výškové mapě ve vztahu k hodnotám v této mapě. Tedy byli-li údaje pořízeny např. odečtem z mapy vzorkováním po 100 metrech a jsou také uvedeny v metrech, pak je měřítko 100. Funkce `getScale()` pouze vrátí aktuální hodnotu měřítka.

```
void setNumberOfRivers( int number );
void setLowlandsLimit( double number );
void setForestsLimit( double number );
void setLowlandsGradient( double number );
void setHighlandsGradient( double number );
void setWinterMode( bool value );
void setTextureResolution( int resolution );
void setHeightSpread( double number );
```

Tato skupina funkcí nastavuje vlastnosti textury. První funkce nastaví počet generovaných řek, implicitně je počet řek dopočítán vhodně dle rozměrů matice dat. `lowlandsLimit` a `forestsLimit` udává nadmořskou výšku, do které se mají vyskytovat louky a lesy. Pokud nejsou tyto hodnoty zadány, jsou dopočítány dle vlastností výškové mapy (podle rozdílu nejmenší a největší nadmořské výšky). `lowlandsGradient` a `highlandsGradient` jsou limitní sklony svahů pro tvorbu skal. Hodnoty těchto proměnných se zadávají ve stupních. `winterMode` rozhoduje o módu vykreslování

– `true` znamená, že krajina bude vypadat jako zimní, při použití `false` bude krajina vypadat jako v létě. Rozlišení textury (počet texelů) lze nastavit funkcí `setTextureResolution()`. Proměnná `heightSpread` určuje rozmezí, ve kterém bude jeden pás přecházet v druhý – to znamená, že pokud nastavíme tuto hodnotu třeba 300, pak bude třeba les přecházet ve vysokohorskou krajinu v rozmezí 300 výškových metrů – např. od 1000 do 1300 m. n. m.

## A.3 Konkrétní příklad

Na závěr si můžeme uvést konkrétní příklad použití:

```
RealisticSurface * terrain = new RealisticSurface();

if(terrain->readData( "data2.csv" ) == NULL){
  exit(1);
}

terrain->setNumberOfRivers( 200 );
terrain->setLowlandsLimit( 500 );
terrain->setForestsLimit( 900 );
terrain->setHighlandsGradient( 30 );
terrain->setWinterMode( false );
terrain->setScale( 100 );
terrain->setTextureResolution( 3000000 );
terrain->setHeightSpread( 150 );
terrain->filterData( 4 );

root->addChild(terrain->createTerrain());
```