VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

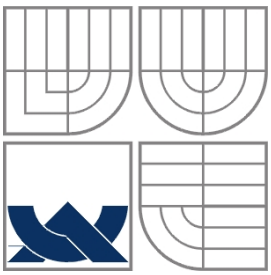
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# VIZUALIZACE TEKOUCÍ VODY V KRAJINĚ

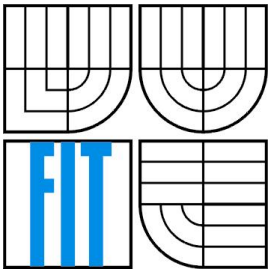
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE ADAM VLČEK
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# VIZUALIZACE TEKOUCÍ VODY V KRAJINĚ

WATER FLOWING IN THE LANDSCAPE VISUALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

ADAM VLČEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL SEEMAN

BRNO 2007

*Zadání bakalářské práce/5701/2006/xvlcek12*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

# Zadání bakalářské práce

Řešitel: **Vlček Adam**
Obor: Informační technologie
Téma: **Vizualizace tekoucí vody v krajině**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte dostupnou literaturu. Prostudujte běžně používané metody v současnosti.
2. Navrhněte postup pro realistickou vizualizaci toku vody a pro detekci vzniku jezer. Navrhněte další možné optimalizace.
3. Implementujte algoritmus zobrazení krajiny v reálném čase.
4. Diskutujte výsledky. Diskutujte možnosti dalšího rozšíření aplikace, např. pro simulaci eroze nebo povodní.

Literatura:

- Žára, J., Beneš, B., Felker, P.: Moderní počítačová grafika, Computer Press, 1998.
- Dle pokynu vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Seeman Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
*vedoucí ústavu*

**LICENČNÍ SMLOUVA**
**POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO**

uzavřená mezi smluvními stranami

**1. Pan**

Jméno a příjmení: **Adam Vlček**
Id studenta: 80400
Bytem: Šumická 669, 664 07 Pozořice
Narozen: 02. 08. 1983, Brno
(dále jen "autor")

a

**2. Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

**Článek 1**
**Specifikace školního díla**

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Vizualizace tekoucí vody v krajině
Vedoucí/školitel VŠKP: Seeman Michal, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

## Článek 2
## Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

## Článek 3
## Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

..... .....

Nabyvatel Autor

Originály zadání a licenční smlouvy jsou vevázány v originále bakalářské práce.

# Abstrakt

Tato práce se zabývá vizualizací tekoucí vody v krajině se zaměřením na zobrazování výsledků v reálném čase. Za účelem získávání dat pro vizualizaci jsou zde probrány i základní postupy využívané při generování umělého terénu a přírodně vypadajících textur, stejně jako algoritmy simulující přelévání vody po podkladu. Cílem je tak vytvoření interaktivního prostředí, které bude schopno poskytovat zajímavý vizuální výstup v přímé reakci na uživatele.



# Klíčová slova

vizualizace terénu a vody, simulace pohybu vody, generování terénu, fraktály, OpenGL, GLSL



# Abstract

This work covers visualization of water flowing in the landscape with focus on displaying the results in real time. For in order to obtain data for visualization also common procedures for artificial terrain and natural looking textures are introduced, as well as simple water flow simulation algorithms. The final objective of this work is to present an interactive environment, which will offer interesting visual output in direct response to the user.



# Keywords

terrain and water visualization, water flow simulation, terrain generating, fractals, OpenGL, GLSL



# Citace

Adam Vlček: Vizualizace tekoucí vody v krajině, bakalářská práce, Brno, FIT VUT v Brně, 2007

# Vizualizace tekoucí vody v krajině

## Prohlášení

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

.....
Adam Vlček

## Poděkování

Děkuji svému vedoucímu Ing. Michalu Seemanovi za poskytnuté rady a podporu.

# Obsah

# 1 Úvod

Cílem této práce je navrhnout a následně realizovat jednoduchý simulátor pohybu vody v krajině se zaměřením na získávání kvalitních vizuálních výsledků v reálném čase. Projekt tak zasahuje hned do několika zajímavých oblastí.

Aby bylo vůbec možno něco zobrazit, musíme nejdříve získat zobrazovaná data. V našem případě jsou nejdůležitějšími daty údaje o tvaru terénu, která je v ideálním případě možno načíst ze souboru dat získaných měřením. Mnohdy může být ale snazší a zajímavější různými algoritmy terén vygenerovat, protože ne vždy jsou k dispozici patřičná data v dostatečném rozlišení. Nezanedbatelnou vlastností pak je možnost terén následně snadno upravovat přímo v programu. O terénu detailně pojednává druhá kapitola.

Krom terénu je potřeba nějak zjistit i informace o rozložení vody na něm. Samozřejmě je možné opět načíst již dříve získané hodnoty, ale zvlášť pokud nechceme pouze sledovat statickou scénu, jsou taková data obsáhlá a velmi obtížně se získávají. Proto je jedním z cílů tohoto projektu i implementovat algoritmus simulující přesun vody v důsledku gravitace. Pro téma vody je tak vyhrazena vlastní kapitola s číslem tři.

Nejlépe pak lze dosažení působivých vizuálních výsledků za použít trojrozměrného zobrazení, jehož silná podpora je v dnešní době již nedílnou součástí každého moderního počítače. Takovéto zobrazení skýtá velmi dobrou představu o reálném tvaru terénu i pohybující se vodě a zároveň je možno dosáhnout velmi vysoké kvality obrazu. Obecným technikám používaných pro zkvalitnění zážitku z 3D grafiky renderované v reálném čase je tak věnována kapitola čtvrtá.

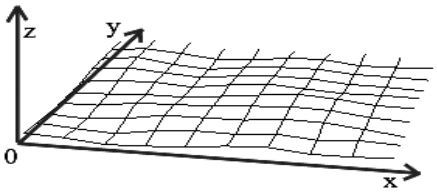
Jelikož tato práce v různých místech zasahuje do značného množství rozsáhlých oblastí, nemůže zde být vše zpracováno příliš detailně, a tak se snažím v každé kapitole vždy uvést do problému, zmínit jeho obvyklá řešení, následně se blíže rozepsat o použité variantě a na závěr zmínit potenciální vylepšení a navrhnout další vývoj.

# 2 Terén

Jak plyne již z úvodu, terén je základem celého projektu. Jednak tvoří nezanedbatelnou část vizuálního dojmu ze scény, hlavně ale slouží jako podklad pro vrstvu vody, která po něm teče. Proto je velmi důležité zvolit vhodnou reprezentaci, která bude z hlediska rychlosti a kvality vyhovovat zároveň pro zobrazování i pro simulaci pohybu vody.

## 2.1 Reprezentace terénu v programu

Existuje mnoho způsobů, jak s terénem v programu pracovat, ale ne každý je vhodný pro výpočty v reálném čase. Asi nejpřesnější bývají vektorové modely, které ovšem za svou přesnost platí velkou výpočetní náročností a obtížnou implementací. Pro naše účely je jednoznačně lepší rastrová reprezentace, v rámci které se ještě musíme rozhodnout mezi dvěma variantami: trojrozměrnými *voxely* a dvourozměrnými *pixely*. Je zřejmé, že trojrozměrná varianta je výpočetně náročnější než dvourozměrná, přičemž rozdíl v kvalitě výstupu je ve většině základních situací poměrně malý, a tak nám dvourozměrná varianta vychází vítězně, stejně jako v [FER07]. Je třeba si ale uvědomit, že ve složitějších situacích už tento model neobstojí a na převisy a jeskyně můžeme zapomenout.




*Obrázek 2.1: Drátěný model terénu v kartézském souřadném systému*

Rastr terénu je tedy jakási čtvercová síť, kde každý bod obsahuje informaci o své výšce, která je do určité míry platná i pro jeho okolí a pro plynulost jsou tyto hodnoty interpolovány, aby se zamezilo vzniku ostrých hran mezi body. Obrázek 2.1 zobrazuje drátěný model vytvořený s použitím lineární interpolace.

### 2.1.1 Rozdělení na obdélníky

Rastrová reprezentace ve své základní verzi má jednu velkou nevýhodu – pokrývá celou plochu stejně hustě, aniž by zohledňovala důležitost nebo prostorovou složitost konkrétních oblastí a jim věnovaný výpočetní výkon. Proto je systém navržen tak, aby bylo možno celý terén rozdělit do více obdélníků o různé hustotě bodů.

Za tímto účelem je pole vrcholů v každém obdélníku řešeno pomocí speciálního objektu, který zprostředkovává přístup k jednotlivým bodům obdélníku. To plyne z potřeby rozlišovat funkci vnitřních bodů od bodů okrajových, které se sice navenek tváří všechny stejně, ale za pomoci dědičnosti a virtuálních metod jsou implementovány rozdílně.

Díky tomuto rysu, znázorněnému na obrázku 2.2, je možné pracovat s celým dvourozměrným polem při většině operací mnohem snáze, protože i když je možno bez problémů pracovat s prvky o indexech každého rozměru v rozsahu *0* až *X*, reálná data jsou uložena jen na indexech *1* až *X-1*. Na indexech *0* a *X* se pak nacházejí body, které zprostředkovávají návaznost na okolí a jejich hodnoty jsou tak počítány na základě údajů z vrcholů náležících do okolních obdélníků.

### 2.1.2 Vnitřní body

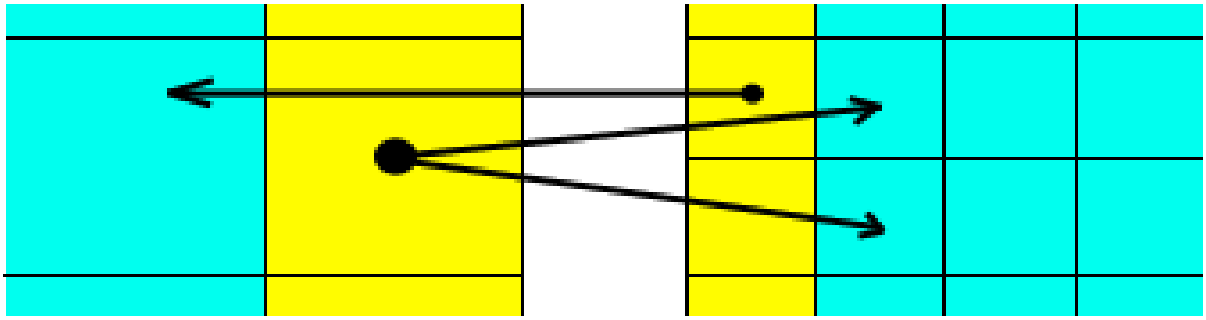
Vnitřní body obdélníku tvoří základ celého systému. Právě tyto body obsahují data o terénu a po něm se valící vodě, kterými jsou informace o nadmořské výšce terénu, výška vodního sloupce a rychlosti přelévání vody do nejbližších osmi sousedů. Tyto body jsou zároveň i buňkami celulárního automatu, který provádí simulaci přesunu vody v důsledku gravitace, čemuž je věnována vlastní kapitola.

Při budoucím vývoji právě sem budou přibývat údaje o vrstvách terénu, množství vodou přenášeného materiálu a další potřebné údaje.

### 2.1.3 Okrajové body

Aby bylo možno jednoduše navazovat více obdélníků s různými hustotami bodů, obsahuje každý obdélník na svých okrajích jednu řadu těchto speciálních bodů, které se sice navenek chovají stejně jako vnitřní, ale jejich podstata je naprosto odlišná. Svým umístěním jakoby přesahují rozměry svého obdélníku a zasahují tak do prostoru sousedních. Samy o sobě neobsahují žádná data, pouze seznam vrcholů ze sousedního obdélníku, se kterými se překrývají. Na základě různých vah těchto vrcholů pak jsou počítány hodnoty jako nadmořská výška terénu nebo vodní sloupec a při přelévání vody je její množství opět na základě vah rozděleno.

Jedinou nevýhodou tohoto návrhu je, že výpočty prováděné v těchto bodech jsou časově náročnější, takže je dobré se snažit jejich počet omezit. Jsou možné i optimalizace, kdy se pro každý simulační krok nejdříve vypočítají hodnoty do proměnných a v průběhu výpočtu kroku se pak pracuje jen s nimi.



*Obrázek 2.2: Žluté okrajové body obsahují seznam odpovídajících vnitřních modrých bodů v sousedním obdélníku. V prostoru se pak okrajové body částečně kryjí se svými partnery.*

## 2.2 Získávání terénu

Pro kvalitní zobrazení a simulaci musí být kvalitní podklady. Nejjednodušší a nejrychlejší je kvalitní data načíst ze souboru, ale dostat se k takovým souborům nemusí být lehké, a proto je vhodné umět si data obstarat i jinak. Naštěstí existují i postupy, jak terén více či méně náhodně vygenerovat.

### 2.2.1 Načítání souborů

Pro praktické využití v reálných aplikacích je samozřejmě nutné, aby program byl schopen načíst data získaná na základě reálných měření. Ta mohou být provedena například klasickými geodetickými postupy, skenováním povrchu pomocí družice nebo třeba v dnešní době i amatérsky pomocí zaznamenávání nadmořských výšek a souřadnic pomocí GPS. Získaná data pak bývají uložena v různých formátech, jejichž načítání může být více či méně složité. Pro naše účely naprosto dostačuje, když je program schopen načíst výškové pole ve formě obrázku, kde intenzita barev reprezentuje nadmořskou výšku. Takto uložená data mají za cenu omezené přesnosti, která pro nás není příliš podstatná, několik výhod. Tou hlavní je bez pochyb schopnost již zavedených geografických informačních systémů exportovat svá data do běžných obrázkových bitmap, stejně jako možnost jejich vytváření a upravování v kterémkoli grafickém editoru. Nevýhodou pak je absence údaje o rozsahu výškových hodnot, kterou je nutno zadat jinak.

Pro snadnou práci pak jistě není na škodu, když program umí svůj aktuální stav uložit do souboru a později ho opět načíst. Jelikož funkce tohoto programu je poměrně specifická, je zbytečné uvažovat o nějakém standardním formátu souboru, a proto je nutno navrhnout vlastní.

Nejjednodušší variantou je soubor, který bude obsahovat hlavičku s podpisem programu a jeho verzí, následovanou počtem uložených obdélníků. Zbytek pak bude pole obdélníků, přičemž každý záznam bude obsahovat parametry jako velikost bodů, rozměr rastru a pole bodů.

Takovýto formát je sice velmi jednoduchý pro implementaci, avšak při rozšiřování programu vyvstává nutnost řešit různé verze souborů a většinou je i úplně odlišně načítat. Jako lepší varianta se pak jeví složitější implementace obecnějšího formátu založeného na blocích, které obsahují identifikaci svého druhu, velikost v bytech a samotný obsah. Pak je možno při přeskakování neznámých bloků načítat dokonce i novější verze souborů, což je při primitivní implementaci prakticky vyloučeno.

Pokročilejší verze souboru by pak obsahovala hierarchii bloků, kde prvním by byla opět hlavička s podpisem a verzí programu, následovaná bloky obdélníků. Ty by pak obsahovaly bloky s jednotlivými druhy údajů (vrstvami) pro body rastru s pouze jedením druhem informace, aby si načítající program mohl snáze vybrat, co dokáže načíst a co raději přeskočit. Tak vzniknou například bloky s nadmořskou výškou terénu, hloubkou vody nebo s údaji o jejím pohybu.

### 2.2.2 Generování umělého terénu

Získávání vlastních reálných dat je obvykle velmi zdlouhavé a nákladné. V praxi tyto náklady mnohonásobně přesahují cenu za vývoj zpracovávajícího software. Stejně tak je nepraktické kreslit si pro každou příležitost mapy ručně v grafickém editoru a pak je následně importovat do programu. Z toho důvodu bylo implementováno několik algoritmů, které se snaží generovat vizuálně zajímavý terén automaticky. Problematika je to velmi obsáhlá a složitá, a tak se zde zabývám jen několika nejzákladnějšími metodami, které se používají například v počítačových hrách. Ve všech případech se dá vysledovat fraktálová povaha těchto algoritmů, jejich výsledky při různé úrovni detailů jsou zachyceny na obrázku 2.3. Základnímu principu jednoduchých fraktálových terénů se věnuje například [MAR02].

Zájemci o tuto problematiku jistě znají například velmi dobrý software jménem *Terragen [TER07]*, který obsahuje kromě pokročilých terén generujících algoritmů i výbornou vizualizaci, avšak jeho zaměření má k renderování v reálném čase značně daleko.

#### 2.2.2.1 Náhodné body

Podstatou prvního zde zmíněného algoritmu je vygenerování databáze velkého počtu bodů s náhodnými souřadnicemi, nadmořskou výškou a mohutností. Aby nedocházelo k výrazným změnám v charakteru terénu u jeho okrajů, je potřeba tyto body generovat i v určité vzdálenosti mimo terén, která by měla odpovídat maximální mohutnosti bodu. Pro každý bod rastru se pak spočítají vlivy všech vygenerovaných bodů na základě jejich vzdálenosti od bodu rastru. Na použité funkci velmi záleží výsledný tvar terénu i rychlost generování, která bohužel není příliš velká, avšak pro rozměry terénu zpracovávané v reálném čase naprosto dostačuje. Výsledný terén je zaoblený s mnoha kopci a jezery, takže základním požadavkům na simulaci tekoucí vody dobře vyhovuje, nicméně jeho realističnost je pouze omezená a není možno počítat s tvorbou údolí a koryt řek.

#### 2.2.2.2 Kreslení přímek

O něco reálněji působí na pohled metoda, jejíž podstatou je kreslení velkého množství různě širokých a hlubokých přímek přes rastr terénu. I když tato metoda rovněž nepatří mezi nejrychlejší, vygenerovaný výsledek vypadá reálněji než u metody první, protože přímky mají spíše tendenci tvořit hřebeny a údolí, nicméně k reálnému terénu má stále velmi daleko, protože neexistuje žádný princip, podle kterého by se tvořila koryta řek. Kromě toho má takto generovaný terén tendenci obsahovat mnoho ostrých předělů, které vytváří při simulaci velká množství malých jezírek, která bohužel nevypadají příliš dobře. Zmíněný neduh lze odstranit dodatečným vyhlazením.

#### 2.2.2.3 Perlin noise

Velmi oblíbeným algoritmem pro generování náhodných přírodních útvarů je *Perlin noise*. Tento fraktálový algoritmus je založený na pseudonáhodném generátoru náhodných čísel, který pro danou vstupní hodnotu vrací pseudonáhodné číslo, které je však při opakovaném volání se stejnými parametry vždy stejné. Pomocí něj pak můžeme získat obecně n-rozměrný šum, který je v několika oktávách s náležitou persistencí skládán do výsledné křivky. Mezi výhody algoritmu patří oproti výše zmíněným vysoká rychlost, poměrně dobré výsledky a široké spektrum použití. K opravdu reálnému terénu má ovšem stále daleko. Více o algoritmu Perlin noise lze nalézt například v [PER99].

#### 2.2.2.4 Další možnosti

Existuje samozřejmě mnoho dalších více či méně dobrých a rychlých algoritmů. Mezi nimi jistě vynikají ty, které lépe využívají vlastností fraktálů. Je například možné už při generování terénu zohlednit jeho formující faktory, jako je zejména vodní eroze. Jednou z těchto metod je *growing*, který spočívá ve vygenerování počátečních bodů s určitými parametry, ze kterých pak terén na základě pravidel roste do prostoru. Největší problém je zde s navazováním lokací vygenerovaných z více bodů.

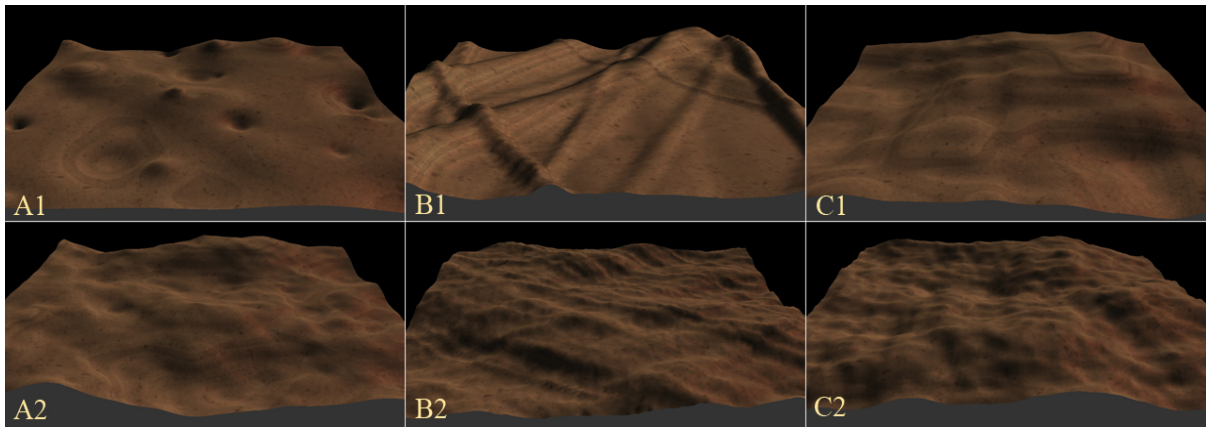
Na závěr musím zmínit, že zde zmíněné algoritmy se zdaleka neváží pouze na generování terénu. Lze jimi vytvářet například realistické mraky, textury kamenů, dřevo nebo vlnky na vodě. Právě textura vlnek pro shader vody je generována pomocí Perlin noise ve třech rozměrech. Celé téma generování realistických kusů přírody je však natolik obsáhlé, že by na něm bylo možno postavit celou diplomovou práci, a tak mi nezbývá, než ho pro tuto chvíli opustit.

#### 2.2.2.5 Dodatečné úpravy

Po načtení nebo vygenerování terénu je možno výsledek vyhladit jednoduchým filtrem, který může odstranit některé nežádoucí artefakty a terén tak učinit vhodnější pro simulaci toku vody v celulárním automatu. Filtr funguje na základě klasické matice 3x3 bodů, z nichž se vypočítá průměr výšek, který se ukládá do nového rastru, aby nebyly ovlivněny výpočty následujících bodů. Vzhledem k reprezentaci okrajových bodů terénu pomocí odkazů na existující body není nutno řešit problém s neznámými hodnotami za hranicemi rastru, což vyhlazovací algoritmus ještě více zjednodušuje.

Podobným způsobem je samozřejmě možno implementovat i mnoho dalších klasických filtrů, jako je například zvýraznění hran, ale jejich praktický význam je pro práci s terénem minimální. Stejně tak je samozřejmě použít i matici o jiných rozměrech.

Mezi další úpravy lze počítat i manipulaci s terénem na základě příkazu uživatele. Pomocí kruhového kurzoru lze manipulovat s výškou terénu a vyhlazovat nějakou jeho část. Tyto operace zasahují větší plochu a je potřeba zajistit návaznost na nezasažené okolí. Proto se účinek operace směrem od středu kurzoru postupně snižuje, až se na kruhem vyznačené hranici ztrácí úplně, a tak vytváří při editaci plynulé přechody. Velmi vhodná se pro tyto účely jeví sinová křivka, která má maximum ve středu kruhu a minimum na jeho okraji, ale lze využít i lineární funkci.




*Obrázek 2.3: Řada A1,B1,C1 ukazuje metody bodů, přímek a Perlin noise při nízkých detailech. Řada A2, B2, C2 tytéž metody při detailech dostatečných. Nejdramatičtější zvýšení výpočetní náročnosti vykazuje metoda bodů, průměrně pak vychází kreslení přímek a nejlepšího poměru kvality ku výpočetnímu času dosahuje Perlin noise.*

## 2.3 Vizualizace terénu

Terén je nedílnou součástí scény, sám zabírá její podstatnou část a utváří do značné míry prostředí pro vodu. Z toho důvodu je zapotřebí ho zobrazovat co nejlépe.

### 2.3.1 Vrcholy a jejich uspořádání

Terén je reprezentován dvourozměrným rastrem bodů, a tak se jeví jako nejlepší řešení ho tak i zobrazit. V běžné 2D grafice reprezentuje hodnota bodu rastru jeho střed a je platná pro celé jeho okolí. Kdyby se takto zobrazovalo i zde, byl by výsledek plný ostrých hran, které nepůsobí ani trochu

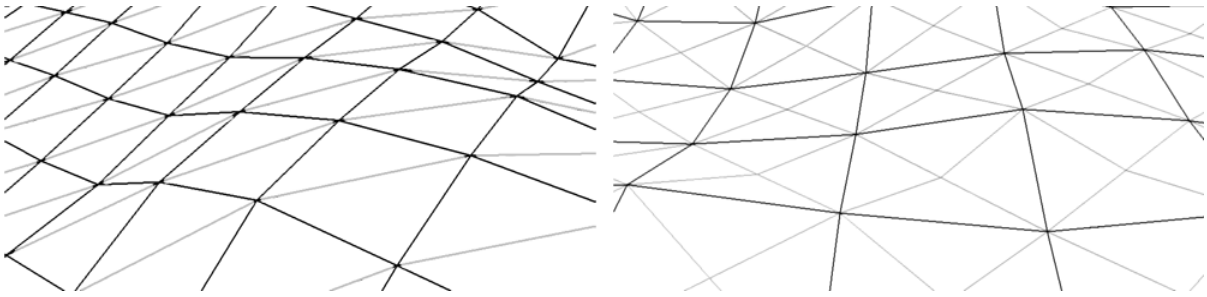
reálně. Proto jsou body rastru brány jako vrcholy plošek a hodnoty sousedních vrcholů jsou tak lineárně interpolovány.

Jelikož má rastr obecně obdélníkové body, zdálo by se jako ideální jeho body v 3D rovněž spojit obdélníky, přičemž OpenGL dokonce i zadávání takovýchto grafických primitiv přímo umožňuje. Bohužel však čtyři body téměř nikdy neleží v rovině, a tak je nutno pro jejich spojení použít trojúhelníky, kde je již tato základní podmínka 3D grafiky splněna. Vyvstává tím ale problém, jak tyto trojúhelníky do obdélníku uspořádat.

Nejjednodušší a nejrychlejší variantou je prostě obdélník rozdělit na dva pravoúhlé trojúhelníky. Takovéto zobrazení naprosto dostačuje pro běžné aplikace, jako je třeba většina počítačových her pracujících s rastrovým terénem, bohužel to ale nestačí zde. Důvodem je simulace toku vody a její zobrazení na terénu. Pokud je totiž terén složen z výše popsané sítě pravoúhlých trojúhelníků, bývá problém s diagonálním spojením dvou společných bodů, které tvoří v jednom směru určitou nepravidelnost, která se při zobrazení druhé vrstvy, kterou je voda, ještě více zvýrazní. Stejně tak toto zobrazení nerespektuje ani simulaci, která přelévá vodu i přes jím tvořené virtuální hřebeny. Proto je nutno rozdělit obdélníky lépe.

Pro naše účely je tedy nutno přijít se zobrazením, které je symetrické a respektuje pohyb vody. Symetričnost lze zajistit přidáním středového bodu obdélníku, okolo kterého pak vykreslíme čtyři trojúhelníky. Nadmořskou výšku samozřejmě můžeme vypočítat jako průměr všech čtyř okolních bodů. Vizuální výsledek pak je nejkvalitnější, ale stále nerespektuje simulované proudění vody. Za tímto účelem je nutno vybrat níže položenou diagonálu obdélníku a výšky jejích konců zprůměrovat. Rozdíl mezi těmito variantami demonstruje obrázek 2.4.

Samotné zobrazení v OpenGL pak lze provést více způsoby. Základním je volání funkce `glBegin`, a následná konstrukce trojúhelníků voláním mnoha různých funkcí, což je zbytečné zdržení. OpenGL podporuje i funkce, kterým je možno zadat potřebné údaje ve formě pole, takže se omezí předávání parametrů a některé operace mohou být lépe optimalizovány. Pokud by navíc informace byly statické, je možné takové pole nahrát přímo do paměti karty pouze jednou a celou operaci ještě více urychlit. V našem případě však počítáme se značně dynamickými daty, a tak musíme některé z nejrychlejších technik opustit nebo je využívat jen omezeně.



*Obrázek 2.4: Vlevo drátěný model terénu při použití nejjednodušší metody, vpravo kvalitnější varianta. Přídavné hrany dělících trojúhelníků jsou vyznačeny šedě, černou je základní síť terénu.*

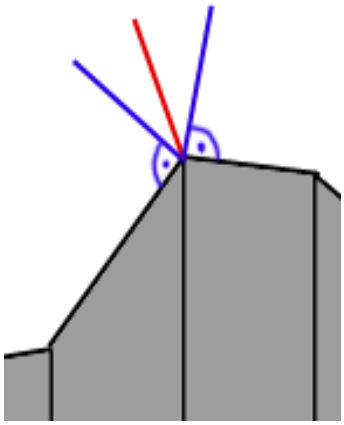
#### 2.3.1.1 Výpočty normál

Jedním z velmi důležitých údajů, které je nutno nějakou formou přiřadit k vykreslovanému vrcholu je

normála. Jedná se o normalizovaný vektor, který směřuje kolmo od roviny vykreslované plochy. Jelikož OpenGL pracuje pouze s vrcholy, je nutno tuto plochu nějak definovat v programu a z ní pak normálu odvodit.

Nejpodstatnějšího využití se normály dočkají při osvětlovacích operacích, které OpenGL provádí buď pro celé jedno grafické primitivum (čára, trojúhelník,...) nebo pro každý jeho vrchol a hodnoty pak interpoluje. Jelikož nám jde o kvalitu, musíme zadávat normály pro všechny vrcholy.

Výpočet normály pro jeden bod probíhá tak, že nejprve vypočítáme pomocí vektorových součinů vektory kolmé k plochám, které procházejí okolními trojúhelníky, jejichž je náš bod společným vrcholem. Normalizované varianty těchto vrcholů pak sečteme a výsledek pro OpenGL opět normalizujeme, případně je možné zapnout jejich automatickou normalizaci až v hardware, což může být podstatně rychlejší.



*Obrázek 2.5: Modře jsou normály jednotlivých plošek, červeně jejich normalizovaný součet.*

S využitím shaderů je možno provést v hardware celý výpočet normál pro rastrový terén. Jednou z variant je zadat pro každý vrchol i okolní výšky bodů a výpočet provést ve vertex shaderu, efektivnější bude předat výškové pole jako texturu a po projití fragment shaderem výsledek získat opět ve formě textury, kterou je možno ve shaderech ihned znovu použít.

Běžně jsou normály pro objekt vypočítány jen jednou, do hry však opět vstupuje dynamičnost scény, která nás nutí normály počítat v každém kroku znovu, což je značně výpočetně náročné a v jiných aplikacích se běžně neprovádí.

### 2.3.2 Úrovně detailů a další techniky urychlování vykreslování

Za normálních okolností se pro zobrazení terénu s výhodou používá techniky úrovní detailů (LOD), která je založena na faktu, že vzdálený terén není nutno vykreslovat v plné kvalitě, protože nás obvykle zajímá spíš dění před námi a vzdálené detailní informace v obraze splývají. Problém je opět dynamičnost našeho modelu, a tak by bylo nutné vytvářet i úrovně detailů dynamicky, což může být rovněž výpočetně náročné. Ještě závažněji vadí, že se zde pracuje se dvěma mnohdy velmi blízkými vrstvami, kde i nepatrná změna ve tvaru jedné z nich může mít velmi značný vliv na výsledné zobrazení. Z těchto důvodů jsem prozatím od omezování detailů upustil. Jako schůdnější se jeví opačný přístup, kdy je možno blízké body proložit křivkou a dosáhnout tak vyšších detailů.

### 2.3.3 Multitextury, texturové souřadnice

Protože kreslení jednotlivých trojúhelníků je poměrně náročné a jejich množství je omezené, byla vytvořena technika nazývaná texturování. Díky ní je možno na každý trojúhelník jakoby nalepit tapetu, která je pak zobrazena s náležitou perspektivou, čímž se dosáhne výrazně lepší vizuální kvality s minimálními náklady na výpočty.

Vzhledem k tomu, že terén je z pohledu reprezentace v programu homogenní, je nutné zajistit, aby se tak nejevil i pozorovateli. Z toho důvodu je na něj mapováno hned několik textur zároveň za použití techniky multitexturingu. Základem zobrazení je textura, která se nanáší na celý obdélník terénu a modeluje tak základní velké nerovnosti a změny ve složení. Aby nedocházelo k přílišnému

rozmazání povrchu při velkém přiblížení, je na každý obdélník mezi body aplikována další opakující se textura, která přidává detaily o velikosti kamení a písku. Pro zlepšení vizuálního dojmu je navíc přidána jednorozměrná textura simulující vrstvy terénu.

Aby se textury správně zobrazovaly, je potřeba předat do OpenGL informace o jejich mapování na vrcholy. Tyto informace lze zadat explicitně pro každý vrchol, jelikož jsou ale v našem případě souřadnice velmi jednoduché a navíc vázané na polohu vrcholů v prostoru, naskýtá se zde dobrá příležitost pro jejich automatické generování. Zejména při multitexturingu tak opět dojde k výraznému omezení volání funkcí a množství předávaných dat. Stačí OpenGL zadat vhodnou transformační matici, která se v tomto případě skládá pouze z koeficientů, kterými se násobí prostorové souřadnice renderovaných vrcholů.

## 2.4 Možnosti rozšíření a dalšího vývoje

### 2.4.1 Vrstvy terénu

Pokud bude někdy nutno implementovat realistickou simulaci eroze, náplav nebo třeba tvorby jeskyní, nezbude než přidat reprezentaci terénu třetí rozměr. Spíše než k vytvoření homogenního 3D rastru ve tvaru kvádru se přikláním k variantě, kdy každý bod bude mít jen tolik dat, kolik bude nezbytně nutné. Data vždy budou začínat v úrovni povrchu, přičemž uchovávána bude i tloušťka vrchního voxelu, ze kterého bude odebírán materiál při erozi a kam budou také ukládány náplavy. Pokud tloušťka tohoto voxelu přeroste určitou hranici, bude uložen a na něm vytvořen nový. Pokud naopak dojde k úplnému odplavení, voxel zanikne a na povrch se propracuje následující. Nejhlubší voxel pak bude mít pro odplavování nekonečnou hloubku a vždy se na něm vytvoří jeden navíc, se kterým bude systém pracovat..

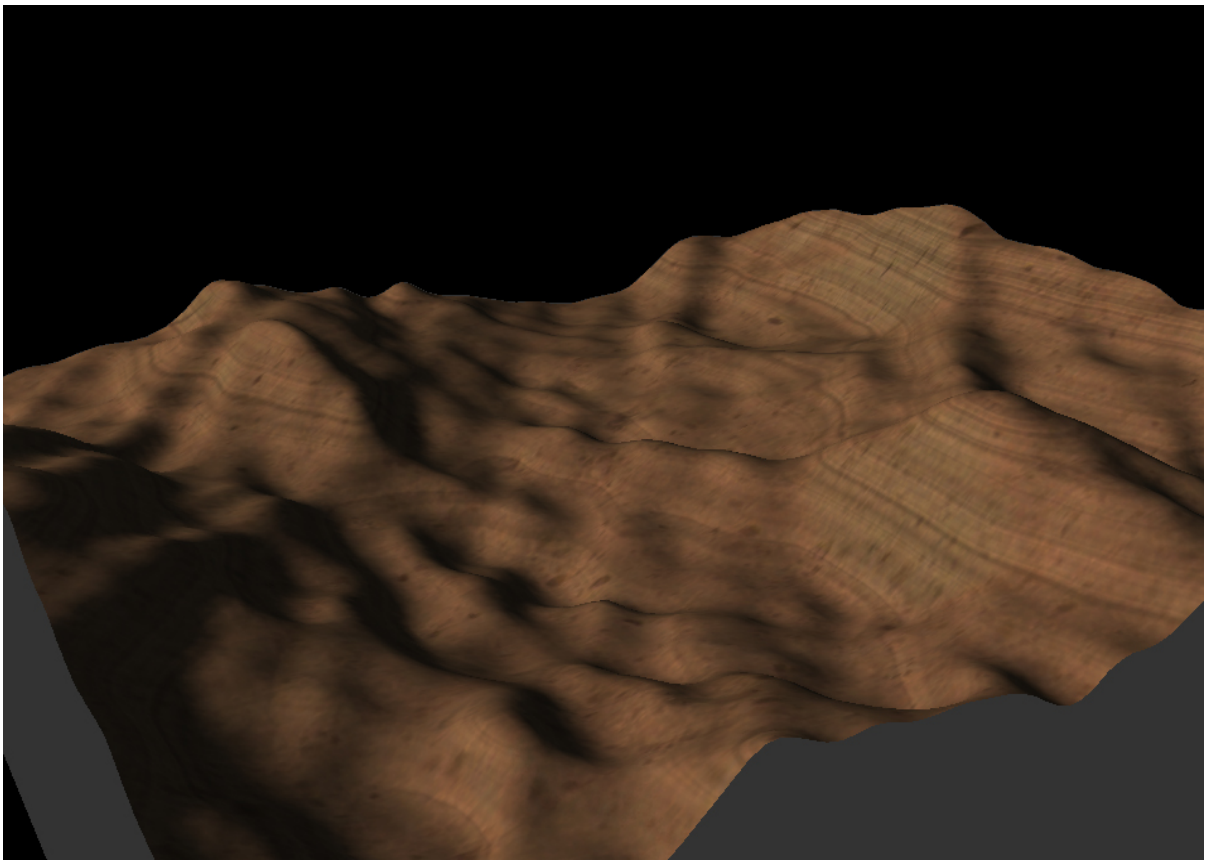
Pokud bude vyžadována i simulace podpovrchové vody a tvorby jeskyní, bude nutno ukládat údaj o zaplnění voxelu pro každý zvlášť. Kromě změny reprezentace terénu pak budou takovéto výpočty možné pouze po naprostém přepracování simulátoru.

### 2.4.2 Využití shaderů a další vylepšení

Do budoucna se počítá s využitím shaderů pro lepší zobrazení drobných nerovností, kde své využití najdou zejména normálové mapy pro *bump mapping* a *paralax mapping*, které jsou schopné při relativně nízkých nárocích vytvořit v rámci plošky iluzi trojrozměrnosti. První upravuje osvětlení na základě výpočtů po bodech se zohledněním normál v textuře, druhý na základě normál posouvá mapování textury, která se pak z různých úhlů jeví různě deformovaná.

Kromě samotných shaderů by jistě bylo zajímavé implementovat různé druhy povrchu, kde by mimo jiné bylo možno shaderů využít k samostatnému mixování textur na základě několika informací renderovaném vrcholu.

Nejlepší kvality by pak bylo možno dosáhnout přidáním vegetace a dalších detailů. V dnešní době již existují postupy, které jsou schopné vytvořit celkem přesvědčivou iluzi trávy i vzrostlých stromů. V této oblasti se poslední dobou prosazuje middleware SpeedTree, který je schopen vykreslovat rozsáhlé zalesněné oblasti v reálném čase.

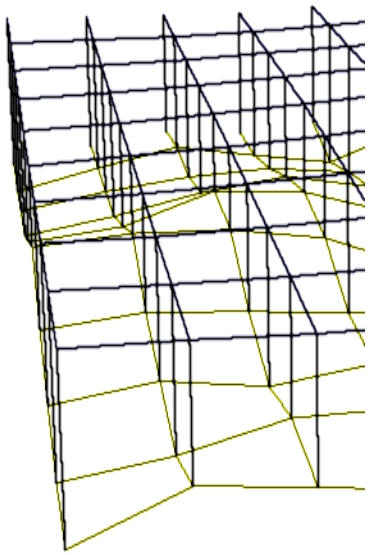


*Obrázek 2.6: Dosažená kvalita generování a vizualizace terénu.*

# 3 Voda

Voda a mnohé přírodní jevy s ní spojené jsou poslední dobou čím dál více diskutovaným tématem. Od lokálních záplav, přes vlny tsunami až po globální oteplování způsobující vzrůst mořské hladiny, vše se jistým způsobem lehce dotýká této práce. Jejím účelem není přesná simulace ani předpovídání těchto jevů, ale spíš jejich názorná interaktivní ukázka, která může některé jedince přivést k bližšímu zájmu o tuto problematiku..

## 3.1 Reprezentace vody v programu



*Obrázek 3.1: Drátěný model vstahu vody k terénu.*

Jelikož je voda velmi úzce svázaná s terénem a reprezentace samotného terénu byla navrhována právě s ohledem na simulaci pohybu vody, je její rozložení uloženo ve stejném dvourozměrném rastru jako terén, což ilustruje obrázek 3.1. Rastr bodů je také základem simulační techniky celulárních automatů, která se používá pro prostorové spojité veličiny. Reprezentace vody tak má naprosto stejná struktura jako terén, počínaje rozdělením na obdélníky a speciální reprezentací okrajových bodů konče.

Program nepracuje se žádnou sítí toků ani jinými vektorovými daty. Rastrový model je vhodný pro velmi snadné modelování náročných prostorových jevů, jako jsou například velké vlny zaplavující pevninu, ale je schopen vytvořit i řeky a klidná jezera. Jeho přesnost závisí na úrovni detailů rastru a na zvoleném časovém měřítku.

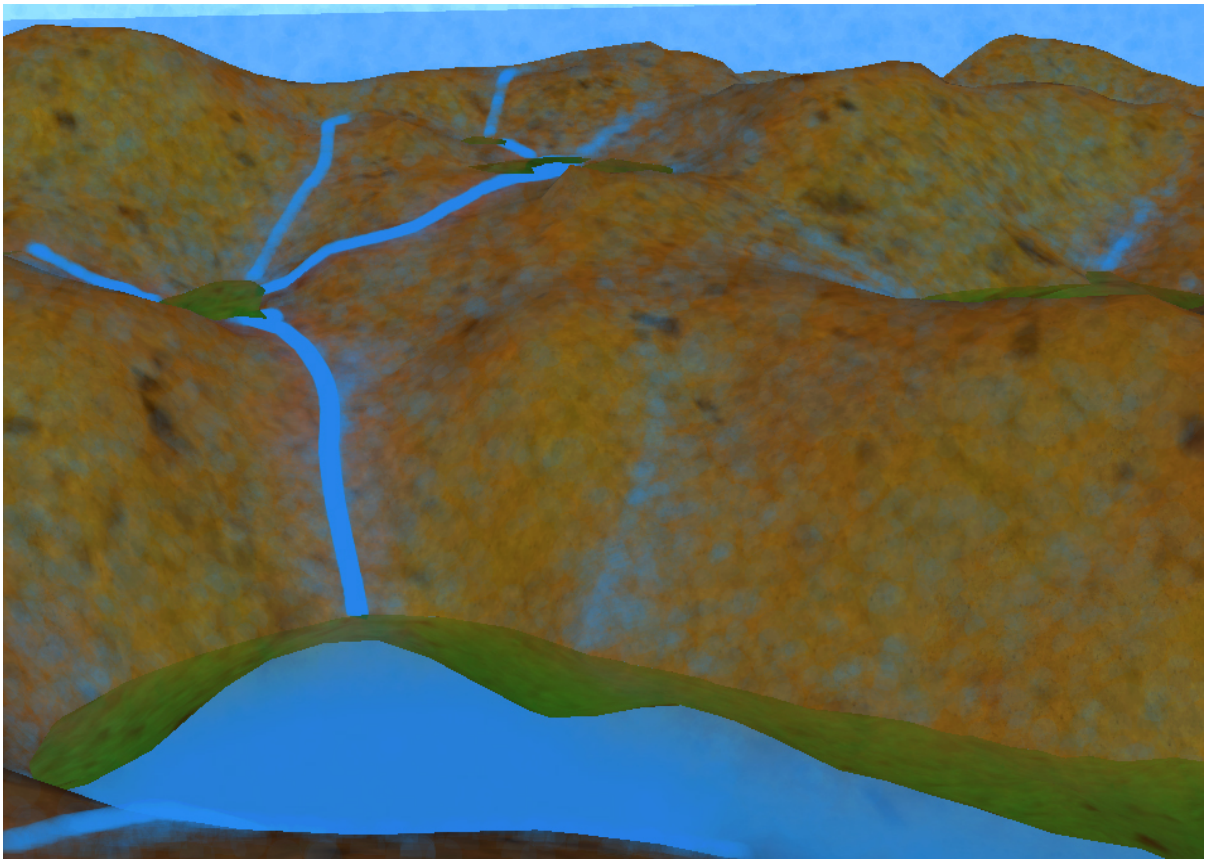
### 3.1.1 Dřívější vývoj

Kromě celulárních automatů je možné pro simulaci tekutiny použít i částice. Pokud se částice mají chovat reálně, musí nejen reagovat na terén, ale navíc i jedna na druhou. V základní verzi algoritmu tak roste náročnost s druhou mocninou počtu částic, což je navíc ve spojení s náročnými výpočty vzdáleností prakticky nevyužitelné. Existují však metody, jak tuto nevýhodu minimalizovat nebo obejít.

První verze tohoto simulátoru měla za cíl pouze vytvořit zajímavý vizuální výstup, a tak si mohla dovolit zjednodušovat až do extrémů. V této verzi částice detekovaly pouze terén pod sebou a tvorba jezer byla zajištěna speciální strukturou jezer. Pro každý dolík v terénu existovala detekční zóna, ke které když se částice přiblížila, tak byla pohlcena a v detekčním bodě se zvedla hladina příslušného jezera. Dále se pak hladina zvedala vždy už při průchodu částice hladinou a jezera rostla a spojovala se až do okamžiku, kdy někde došlo k přetečení přes okraj, kam následně byly přemísťovány pohlcené částice. Tímto jednoduchým způsobem vznikaly toky s kaskádami jezer, která

navíc byla zcela rovná, a tak se v nich daly vytvářet odrazy pomocí stencil bufferu, což je vidět na obrázku 3.2.

Další, a jen z části vyzkoušenou, metodou je rozdělení prostoru opět do jakéhosi rastru, kde každá částice je přiřazena k nějakému bodu a testy kolizí se provádí pouze v rámci tohoto omezeného prostoru, čímž je možno při malém počtu částic ve velkém prostoru dosáhnout velmi výrazného urychlení. Problém pak nastává pouze s rozumnou vizualizací takto získaných dat.



*Obrázek 3.2: Screenshot z dřívější práce využívající navzájem nekolidující částice a speciální objekty jezer pro překonávání prohlubní. Na jezerech je tvořen odraz okolí pomocí planárních reflexí.*

## 3.2 Simulace pohybu

Aby bylo možno s vodou v programu pracovat interaktivně, tedy aby sama dynamicky reagovala na změny způsobené uživatelem, nezbývá než implementovat algoritmus simulující její pohyb. Pohyb tekutin však není úplně triviální záležitost, a tak je nutno jeho principy pro naše interaktivní účely extrémně zjednodušit. V předchozí kapitole byly naznačeny některé postupy pro simulaci tekutin, zde se zaměříme na použitý celulární automat.

### 3.2.1 Celulární automat

Pro simulaci prostorových veličin se velmi často používají celulární automaty. Jejich základní podstatou je rozdělení spojité veličiny na buňky, tedy oblasti ve kterých dochází pouze k malým změnám veličiny, což ideálně odpovídá zde použité rastrové reprezentaci terénu a vody. Jednotlivé buňky pak reagují na základě pravidel a vztahů pouze na své nejbližší okolí. Obsáhlejší úvod lze nalézt například na Wikipedii [WIK07], zatímco přímo simulaci vody a vodní eroze se věnuje [BEN02].

Celulární automaty patří spíše k „silovému“ využití výpočetní techniky, kdy je možno pomocí poměrně jednoduchých programů dosáhnout velmi dobrých výsledků, avšak výpočty mohou být i přes svou jednoduchost pro jejich značné množství hodně časově náročné. Samozřejmě je možné vytvářet i velmi složité a přesné automaty, ale pro tento druh projektu je nejlepší, aby výpočty samy byly co nejjednodušší a pokrytá plocha tedy mohla být co největší. Pokud ale zjednodušíme příliš, nebudou už výsledky realistické ani pro nezkušeného pozorovatele.

### 3.2.2 Návrh implementace

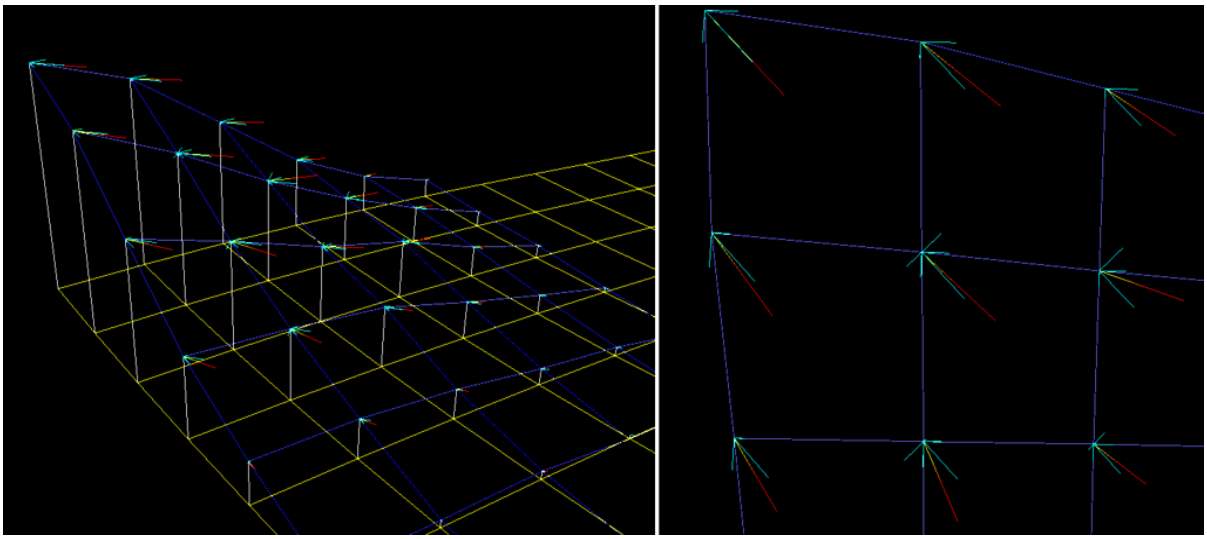
Pro naše účely stačí tato jednoduchá pravidla: 1) voda teče ze svahu dolů, 2) děje se to na základě zrychlení, 3) voda má setrvačnost, 4) existuje tření a v našem modelu nezaznamenané nerovnosti. Z toho odvodíme, že kromě údajů o výšce terénu a výšce na něm ležícího vodního sloupce potřebujeme uchovávat i údaje o rychlosti vody v každé buňce. Prvním nápadem samozřejmě bude uložit do buňky vektor rychlosti. Při bližším prozkoumání ale zjistíme, že takto jednoduché řešení přináší problémy. Tím nejzávažnějším je, že pokud bude vrchol obklopen například nižšími body o stejné výšce, zůstane rychlost v tomto bodě stále nulová a voda z něj nikdy neodteče. Podobné nepříjemnosti se samozřejmě projevují i v méně extrémních situacích, a tak je nutno vymyslet něco lepšího.

Jelikož je samotnou podstatou celulárních automatů interakce mezi sousedícími buňkami, nabízí se možnost ukládat rychlosti pro všechny sousedy. Zde pak je několik variant. Můžeme pracovat pouze se čtyřmi přímými sousedy na osách XY nebo i s dalšími čtyřmi rohovými. Jelikož jednodušší varianta se ukázala být jednoduchá příliš, bude nutno pracovat se všemi osmi sousedy. Pak je ještě dobré zvážit, jestli pro každý vrchol ukládat všech osm rychlostí, nebo vždy jen čtyři a jejich protilehlé reprezentovat pomocí rychlostí v sousedních bodech. Osm rychlostí pro každý bod vykazovalo o něco lepší výsledky, protože je díky němu možné simulovat i jakési proudění, a tak byla zvolena tato varianta. Model sice neukáže jak je proudění hloubkově rozloženo, ale aspoň je schopen ukázat, že v daném místě k nějakému míchání dochází.

Změna rychlosti je prováděna pomocí jednoduchého zjištění rozdílu výšky hladin v aktuálním a v cílovém bodě, následně je zohledněna výška hladiny, protože potůček mezi kameny se bude pohybovat jistě pomaleji než vlna na moři, a nakonec se ještě vloží umělá chyba pomocí pseudonáhodného generátoru, aby se napodobila nedokonalost terénu a různé turbulence. Rychlost přelévání do jedné buňky nesmí být záporná, protože fakt přelévání opačným směrem je počítán při průchodu cílovou buňkou a rychlost přelévání do protilehlé buňky je uložena ve vlastní proměnné.

Na základě vypočítaných rychlostí je později přesouvána voda mezi buňkami. Přitom je nutno zohlednit, že s přesunutím vody do vedlejší buňky se přesune i její energie, a tak je nutno rychlosti ve správných poměrech smíchat. Stejně tak dojde i k určitému zbrzdění protisměrných proudů.

Aby mohl takto jednoduchý simulátor obstojně fungovat, je potřeba mu dodat mnoho různých konstant, které je možno nejlépe odladit až při praktickém testování Výpočty také vyžadují, aby dříve provedené operace neovlivnily v rámci jednoho kroku výpočty následujících buněk, a tak jsou data uložena ve dvou polích, která se vždy po jednom výpočetním kroku střídají.



*Obrázek 3.3: Zobrazení rychlostí v jednotlivých buňkách. Světle modře jsou jednotlivé rychlosti přelévání do sousední buňky, červeně je znázorněn jejich součet. Obrázek vlevo zobrazuje rozlévající se vlnu po terénu, vpravo pak detail rohu z hora.*

## 3.3 Vizualizace vody

Zobrazení vody je jedním z hlavních cílů tohoto projektu. Nejedná se o úplně triviální záležitost, neboť voda vykazuje mnoho vlastností, které v počítačové grafice patří k výpočetně náročnějším. Nejproblematičtější je vodní hladina, která prakticky nikdy není ideálně rovná, což by se do jisté míry dalo vyřešit klasickou texturou, ale vzhledem k dalším vlastnostem, jako je velmi proměnná odrazivost a s ní spojená průhlednost v závislosti na úhlu dopadu paprsků a lom světla, jsou výsledky neuspokojivé. Hladinou navíc problémy nekončí, protože voda bývá poměrně dobře průhledná, přičemž s přibývajícím vodním sloupcem pohlcuje světla stále víc, dokonce různé vlnové délky různě.

### 3.3.1 Odrazy

Jednou ze základních vlastností vody je velmi rozdílná odrazivost v závislosti na úhlu pozorovatele k hladině. Ve spojení se zvlněným povrchem tak vznikají efekty, které bylo ještě donedávna velmi obtížné v reálném čase zobrazovat, což změnil až příchod shaderů. Nicméně odrazy stále zůstávají pro zobrazení v reálném čase velmi obtížnou problematikou. Odrážené objekty lze rozdělit do dvou základních kategorií: globální a lokální, přičemž možnosti jejich zobrazení se radikálně liší.

S odrazy velmi silně souvisí i průhlednost vodní hladiny. Pokud se v ní odráží velké množství světla, není pod jejím povrchem vidět prakticky nic. Stačí však drobná vlnka a hned může být vše jinak. V klasické fixní funkcionalitě je pro tento efekt asi nevhodnější *blending*, nastavený na přičítání hodnoty fragmentu k původní hodnotě framebufferu, čímž supluje fakt, že silný odraz

přezáří vše ostatní. S využitím shaderů je pro změnu lepší využít ve shaderu vypočítanou alfa hodnotu a blending nastavit na míchání podle tohoto koeficientu.

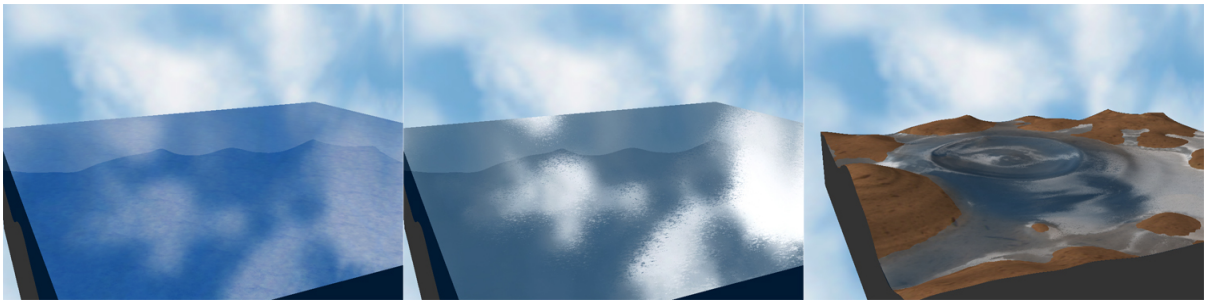
#### 3.3.1.1 Globální odrazy

Globálními objekty jsou z tohoto pohledu ty, které se při pohybu ve scéně nemění. Typickým zástupcem je obloha, která je natolik vzdálená, že se její poloha vůči pozorovateli prakticky stále stejná. Takové odrazy je možno zobrazovat velice snadno a existuje několik různě vhodných a dokonalých postupů, jak toho dosáhnout.

OpenGL v základní verzi nabízí režim automatického generování texturových souřadnic podle normál renderovaných vrcholů. To je ve spojení se speciální texturou, která může být například pořízena extrémně širokoúhlým fotoaparátem, schopno imitovat odrazy prostředí - *environment mapping*.

Tato základní verze má však značné limity, jako například nutnost nové textury pro každý úhel pohledu, a tak byly vyvinuty pokročilejší metody. Jednou z nich je *dual paraboloid mapping*, který sice řeší do určité míry předchozí problém, ale stále není příliš jednoduchý pro použití (získávání správných textur) a navíc vyžaduje dva renderovací kroky.

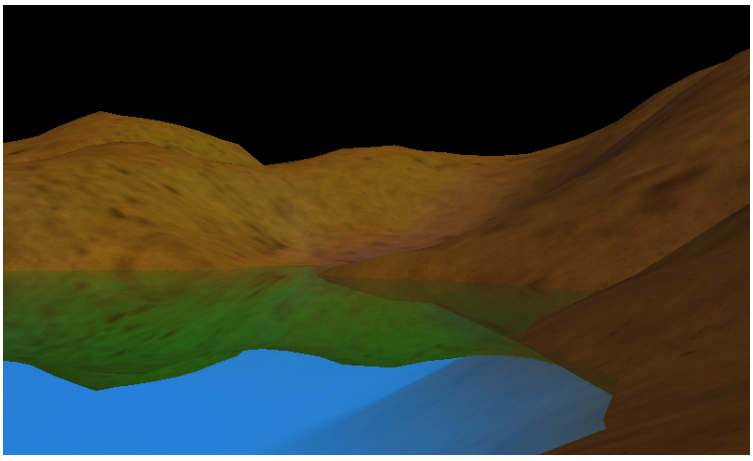
V současné době se pro environment mapping nejčastěji používá jiných speciálních textur – *cube mapping*. Jde vlastně o šest normálních čtvercových textur, které dohromady tvoří krychli, na kterou je pak promítána normála vrcholu, určující výsledný texel. Výsledný efekt je vidět na obrázku 3.4.Takové textury jsou přístupné i v OpenGL pomocí rozšíření, a tak je využívá i tento projekt pro odrazy oblohy. Výhodami tohoto přístupu je vysoká kvalita textury pro všechny směry odrazu bez nutnosti texturu měnit při změně úhlu pohledu, vysoká rychlost i jednoduchost tvorby textur.



*Obrázek 3.4: Zobrazení globálních odrazů oblohy. Vlevo rovná hladina s krychlovou texturou bez využití shaderů. Uprostřed ta samá scéna s použitím shaderů pro vytvoření iluze drobných vlnek. Vpravo je pak vidět krychlová mapa se shadery na zakřivené ploše velkých vln.*

#### 3.3.1.2 Lokální odrazy

Lokální objekty tvoří samotnou scénu, jejich vzdálenost už není zanedbatelná a naopak mnohdy přímo sousedí s odrážejícími plochami, respektive jimi samy jsou. V takových případech už je bohužel výše uvedené zjednodušení naprosto nepoužitelné. Dobrých výsledků se dá ještě docílit u rovných plocho, kdy je odrážený obraz vyrenderován pro každou odrážející plochu znovu a pak například pomocí stencil bufferu složen do jednoho výsledného snímku, ale pro zakřivené plochy je tato metoda prakticky nepoužitelná. V reálných aplikacích se pro tyto účely používá spíše metoda zvaná ray-tracing, která je však velmi pomalá.



*Obrázek 3.5: Planární odraz renderovaný za pomoci stencil bufferu. Na zakřivené plochy je však tato technika nepoužitelná.*

Protože je v programu celá scéna velmi dynamická a odrážející plochy často značně zakřivené, není v současné době možné efektivně zrcadlit okolní terén ve vodě. I když se často tvoří poměrně rovná jezera, málokdy je jejich povrch skutečně natolik rovný, aby se dala využít aspoň výše zmiňovaná technika zrcadlového renderování ukázaná na obrázku 3.5, která je navíc při větším počtu zrcadlících ploch značně časově náročná.

Jistou minimální funkčnost skýtá aspoň variace *ray-tracingu*, kde by bylo možno počítat odraz terénu pro každý bod rastru vody poměrně snadno průchodem přímky po rastru terénu. Výsledkem by pak byla pouze informace, že se ve vodě v daném bodě něco odráží, ale při vhodně aplikovaném shaderu na mapované textury by efekt mohl vypadat relativně dobře.

### 3.3.2 Hloubka

Hloubku vody je možno zobrazovat několika způsoby. Nejjednodušší je modifikovat alpha kanál barvy vrcholu podle hloubky vody v daném bodě. Takové zobrazení je rychlé a při pohledu z hora i dobře vypadá. Problém nastává při umístění kamery poblíž mělké vody a zaměření směrem do hloubky, kdy může být vidět i přes velkou masu vody bez patřičného zamlžení.

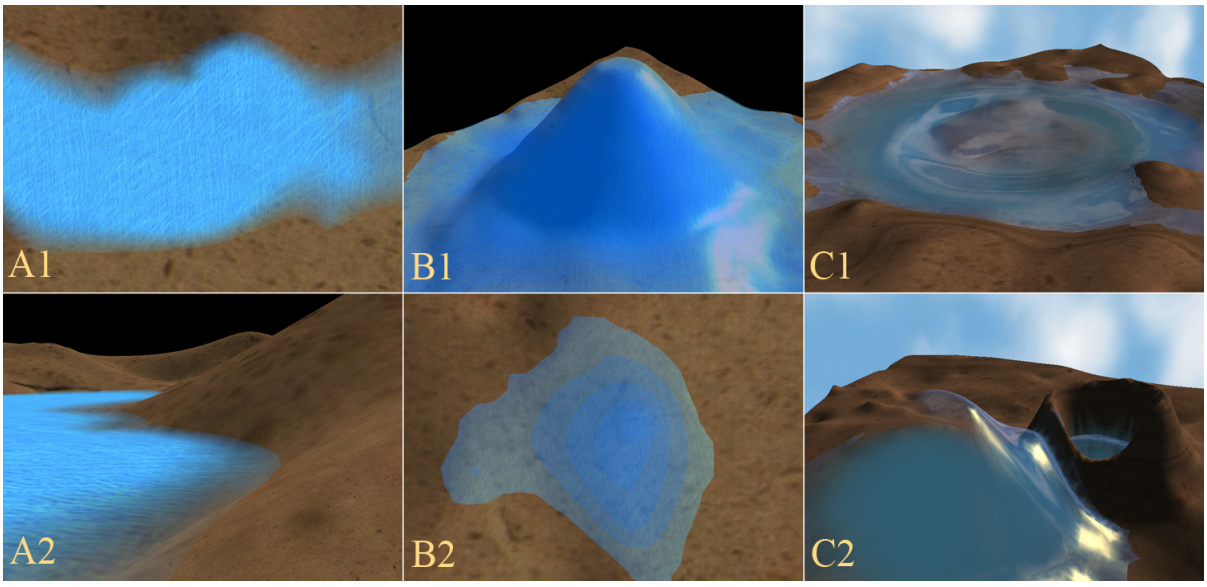
Další jednoduchou variantou je vykreslení hladiny vody v různých hloubkách, přičemž barva hladiny je na základě alpha hodnoty vždy přimíchávána k původnímu obsahu framebufferu jen částečně. Takové zobrazení sice do značné míry řeší problém s mělkou vodou na okrajích z předchozího způsobu řešení, avšak samo má hned několik nedostatků. Nejpodstatnější je, že přechody mezi vrstvami nejsou plynulé a zvyšováním počtu vrstev pro omezení těchto rozdílů značně roste počet vykreslovaných polygonů a s ním i doba vykreslení snímku.

Kromě těchto základních funkcí nabízí OpenGL rozšíření v podobě speciálního režimu mlhy, kdy je možno pro každý vykreslovaný vrchol zadat údaj o jeho zamlžení místo jeho počítání na základě standardních rovnic. Takto lze dosáhnout již velmi slušných výsledků při zachování vysoké rychlosti, nicméně stále je možno nalézt pár neduhů. Asi nejvýraznějším je interpolace mlhových souřadnic, takže zamlžení není automaticky limitováno jen pod hladinu. Za normálních okolností je tento jev prakticky nepozorovatelný, dokud se vedle sebe nevyskytnou dva body se značně rozdílnými vodními sloupci.

Poslední zde uvedenou možností bude využití shaderů a víceprůchodového renderování. Za pomocí shaderů je možno vyrenderovat terén i s jeho hloubkovou hodnotou v z-bufferu a následně tyto údaje využít při porovnávání se z-bufferovou hloubkou povrchu vody a přesně tak určit masu vody pod daným pixelem a tak i jeho výslednou barvu. Mezi problémy bude patřit přesnost z-bufferu a nejspíš i přístup k jeho hodnotám přímo ze shaderu. V případě omezení přístupu lze použít

renderování do hloubkové textury, se kterou už je možno pracovat bez problémů, ale znamená to další prodloužení doby pro renderování obrázku.

V aktuální verzi využívám variantu s mlhou. Kromě varianty se shadery byly postupně odzkoušeny a vyhodnoceny jako nevyhovující i ostatní zmiňované postupy, jejichž výhody a nevýhody ilustruje obrázek 3.6.

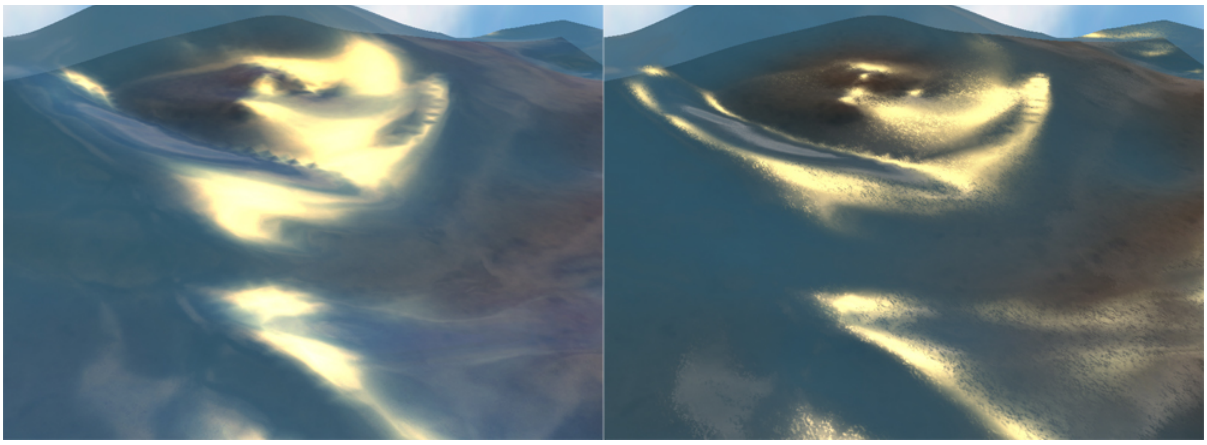



*Obrázek 3.6: Různé techniky zobrazení hloubky vody. Horní řada A1, B1, C1 ukazuje dobré vlastnosti technik zadávání alfa hodnot, vykreslování více vrstev a zadávání mlhových souřadnic. Spodní řada A2, B2, C2 pak ukazuje hlavní nedostatky těchto technik. Jako technika s nejlepším poměrem kvality k náročnosti výpočtů vychází varianta s mlhou.*

### 3.3.3 Shadery

V programu je implementován i jednoduchý shader pro zobrazení vody. Za účelem možnosti porovnání výsledků je možno plynule přepínat mezi zobrazením pomocí klasické statické funkcionality a shaderem. Pomocí shaderů lze dosáhnout téměř realistického zobrazení vody. Nejdůležitějšími rysi implementovatelnými až za použití shaderů jsou jemné vlnky na vodě s různým odrazem světla a lom na vlnkách. K tomu je potřeba vytvořit 3D texturu s normálovou mapou, která je generována za pomocí algoritmu Perlin noise ve 3D. Shader pak počítá osvětlení a odrazy pro každý renderovaný fragment zvlášť, a tak může k normálám definovaných ve vrcholech přičítat odchylku branou z 3D textury. Rozdíl je vidět i na obrázku 3.7, nicméně ten nemůže zachytit shaderem prováděnou animaci.

Pro statické zobrazení by stačila pouze klasická 2D textura, ale pokud chceme animovat změnu vlnek v čase, je právě postupný přechod mezi vrstvami 3D textury interpolovanými v hardware velmi dobré řešení.



*Obrázek 3.7: Rozdíl ve zobrazení statickou funkcionalitou OpenGL vlevo a s použitím jednoduchého shaderu vpravo.*

### 3.3.4 Vyhlazení rastru

Stejně jako terén, je i voda vykreslována pomocí čtyř trojúhelníků mezi čtyřmi vrcholy rastru, přičemž se přidává jeden středový bod. Na rozdíl od terénu pak je výška středového bodu počítána jako průměr dvou bodů vyšší diagonály obdélníku, aby tak byl zohledněn fakt přelévání vody. To ale bohužel k dobrému zobrazení nestačí. Jelikož je vodní hladina v daném bodě reprezentována jako výška vodního sloupce nad bodem, může být hladina zobrazena v důsledku interpolace trojúhelníky i zde, pokud je nějaký sousední bod zatopen. Právě rozumné zobrazení výsledné hladiny se ukázalo jako jeden z největších problémů.

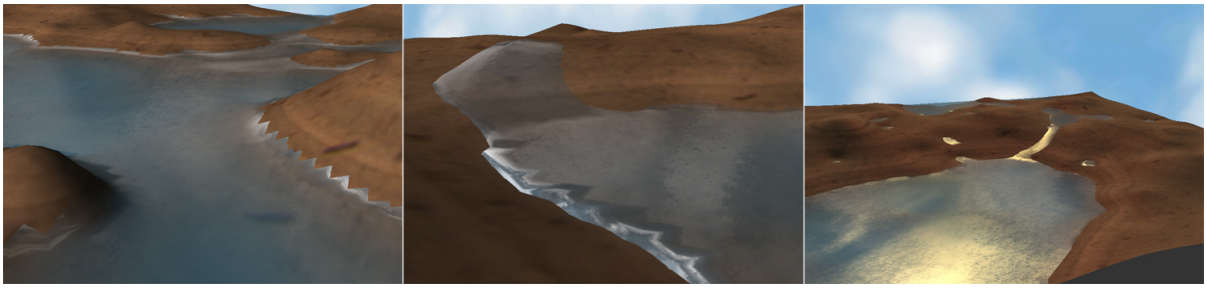
Jako první řešení, aby voda nezakrývala celý povrch terénu, se nabízí posunout celou hladinu o kousek dolů, čímž dojde k jejímu zobrazení až v případě jistého minimálního množství vody, které lze chápat jako vodu podpovrchovou nebo vsáklou do písku. Bohužel se tím ale nevyřeší problém s příkrými trojúhelníky vodní hladiny například u jezer s šikmými břehy, které se jednoznačně nedají pokládat za kvalitní výstup.

Za účelem odstranění tohoto neduhu byl vyvinut následující algoritmus: Nejdřív je hladina vody v pomocném rastru přepočítána tak, že jakmile hladina *water* překročí určité minimum *min*, odečte se od ní navíc hodnota $x = 1 - water / min$.

To sice poskytuje lepší vizuální výsledek díky plynulým přechodům, ale místo aby trojúhelníky vystupovaly nahoru, propadají se na okrajích zase příliš příkře do země. Takto získaná data jsou tedy předmětem dalšího zpracování. V dalším kroku se zjišťuje, jestli v okolí bodů, jejichž nyní vypočítaná hladina se nachází pod terénem, nejsou body s viditelnou hladinou. Pokud se tak totiž stalo, je možno sousednímu podpovrchovému bodu přiřadit výšku jeho povrchového souseda a tak vyrovnat hladinu na okraji jezer a podobných útvarů. Pokud je v okolí nadpovrchových bodů více, jejich výšky se průměrují.

Aby se navíc zabránilo zobrazování trojúhelníků vody v místech, kde by vůbec nebyly vidět, protože jsou plně překryty terénem, poznamená se tato skutečnost do pole vrcholů jako velmi nízká hodnota, na kterou je pak při zobrazení vrchol testován a nemusí se opět provádět porovnávání s výškou terénu.

Postup vyhlazování a konečný výsledek je vidět na obrázku 3.8. Pokud by se do modelu nevkládala umělá odchylka, výsledek by byl velmi hladký a plynulý. Simulací nerovností a turbulencí však mohou vznikat některé artefakty. Stejně tak může nějakou chvíli trvat, než se okraj jezera ustálí.



*Obrázek 3.8: Postup vyhlazování. Vlevo vidět nevyhlazená varianta pouze lehce posunutá dolů. Uprostřed se nachází první fáze vyhlazování a vpravo pak finální výsledek.*

## 3.4 Možnosti rozšíření a dalšího vývoje

### 3.4.1 Vrstvy vody

Pokud v budoucnu vyvstane potřeba simulovat podpovrchovou vodu, vodopády, nebo třeba jen opravdu reálné proudění například mořských proudů, které se značně liší v různých hloubkách, bude bezpodmínečně nutné přidat do reprezentace vody třetí rozměr. To si samozřejmě vynutí i kompletní přepsání celého simulátoru a velmi pravděpodobně značně omezí jeho interaktivitu, protože objem výpočtů přeroste možnosti současných počítačů. Možná se ukáže že pro takovéto výpočty bude vhodnější výše zmíněná rastrově-částicová metoda, zvlášť pokud bude vody v terénu jen malé množství, kde je výhodnější počítat jen s malým množstvím částic než se všemi body rastru.

### 3.4.2 Vylepšení shaderů

Při využití výpočtů masy vody pomocí z-bufferu je možno reálně zobrazovat hloubku vody tak, že zjistíme hloubku vyrenderovaného podkladu a následně hloubku renderované vody v z-bufferu. Jelikož ale není v průběhu renderování možné ze z-bufferu zároveň číst, je nutné podklad vyrenderovat předem do textury. K tomu lze mimo jiné využít i rozšíření, které dovoluje současně renderovat do několika cílů včetně textur. Tím si zároveň můžeme připravit podklad pro renderování lomu světla na vlnkách a další operace.

Shader vody je dále možno vylepšit o *paralax mapping* drobných vlnek, který ještě zvýší iluzi jejich trojrozměrnosti. Velmi vhodné by také bylo použít podobnou techniku na vytvoření dojmu lomu světla na vlnkách pomocí vyrenderování dna předem a při renderování vody pak používat tento obrázek pro jeho deformované mapování.

# 4 Vizuální kvalita

V této kapitole se hodlám věnovat několika obecným postupům, které slouží k celkovému zlepšení vizuální kvality trojrozměrné grafiky, a tak se o nich nehodilo psát v dřívějších kapitolách. Základy OpenGL a trojrozměrné grafiky jsou popsány v [WOO97] a [KRS07], takže se zmíním spíše o pokročilejších technikách.

## 4.1 Textury

Pro kvalitní 3D grafiku bohužel nestačí pouze teoretické podklady a dobré algoritmy. Stejně tak je důležitý i dobře vypadající trojrozměrný model scény, ale i ten by vypadal velmi stroze bez použití textur. Aby se nemusely veškeré detaily scény vykreslovat pomocí obrovského množství trojúhelníků, používají se raději rastrové obrázky, které se na menší počet trojúhelníků nanášejí jako tapety, čímž je možno dosahovat vysoké úrovně detailů při zachování rychlosti zobrazení.

Existuje mnoho druhů textur, z nichž většina je použito i v tomto programu. Základní rozdělení je založeno na počtu rozměrů, které má rastr textury. Jednorozměrná textura je například použita pro znázornění vrstev terénu, dvojrozměrná pro jeho povrch a trojrozměrná pro animaci vody.

Trochu mimo stojí krychlová textura tvořená šesti čtvercovými 2D texturami, které dohromady tvoří povrch krychle. Takové textury nacházejí využití zejména při zobrazování odrazů a jiných světelných efektů, ale stejně dobře se dají použít i při vykreslování *sky box*, což je extrémní zjednodušení oblohy a vzdáleného okolí scény, které je použito i zde.

Dalšími parametry jsou pak obsah a význam textury, která může obsahovat od indexu barvy v paletě přes RGBA komponenty až po hloubková data a normály povrchu.

### 4.1.1 Vytváření textur

Jelikož se jedná o obrázky, není vytváření kvalitních texturu tak jednoduché, jak se na první pohled může zdát. V praxi je vytváření textur svěřeno profesionálním grafikům a programátor se o ně nemusí zajímat. V rámci této práce byly jako zdrojová data použity textury z volně přístupných internetových databází, které jsem navíc dodatečně upravoval, aby lépe vyhovovaly svému účelu. Prvním kriteriem jsou rozměry v mocninách dvojky, ale hlavní práce se odehrála na jejich úpravě pro plynulé navazování při opakování, aby nevznikaly mezi jednotlivými čtverci ostré předěly a z dálky viditelné vzory.

### 4.1.2 Generování textur

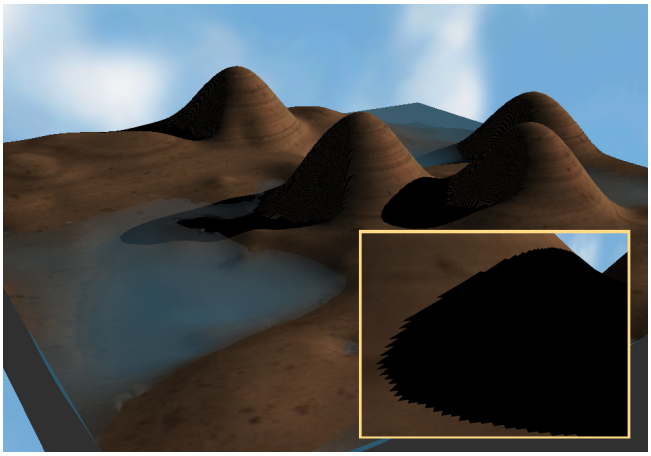
Místo focení, ručního upravování a následného načítání textur ze souborů je možné vše generovat i programem, což je samozřejmě z hlediska této práce mnohem zajímavější. U textur mraků a terénu by jistě bylo možné vytvořit algoritmy pro jejich realistické generování, kde by opět našly uplatnění například metody použité při generování rastru terénu. Bohužel, stejně jako mnoho dalších funkcí se generování textur do tohoto projektu zatím nevešlo, a tak zůstává pouze jako námět na další práci. Jedinou výjimkou je generování trojrozměrné opakující se textury vlnek pro normálovou mapu shaderu vody, kde je použit upravený algoritmus Perlin noise.

## 4.2 Stíny

Stíny jsou v 3D grafice velmi závažným tématem. Stínování objektů podle normál plošek je operace velmi jednoduchá, ale s vrháním stínů na ostatní objekty nemá nic společného. Například metody u *raytracing* je výpočet zastínění velmi snadný, nicméně metoda je to pomalá a pro zobrazení v reálném čase se naprosto nehodí. OpenGL navíc používá renderování s využitím z-bufferu, což je technika prakticky nekompatibilní.

V souvislosti s OpenGL se často uvádí technika *shadow volumes* využívající stencil buffer. Její podstatou je vytvoření bitmapy, která reprezentuje zastíněné oblasti na obrazovce a scéna je následně vyrenderována ve dvou krocích. Jednou celá osvětlená a jednou maskovaná stencil bufferem bez osvětlení. Jedná se spíše o techniku používanou dříve, která generuje na obrazovce přesné ostré stíny a její použití není příliš jednoduché a rychlé. V dnešní době se však mnohem více prosazuje metoda *shadow mapping*.

### 4.2.1 Shadow mapping



*Obrázek 4.1: Metoda shadow mapping. Ve výřezu je vidět přílišné zvětšení rastru.*

Jednou z nejlepších variant pro renderování stínů v reálném čase je *shadow mapping*. Jeho podstata je velmi jednoduchá – z pohledu zdroje světla se vyrenderují všechny objekty vrhající stíny a hloubková komponenta výsledku se pak použije jako textura, která se s pomocí automatického generátoru texturových souřadnic nanese na všechny potenciálně zastíněné objekty.

Aby toto bylo možné, je potřeba OpenGL rozšířit o podporu hloubkových textur a porovnávací operace, které zajistí správné renderování renderování stínů. Scéna je pak obvykle kreslena dvakrát. Nejdřív celý obraz jako by byla celá scéna ve stínu a pak se na základě hodnot stínové mapy vykreslí nezastíněné oblasti v normálních barvách. Jako další vylepšení je možno přidat renderování mapy přímo do textury a renderování scény za pomoci shaderů, které jsou schopny korektně vykreslit zastíněné i osvětlené oblasti v jednom průchodu.

Velkými výhodami této metody jsou její jednoduchost, rychlost a univerzálnost. Nevýhodami jsou výrazný pokles rychlosti při renderování stínů z více zdrojů a rastrová povaha stínů. První problém je způsoben nutností vykreslit celou scénu a následně použít více stínových map pro každý zdroj světla, druhý se negativně projevuje při velkém přiblížení kamery k zastíněnému objektu a při pohledech proti zdroji světla, přičemž druhý problém se dá částečně eliminovat zvětšením rozměrů stínové mapy a vhodnou úpravou projekce při jejím renderování a nanášení. Asi nejlépe vypadá úprava renderování a projekce stínů s pomocí techniky *trapezoid shadow mapping*, která je schopná renderovat stínovou texturu blíže ke kameře ve větších detailech než vzdálenější objekty.

### 4.2.2 Další možnosti

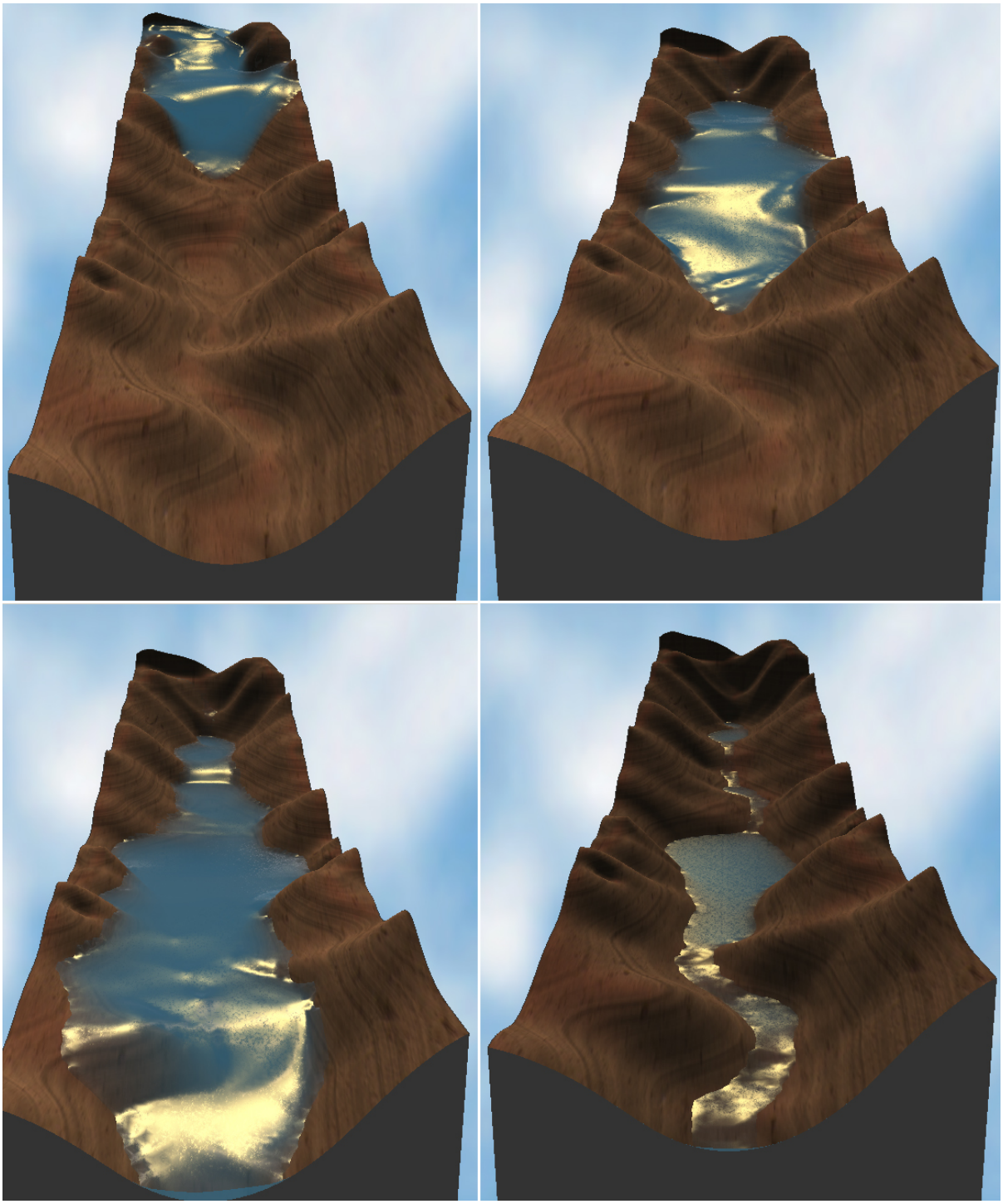
V důsledku reprezentace terénu jako výškového pole se jako poměrně zajímavá možnost výpočtu stínů jeví i jistá obdoba *raytracingu*. Její podstatou je projití všech vrcholů na přímce směrem ke zdroji světla a zjištění, jestli ho některý z vrcholů nepřekrývá. Vzhledem k povaze výpočtů a množství procházených dat je tato metoda velmi rychlá a jistě stojí za zvážení. Mezi nevýhody samozřejmě patří nízká přesnost a silně omezené možnosti nasazení.

## 4.3 Shadery

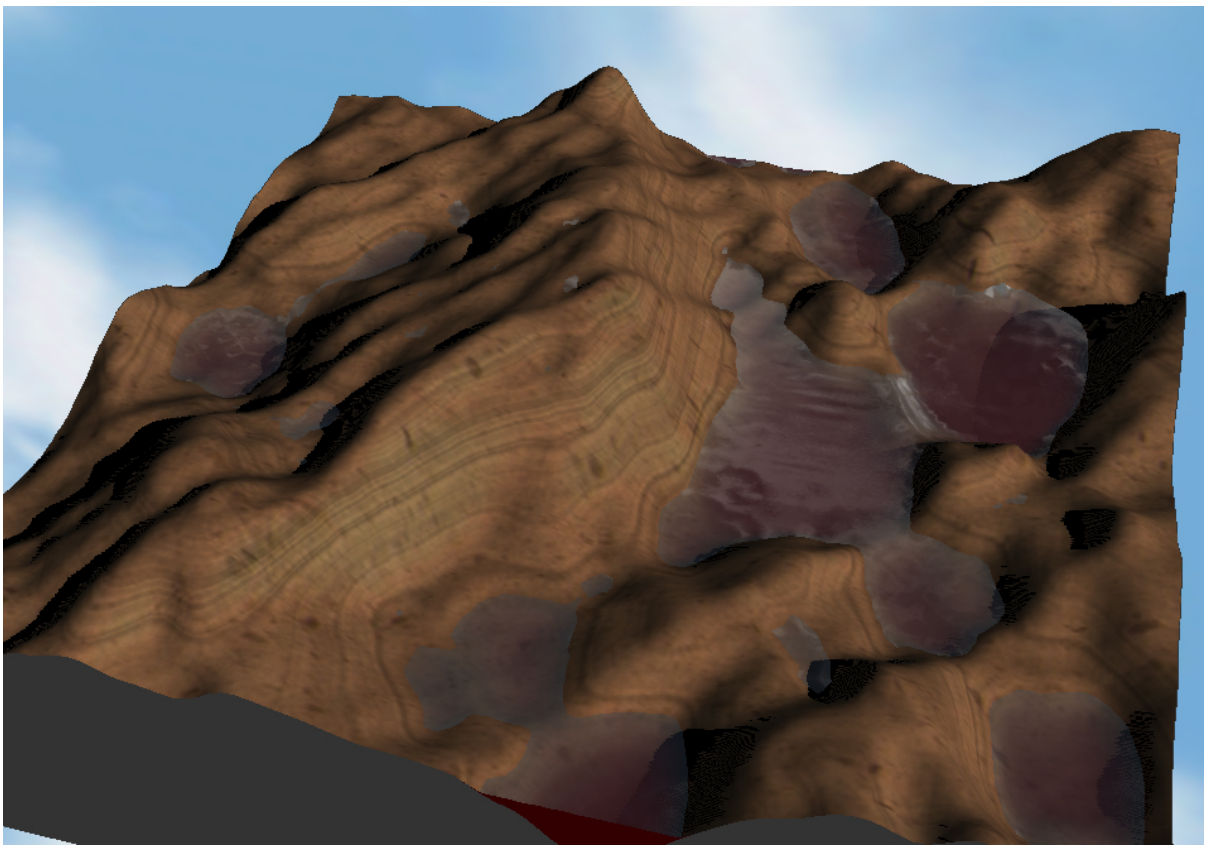
V programu je implementován i jednoduchý shader vody. Ten se skládá z vertex shaderu, programu pro práci s jednotlivými vrcholy, kde se provádí transformace vektorů pomocí matic do výsledného prostoru a některé operace, které následně usnadní práci fragment shaderu, který se stará o vykreslování jednotlivých pixelů do framebufferu.

Výpočetní síla a možnosti současných shaderů jsou velmi značné a tento projekt zdaleka nevyužívá všech jejich možností. OpenGL používá pro tvorbu vrcholových a fragmentových programů jazyk GLSL, který je popsán v [KES06].

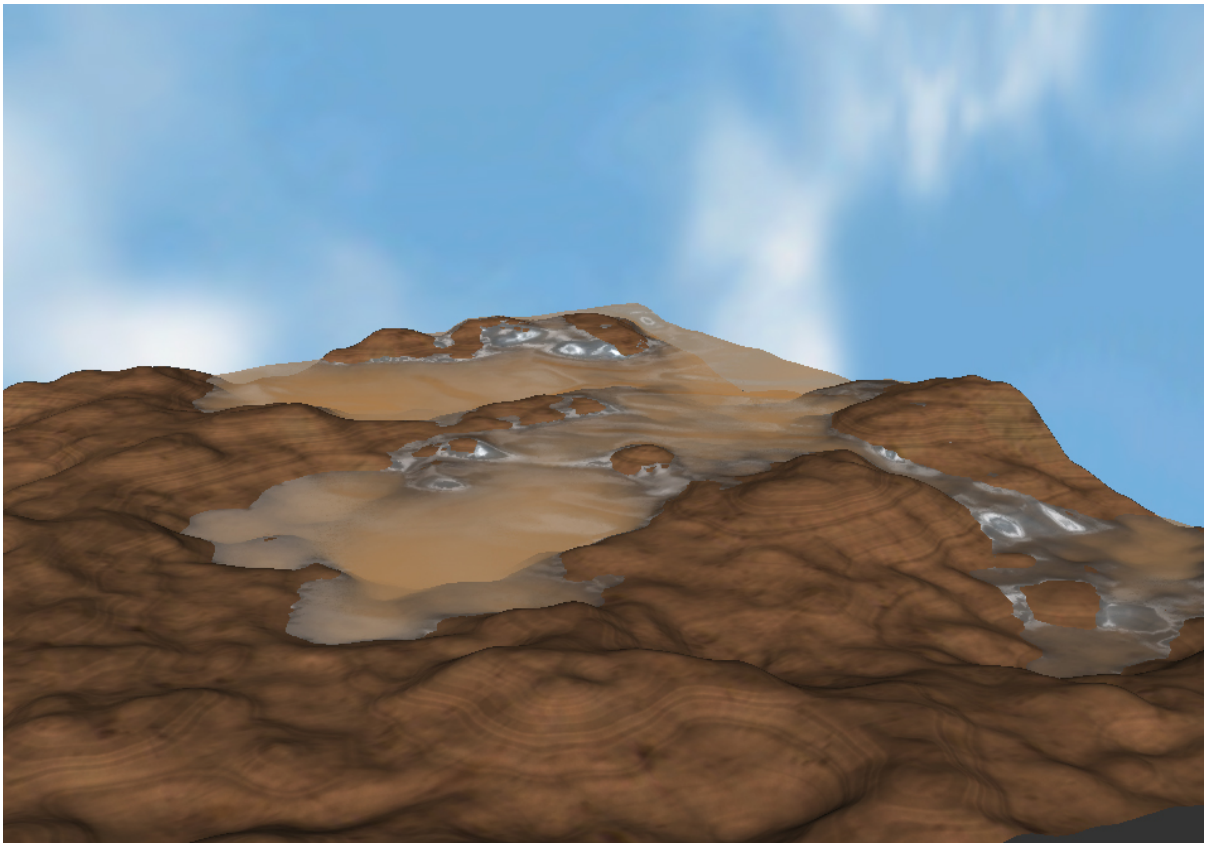
## 4.4 Dosažené výsledky



*Obrázek 4.2: Animace záplavové vlny v údolí.*



*Obrázek 4.3: Silně zvrásněný terén zalitý rudou vodou, zobrazeny stíny.*



*Obrázek 4.4: Velké vlny v rozsáhlém terénu.*

# 5 Závěr

Tato práce dle zadání rozebírá možnosti vizualizace tekoucí vody v krajině s bližším zaměřením na zobrazování výsledků v reálném čase. Byly zde nastíněny základní i pokročilejší dnes používané techniky generování terénu, simulace toku vody a kvalitního zobrazení trojrozměrné grafiky v reálném čase.

V druhé kapitole je navržen algoritmus rozdělující rastrový terén do více oblastí o různé hustotě a mechanizmus interakce mezi jednotlivými takto vzniklými obdélníky. Cílem je směřování výpočetní síly na uživatelem definovaný prostor a ve výsledku možnost pracovat s prostorově rozsáhlejším terénem. Dále jsou diskutovány algoritmy pro umělé generování terénu a možnosti jeho zobrazení.

Kapitola třetí pak mimo jiné pojednává o simulaci toku vody v terénu. Za tímto účelem je navrhnut jednoduchý celulární automat, který se snaží dodržet v rámci zachování rychlosti výpočtů všechny základní fyzikální vlastnosti vody.

Pro kvalitní zobrazení trojrozměrné grafiky je navrženo využití pokročilých technik multitexturing, krychlové a trojrozměrné textury, programovatelné shadery využívající normálových map a renderování stínů.

Na základě tohoto rozboru byl implementován interaktivní simulátor pohybu vody terénem s důrazem na vizuální kvalitu výstupu. Kromě možností dalšího zlepšení vizuální kvality, například v podobě přidání vegetace a dalších detailů, zde byl nastíněn i předpokládaný směr vývoje ke zlepšení fyzikální simulace k plně trojrozměrné vodní erozi, což již značně přesahuje rámec aktuálního zadání. Projekt však byl navrhován s úmyslem tato rozšíření v budoucnu postupně implementovat, čemuž odpovídá i jeho struktura. V současném stavu práce je zadání splněno.

# Literatura

FER07: Fernandes, A., R., Terrain Tutorial, , http://www.lighthouse3d.com/opengl/terrain/
MAR02: Martz, P., Generating Random Fractal Terrain, 2002,
http://www.gameprogrammer.com/fractal.html
TER07: Planetside Software, Terragen Website, 2007, http://www.planetside.co.uk/terragen/
PER99: Perlin, K., Making Noise, 1999, http://www.noisemachine.com/talk1/index.html
WIK07: Wikipedia contributors, Cellular automaton, 2007,
http://en.wikipedia.org/w/index.php?title=Cellular_automaton&oldid=128057647
BEN02: Benes, B., and Forsbach, R., Visual Simulation of Hydraulic Erosion, 2002
WOO97: Woo, M., Neider, J., Davis, T., OpenGL Programming Guide, 1997
KRS07: Kršek, P., Základy počítačové grafiky, Studijní opora, 2007
KES06: Kessenich, J., The OpenGL Shading Language, 2006

# Seznam příloh

# Příloha A: Manuál

Program je napsán pro okenní prostředí Microsoft Windows. Z toho plyne, že je možné ho téměř celý ovládat myší. Jednak pohybem myši po zobrazované scéně a jednak prostřednictvím menu. Kromě myši je většina operací dostupná i přes klávesové zkratky a pro některé operace je potřeba kombinace myši i klávesnice.

## Pohyb scénou

- Vypnutí automatické rotace – A
- Pohyb scénou – pohyb myší se stisknutým levým tlačítkem
- Otáčení scény – pohyb myší se stisknutým pravým tlačítkem
- Přibližování scény – pohyb myší se stisknutým levým i pravým tlačítkem

## Interaktivita

- Editační režim – Tab
- Přidání hory vody – Enter
- Přidání pramene – Mezerník
- Reset rychlosti vody – R
- Celkový reset vody – Ctrl + R
- Změna velikosti kurzoru – pohyb myši se stisknutým pravým a levým tlačítkem a klávesou Shift
- Editace výšky terénu – pohyb myší se stisknutým levým tlačítkem a klávesou Shift
- Vyhlazování oblasti kurzoru – S
- Vyhlazení celého terénu – Alt + S

- Nápověda – F1
- Zapnutí deště – F2
- Vypnutí animace vody – F3
- Vypnutí akváriového efektu - F4
- Generování nového terénu – F5
- Parametry generování terénu a import bitmapy – F6
- Změna časového měřítka – klávesy plus a mínus

## Zobrazení a další možnosti

Všechny zmíněné možnosti a mnoho dalších lze nalézt v nabídce programu. V položce View je možno nastavit mnoho parametrů zobrazení. Nabídka Terrain obsahuje položky vztahující se ke generování terénu a položka Water zastřešuje nastavení simulace toku vody.

V programu je možno přepínat z fixované funkcionality OpenGL na shadery, natáčet osvětlení, volit zobrazení stínů, nastavovat barvu vody, časové měřítko simulace, intenzitu srážek, rozměry a algoritmus generování terénu včetně jeho základního tvaru.

# Příloha B: Implementace

Pro vytvoření tohoto programu byl vybrán programovací jazyk C++ pro jeho velké rozšíření, podporu, dostupné knihovny a možnost objektového programování. Mezi výhodné vlastnosti dnešních překladačů patří široké spektrum úrovní programování, kdy lze běžně vkládat do kódu v případě nutnosti i úseky jazyka assembler a hned vedle využívat objekty s virtuálními metodami a mnohonásobnou dědičností.

Za účelem snadné práce s 3D grafikou je jako základní API využívána multiplatformní grafická knihovna OpenGL, která dovoluje využívat většiny vlastností aktuálního hardware, aniž by se programátor musel příliš starat o jeho specifika. Pokud jsou však vyžadovány některé pokročilejší funkce, je nutno použít mechanizmus rozšiřování, který sice umožňuje přístup ke všem vymoženostem, které daný hardware nabízí, ale zase si vynucuje implementace záložních variant kódu. Alternativou k OpenGL by mohla být knihovna DirectX, která ale není zdaleka multiplatformní. Informace o OpenGL byly čerpány převážně z [OpenGL] a různých internetových tutoriálů.

Jako vývojové prostředí bylo pro nabízené funkce vybráno Microsoft Visual C++, což bohužel znamená i to, že program sám je napsán pro Microsoft Windows, čímž je omezena jeho přenositelnost. Původně jsem měl v plánu využít technologii .NET, ale ukázalo se, že OpenGL běží v poskytovaných formulářích nesmyslně pomalu a i rychlost samotných výpočtů utrpěla nepřiměřený šok, a tak je program nakonec napsán za použití starého Win32 API.

## Struktura programu

Program je rozdělen do několika modulů, které reprezentují určitou vrstvu programu a nachází se v samostatných souborech. V souboru *main.cpp* se nachází vstupní bod programu, inicializace oken, OpenGL a potřebných proměnných, stejně jako hlavní smyčka programu a zpracování zpráv.

Objekt *terrain* pak reprezentuje veškerá data potřebná pro simulaci a zajišťuje zobrazení globálních objektů, jako je *sky box* a zobrazování stínů. Sám se pak skládá z objektů *cluster*, které reprezentují jednotlivé obdélníky. Tyto objekty se starají o simulaci a vykreslování vlastních dat, přičemž obsahují pole dvou objektů *vertex_2d_field*, které zapouzdřují samotná data vrcholů a sdružují okrajové a vnitřní body rastru. Objekty z pole dat *vertex_2d_tield*, se pravidelně střídají v aktivitě, přičemž jeden z nich má vždy roli cíle a jeden roli zdroje dat. Ukazatel na aktuální pole se jmenuje *cvf* (zkratka „current vertex field“) a ukazatel na staré hodnoty *ovf* („old vertex field“).

Objekt *vertex_2d_field* obsahuje pole ukazatelů na virtuální bázové objekte *vertex_base*, do nichž jsou pak podle umístění ukládány ukazatele buď na objekty *vertex_border* pro okraje nebo na *vertex_cluster* pro datové body. Pro snadný přístup k jednotlivým prvkům je implementována mapovací funkce s parametry ve dvou rozměrech.

## Přídavné knihovny

Pro ulehčení některých operací jsem využil svou starou knihovnu pro práci s OpenGL. Tato knihovna není přímo součástí projektu a i když vykazuje některé zajímavé rysy, byla z velké části napsána ještě před nástupem na VUT, takže obsahuje i mnoho nedořešených úseků. Obsahuje velké množství kódu, z nějž je v tomto projektu použito jen málo, ale raději než ji rozdrobovat nebo používat kód cizí jsem se rozhodl využít ji.

Jejím základním cílem bylo vytvořit sjednocující objekt, pomocí kterého by bylo možno spravovat 3D scénu vykreslovanou za použití OpenGL. Pro tyto účely existuje superobjekt, který obsahuje knihovny pojmenovaných dílčích objektů různého typu. Knihovna pracuje s 3D objekty, texturami, materiály, světly, kamerami, vektory, maticemi, fonty a různými dalšími komponentami. Jak už tomu tak u podobných ambiciózních projektů bývá, vývoj se zastavil někde uprostřed. Mnoho kódu ale funguje, a tak není důvod ho nevyužít. Tento projekt využívá zejména zde implementované vektory, matice, materiály, textury, správu rozšíření OpenGL a objekty výjimek. Knihovna sama vznikala za přispění mnoha převážně internetových zdrojů, které jsou zmíněny na příslušných místech kódu.

# Příloha C: Pohled z úhlu geografických informačních systémů

Geografické informační systémy (GIS) jsou charakteristické modelem zemského povrchu, který bývá rozdělen do několika vrstev obsahujících data se stejným významem. Pomocí různých algoritmů je pak možno z GIS získávat nová data. Soubory vrstev a algoritmů závisí na konkrétních požadovaných výsledcích a mohou se v různých programech značně lišit. Typickými uživateli GIS je armáda a státní správa, ale také například v dnešní době značně medializované předpovědi počasí, změn klimatu, zasažení oblastí vlnami tsunami a lokálními záplavami.

Vrstvy mohou být buď vektorové nebo rastrové. Zatímco vektorové vrstvy obvykle zaznamenávají body a linie mezi nimi, jako například cesty nebo vedení elektrického napětí, rastrové vrstvy dělí prostor do sítě stejně velkých buněk, ve kterých se nějaká spojitá veličina mění pouze v menší míře, což bývá nejčastěji druh půdy, podloží nebo hodnoty nadmořské výšky.

Základem tohoto projektu jsou pak dvě rastrové vrstvy, z nichž ta první určuje nadmořskou výšku pro jednotlivé body a druhá ji pak překrývá s údajem reprezentujícím sloupec vody na daném bodě a jeho vlastnosti. Za poslední vrstvu pak lze považovat prameny, které už mají své souřadnice a tvoří tedy vrstvu vektorovou.

I když v současné podobě je praktické využití pro reálné předpovědi nepravděpodobné, protože hlavním cílem byla simulace a vizualizace výsledků v reálném čase a nikoli dokonalý fyzikální model, jsou poskytované výsledky pro některé experimenty a demonstraci základních jevů dostatečně dobré a velká interaktivita umožňuje jejich rychlé provedení. Systém tak může plnit funkci jakéhosi dema, které může velmi zhruba přiblížit funkci reálných GIS.