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

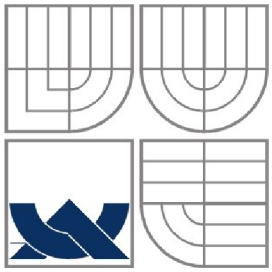
VIZUALIZACE HMATOVÝCH A ZVUKOVÝCH VJEMŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

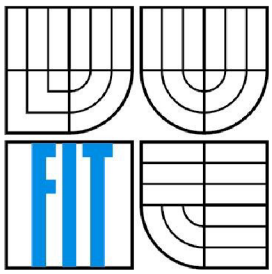
AUTOR PRÁCE
AUTHOR

ROMAN LUKŠ

BRNO 2014



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 HMATOVÝCH A ZVUKOVÝCH VJEMŮ

VISUALIZATION OF TACTILE AND AURAL SENSATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
ROMAN LUKŠ

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. LUKÁŠ POLOK

BRNO 2014

Abstrakt

Cílem této bakalářské práce je návrh a implementace vizualizace hmatových a zvukových vjemů běžících v reálném čase. Vjemy mají sloužit pro orientaci hráče v jednoduché 2D hře místo zraku. Vjemy jsou vizualizovány pomocí shaderů na GPU. Zvukové vjemy jsou implementací simulace vlnění výpočetně nenáročnou explicitní metodou. Hmatový vjem je tvořen pomocí pohyblivých čar. K vykreslování je použita knihovna OpenGL. V práci jsem vytvořil funkční herní prototyp obsahující dvojici vjemů.

Abstract

The purpose of this thesis is to design and implement visualization of aural and tactile sensations in the real-time. Sensations are to replace vision in simple 2D platform game. Visualizations are implemented on GPU. Vizualization of aural sensations implement computationally lightweight explicit method. Vizualization of tactile sensations are implemented as the animated lines. OpenGL is used to draw graphic output. Functional game prototype with vizualization of these two sensations is the output of this thesis.

Klíčová slova

nahrazení smyslů, vlnová rovnice, OpenGL, GLSL, hra, 2D, simulace vlnění, SAT, ping-pong algoritmus

Keywords

Sensory substitution, wave equation, OpenGL, GLSL, game, 2D, simulation of waves, SAT, ping-pong algorithm

Citace

Lukš Roman: Vizualizace hmatových a zvukových vjemů, bakalářská práce, Brno, FIT VUT v Brně, 2014

Vizualizace hmatových a zvukových vjemů

Prohlášení

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

……
Roman Lukš
Datum 14. ledna 2014

Poděkování

Rád bych poděkoval vedoucímu za rady, ochotu, trpělivost a pomoc s touto bakalářskou prací.

Obsah

1 Úvod

Cílem této práci je vytvořit jednoduchou 2D hru s vizualizací hmatu a sluchu. Oba tyto smysly převedeme na grafickou reprezentaci a umožníme je tak vnímat jiným smyslem - zrakem. Smysly použijeme k navigaci hráče v úrovních hry.

Jako základu pro vizualizaci sluchu použijeme zobrazení zvukových vln. Kromě fungující fyzikální simulace vycházející z vlnové rovnice budou zvukové vlny šířit také dodatečné informace. Může se jednat o identifikaci zdroje zvuku a další charakteristiky, které jsou špatně pozorovatelné pouze z pohledu na vizualizované zvukové vlny. Například tón hlasu, pokud bude zdrojem zvuku osoba apod. Dodatečné informace mohou být reprezentovány barvou vln nebo dalšími prvky umístěnými a vykreslenými přes vizualizaci zvukových vln (nápisy, textury a jiné grafické prvky, eventuálně další vlny). K simulaci vlnění se v praxi používají dva typy metod. Implicitní a explicitní. Při implicitních metodách je třeba řešit soustavu rovnic a stav simulace je závislý sám na sobě. U explicitních metod stačí z jednoho stavu vypočíst druhý [1]. V práci popíšeme a implementujeme explicitní metodou použitou v [2].

Hmat budeme vizualizovat jako nespojité čáry umístěné na povrch objektů, převážně však podlah. Hmatem vnímáme několik vlastností předmětů. Především se zaměříme na strukturu povrchu. Podstatné jsou také tvrdost (resp. měkkost) a teplota. Aby byla vizualizace názorná, musí ji tento soubor vlastností povrchu ovlivnit vhodným způsobem. Struktura bude mít vliv na tvar a amplitudu vln (sinusoida, čtvercovité, klikaté). Rozdílné tvrdosti znázorníme pomocí barev a teplota ovlivní rychlost pohybu čar.

Z výše uvedeného popisu je zřejmé, že budeme oba smysly vizualizovat podle jejich fyzikálních vlastností.

V první kapitole nejprve specifikujeme, jak mají vypadat hotové vizualizace vjemů. Následují kroky k odvození numerického řešení, kde vyjdeme z fyzikálních zákonů pro popis vlnění. Následuje rozbor matematických vztahů pro křivky. Druhá kapitola se zaměří na implementaci. Popíšeme použité technologie, strukturu programu, řešení práce s herními úrovněmi (textura herní úrovně, změna úrovně, použitý systém souřadnic) a manipulaci se shadery včetně popisu použitých shaderů. Budeme se zabývat detekci kolizí, generováním geometrie pro kolize, pohybem a ovládáním. Navrhneme datovou reprezentaci vjemu a funkce vhodné pro vizualizaci. Popíšeme si vlastní simulaci včetně překážek a dodatečného tlumení. Vysvětlen bude ping-pong algoritmus pro zapisování do textury. V závěru této části se zabýváme obarvením vlnění a generováním zdroje zvuku v shaderu. Další kapitola se týká implementace hmatového vjemu, kde si vysvětlíme implementované chování, rozdíl a typy shaderů pro popis povrchů a způsob definice povrchů v implementované struktuře. V závěru navrhneme možná budoucí rozšíření. V poslední kapitole provedeme vyhodnocení měření rychlosti simulace zvukového vjemu pro různá rozlišení simulované textury.

2 Teoretický úvod

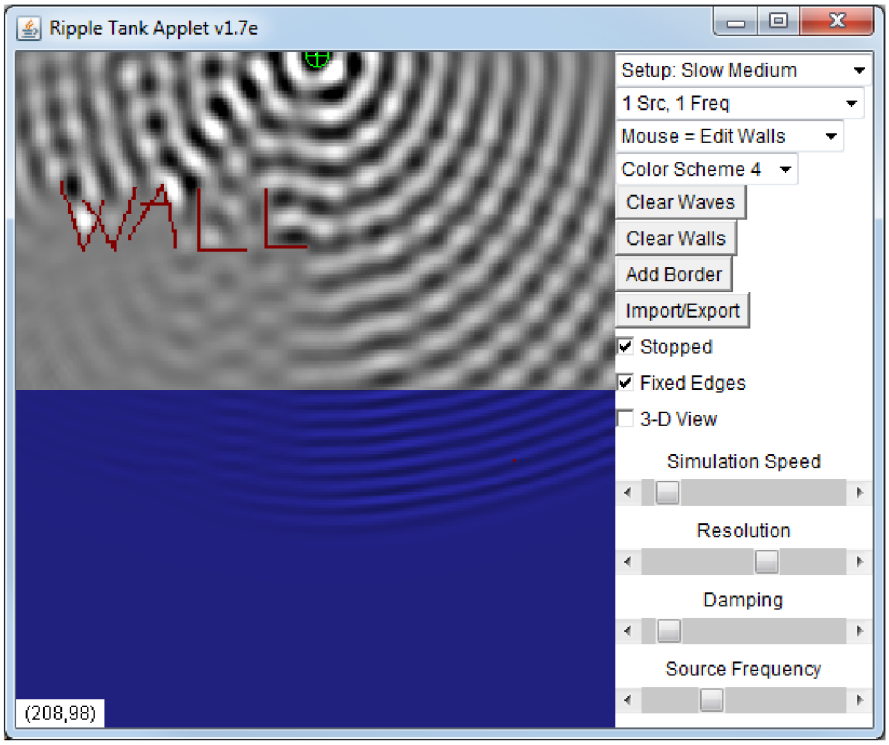
V této kapitole nejprve uvedeme požadavky na hotové řešení. To znamená, jaké jsou cíle vizualizací jednotlivých zvukových a hmatových vjemů. Poté se budeme zabývat numerickým řešením vizualizací a vhodnými simulačními metodami.

Použité zdroje pro psaní této kapitoly byly následující. Kniha zabývající se simulací kapalin a přenosem tepla [1], diplomová práce zkoumající simulaci kapalin v reálném čase [2], online nástroj pro simulaci vlnění [3], článek v časopisu SIGGRAPH na téma rychlých výpočtů chování kapalin [4], kniha zabývající se toky v mělkých vodách [5], článek o numerických výpočtech pro animaci vody [6], učebnice fyziky pro vysoké školy [7], kniha o numerickém řešení diferenciálních rovnic [8], učební text matematické analýzy [9], dvojice témat z online zdroje MathWorld [10] a [11], bakalářská práce s teorií křivek [12] a kniha o počítačové grafice [13].

2.1 Cíle vizualizace zvuku

Cílem vizualizace zvuku je poskytnout očím natolik názornou grafickou reprezentaci zvuku, aby byl pozorovatel schopen rozeznat všechny důležité informace, které by jinak rozeznal s použitím sluchu.

Základem je vizualizace zvukových vln. To znamená, že budeme graficky znázorňovat vznik a postupné zanikání zvukových vln. Zobrazíme, jak vlny vypadají, jak se šíří prostorem, jak se chovají, když narazí na překážku apod. Jedním z cílů práce je proto rozbor problematiky fyzikálního chování, návrh numerického řešení a implementace metody simulace chování zvukových vln.



Obrázek 2.1: Snímek zdroje vln (nahoře - zeleně) s překážkami „WALL“ (červeně) [3]

Nyní se podívejme na obrázek 2.1. Jaké vlastnosti vln můžeme pozorovat? Frekvenci (všimněte si její změny při vstupu vln do modrého prostředí), amplitudu (intenzita bílé barvy reprezentuje výšku vlny) a vlnovou délku (vzdálenost vrcholů sousedních vln). Jak se vlny chovají? Odrazy od překážek, postupné zanikání, rychlost šíření a neovlivňující se překrývající vlny. Můžeme pozorovat místo, kde se nachází zdroj zvuku, případně jeho pohyb.

Tyto informace ale nestačí k názorné vizualizaci sluchového vjemu. Vjem nese další informace. Informace, které jsou těžko pozorovatelné ze samotných vln. Takové informace musíme také vizualizovat. Je zdrojem například reproduktor? Co přehrává? Hudbu, mluvené slovo nebo zvuky? O jaký hudební žánr se jedná? Jakým tónem hlasu osoba mluví? Co říká? O jaké zvuky se jedná?

Jak by měla výsledná vizualizace vypadat, pokud chceme zachytit všechny podstatné detaily? Vhodným krokem je přidání několika barev k výchozí simulaci vln. Pro jednoduchost jsem se rozhodl použít jednu barvu pro zvuky tvořené hráčem, druhou barvu pro zvuky okolí a třetí označit východ z herní úrovně.

2.2 Metody simulace vlnění

V této kapitole budou popsány některé metody, které se používají pro simulaci vlnění. Používají se metody implicitní a explicitní. Implicitní metody vyžadují vyřešit soustavu rovnic a aktuální stav simulace je závislý sám na sobě. Explicitním metodám stačí z jednoho stavu vypočíst druhý pomocí numerické aproximace [1].

2.2.1 Rychlá, stabilní dynamika tekutin pro počítačovou grafiku

Simulační implicitní metoda [4], založená na soustavě parciálních derivací vycházející z rovnic pro mělkou vodu neboli 1-D Saint Venant. Rovnice mělké vody jsou odvozeny [5] z Navier-Stokesovy rovnice:

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla \mathrm{v}\right) = -\nabla \mathrm{p} + \nabla \cdot \mathrm{T} + \mathrm{f} \tag{2.1}$$

Tato metoda vychází z dvojice rovnic pro mělké vody, kde první vyjadřuje druhý Newtonův zákon a druhá zachování objemu:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + g\frac{\partial h}{\partial x} = 0 \tag{2.2}$$

$$\frac{\partial d}{\partial t} + u\frac{\partial (ud)}{\partial x} = 0 \tag{2.3}$$

Po aproximaci rovnic pro mělkou vodu pomocí lineární verze, která vynechává členy obsahující rychlosti a dalších úpravách dostáváme následující rovnici:

$$\frac{\partial^2 h}{\partial t^2} = g(d+h)\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right) \tag{2.4}$$

Tato rovnice je rozdělena na dvě části, kde je každá řešena samostatně a její výsledek použit k řešení druhé rovnice:

$$\frac{\partial^2 h}{\partial t^2} = g(d+h)\left(\frac{\partial^2 h}{\partial x^2}\right) \tag{2.5}$$

$$\frac{\partial^2 h}{\partial t^2} = g(d+h)\left(\frac{\partial^2 h}{\partial y^2}\right) \tag{2.6}$$

Výsledkem diskretizace je následující vztah:

$$A\mathrm{h}^{t+1} = f(h^t, h^{t-1}) \tag{2.7}$$

Kde A je matice závislá na $g,(d+ht),\Delta t,\Delta x$ a Δy. Matice A má pouze tři nenulové prvky v každé řadě. Matice je silně diagonálně dominantní, proto lze k řešení použít Jacobiho iterační metody (konverguje).

Prostorová náročnost této metody je následující. Prvky matice jsou uloženy ve třech RGB kanálech textury, pravá strana rovnice je v alpha kanálu. Další textury jsou současná a další výšková mapa a hloubková mapa. Metoda vyžaduje jednu texturu na uložení dočasných výpočtů. Pro N bodů je tak potřeba 8N hodnot v paměti.

Problém této metody je, že neobsahuje informace o rychlostech toků, která je pro naše účely důležitá z hlediska šíření vlnění.

2.2.2 Numericky efektivní a stabilní algoritmus pro animaci vln vody

Tato implicitní metoda [6] opět vychází z rovnic pro mělké vody, respektive z Navier-Stokesovy rovnice. Rozdíl je ale v použití nelineárních rovnic, které umožňují dynamickou změnu hloubky v čase. K integraci je použito numerické metody, která kombinuje výhody Eulerovy (pravidelnost) a Lagrangeovy (stabilita) metody – semi-Lagrangian. V případě Eulera je pozorovatel na místě, metoda však vyžaduje malé časové kroky, aby byla stabilní. Langrange přístup nechá pozorovatele cestovat s pohybující se částicí kapaliny. Tento přístup umožňuje větší kroky v čase.

K aproximaci prostorových výrazů se použije poloviční krok v obou směrech. Tato metoda je dána následujícím vztahem:

$$A\mathrm{h}^{t+1} = f(h^t, u^t, v^t, \mathrm{d}, \Delta\mathrm{t}, \Delta\mathrm{x}, \Delta\mathrm{y}) \tag{2.8}$$

Kde A je matice závislá na *g, d, h^t, Δt, Δx a Δy*. Matice A má tři nenulové prvky v hlavní a po dvou nenulových prvcích na vedlejších diagonálách. Matice je symetrická a pozitivně definitivní, proto je možné ji řešit pomocí iterativní metody CGM (conjugate gradient method).

Prostorová náročnost této metody je následující. Vyžaduje dvě RGBA textury a dvě poloviční RG textury pro uložení výpočtů metody CGM. Dále pak jedna RGBA textura a texturu s jedním kanálem pro koeficienty matice A a pravou stranu rovnice. K uložení hodnot u a v je třeba dalších textur s dvěma kanály. Pro uložení N bodů je tak třeba 17N+2√N hodnot.

2.2.3 Interaktivní simulace kapalin v reálném čase

Budeme potřebovat simulovat fyzikální vlastnosti a chování zvukových vln jako je šíření vln, slábnutí, odrazy, pohlcení, apod. Tuto explicitní metodu [2] určenou k simulaci kapalin je možné po změně některých konstant (viskozita kapaliny, tlumící člen, rychlosti vln) použít i pro simulace vlnění zvuku. Tuto metodu použijeme v práci při implementaci simulace vlnění.

Při simulování chování vln vyjdeme ze základní vlnové rovnice s Laplaceovým operátorem ∇^2 [7]:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \tag{2.9}$$

Po rozšíření o prostorové výrazy pro *x* a *y* a také tlumící člen dostáváme výraz:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \mu \frac{\partial u}{\partial t} \tag{2.10}$$

kde *t* je čas, *x* a *y* jsou prostorové souřadnice, *u* je funkce času a prostorových souřadnic, *c* je rychlost vlny, μ je tlumící člen.

Obecné řešení rovnice 1. řádu [8]:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.11}$$

My však potřebujeme provést výpočet numericky za použití počítače. K tomu využijeme možnost aproximaci tohoto výrazu pomocí dílčího kroku h [9]. Pokud se při praktickém použití ukáže, že chybovost této jednokrokové aproximace je příliš vysoká, použijeme metodu vícekrokovou.

Při numerické aproximaci řešení použijeme poloviční krok v obou směrech:

$$f'(x) = \frac{f(x+0.5h) - f(x-0.5h)}{h} \tag{2.12}$$

Pro řešení rovnice 2. řádu potom platí:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2.13}$$

A pro parciální diferenciály 2. řádu:

$$\frac{\partial^2 f}{\partial x^2}(x,y) = \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2} \tag{2.14}$$

$$\frac{\partial^2 f}{\partial y^2}(x,y) = \frac{f(x,y+h) - 2f(x,y) + f(x,y-h)}{h^2} \tag{2.15}$$

Po dosazení do rovnice získáváme tento výraz:

$$\frac{u_{t+1,x,y} - 2u_{t,x,y} + u_{t-1,x,y}}{(\Delta t)^2} = c^2\left(\frac{u_{t,x+1,y} - 2u_{t,x,y} + u_{t,x-1,y}}{(\Delta x)^2} + \frac{u_{t,x,y+1} - 2u_{t,x,y} + u_{t,x,y-1}}{(\Delta y)^2}\right) - \mu\frac{u_{t,x,y} - u_{t-1,x,y}}{\Delta t} \tag{2.16}$$

Protože krok v prostoru bude stejný pro obě prostorové souřadnice, lze výraz zjednodušit (zlomky prostorových výrazů budou mít stejné jmenovatele):

$$\frac{u_{t+1,x,y} - 2u_{t,x,y} + u_{t-1,x,y}}{(\Delta t)^2} = c^2\frac{u_{t,x+1,y} - 4u_{t,x,y} + u_{t,x-1,y} + u_{t,x,y+1} + u_{t,x,y-1}}{(\Delta x)^2} - \mu\frac{u_{t,x,y} - u_{t-1,x,y}}{\Delta t} \tag{2.17}$$

Po roznásobení zlomků a dalších úpravách dostáváme následující výraz:

$$u_{t+1,x,y} = \frac{c^2(\Delta t)^2}{(\Delta x)^2}\left(u_{t,x+1,y} + u_{t,x,y+1} - 4u_{t,x,y} + u_{t,x,y-1} + u_{t,x-1,y}\right) + (2 - \mu\Delta t)u_{t,x,y} + (\mu\Delta t - 1)u_{t-1,x,y} \tag{2.18}$$

Pro výpočet hodnoty jednoho čtverce vlny na souřadnicích *x*, *y* pro následující časový okamžik *t+1* budeme potřebovat hodnoty jeho okolí v čase *t*. Jedná se čtverce na souřadnicích *x+1*, *y+1*, *y-1*, *x-1*. Výpočet zahrnuje i předchozí hodnotu čtverce v čase *t*.

Vliv má také tlumící člen μ. Tlumící člen udává, jak rychle budou zvukové vlny slábnout s rostoucí vzdáleností od zdroje zvuku. Čím vyšší je tato hodnota, tím obtížněji se bude zvuk šířit okolím a rychleji slábnout.

V každém diskrétním kroku simulace spočítáme z aktuálních hodnot nové. Současné hodnoty si uschováme pro výpočet v příštím kroku a předchozí si uložíme kvůli výpočtu hodnoty tohoto členu:

$$(\mu\Delta t - 1)u_{t-1,x,y} \tag{2.19}$$

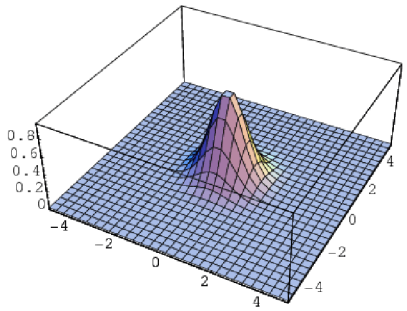
Posledním důležitým bodem výpočtu je určení hodnot překážek, od kterých se budou vlny odrážet. Pro každý čtverec takové překážky nastavíme jeho hodnotu na 0. To znamená, že pokud na ni vlna narazí, předá jí část své energie a odrazí se zpět. Hodnota překážky však zůstává nadále nulová.

Prostorová náročnost této metody je následující. K uložení N bodů je potřeba textura s hodnotami výšek hladiny, textura s hodnotami rychlostí vln *v* a *u* a textura hloubky resp. překážek. Je tak potřeba 7N hodnot v paměti.

2.2.4 Počáteční vlna

Jako počáteční tvar vytvářené zvukové vlny použijeme tvar vycházející z Gaussovy funkce pro dvě dimenze:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-[(x-\mu_x)^2+(y-\mu_y)^2]/(2\sigma^2)} \tag{2.20}$$



Obrázek 2.2: Gaussova funkce ve dvou rozměrech [10]

Jedná se o kruhovou Gaussovu funkci se standardní odchylkou rovnou pro *x* a *y*, která udává rozměry kuželu grafu. Hodnota zlomku určuje amplitudu. Protože chceme mít počáteční vlnu jako důlek, vynásobíme celý výraz hodnotou *-1*. Kužel tak bude směřovat vrcholem dolů.

2.3 Cíle vizualizace hmatu

Podobně jako u vizualizace zvuku musí i hmatový vjem poskytnout očím natolik názornou grafickou reprezentaci povrchu, aby byl pozorovatel schopen rozeznat všechny důležité informace, které by jinak rozeznal s použitím hmatu.

Uvažujeme, že hmatový vjem nám zprostředkovává tyto informace o povrchu: strukturu, tvrdost a teplotu. Tento vjem budeme vizualizovat jako pohybující se různé křivky na povrchu objektů. Vizualizace se musí zobrazit pouze při dotyku s objektem, protože cílem je nahradit hmat, kterým bez dotyku nelze vnímat.

2.3.1 Struktura

Nejdůležitější vlastností povrchů je pro nás struktura. Struktura bude definovat výchozí typ křivky. Základní typy mohou vycházet ze sinusoidy, čtverců nebo mohou být klikaté. Každý typ lze dále specifikovat délkou periody a velikostí amplitudy. Cílem je co nejlépe znázornit původní strukturu pouze za použití vlastností křivek. Základní rozdělení podle struktury bude vycházet z druhů materiálů. Například kovy, plasty, přírodní materiály, tekutiny, sklo, apod.

2.3.2 Tvrdost

Druh materiálu ovlivňuje jeho tvrdost. Pro rozlišení různých tvrdostí použijeme několika základních barev nebo šířku křivky. Další možné budoucí rozšíření je vizualizovat pnutí nebo tlak, který působí v místě dotyku s povrchem.

2.3.3 Teplota

Teplota je pro naše účely méně závislá na struktuře nebo tvrdosti a naopak. Vizualizace hmatu bude teplotu povrchů znázorňovat rychlostí, kterou se budou křivky pohybovat. Studenější objekty budou mít na svém povrchu pomalu se pohybující křivky. Naopak objekty s vysokou teplotou budou na svém povrchu ukazovat rychle se měnící křivky. Grafická reprezentace imituje rychlost kmitání molekul podle teploty dané látky.

2.4 Křivky

Tato kapitola pokládá teoretický základ ke křivkám. Rasterizací křivek získáme při implementaci základ pro hmatový vjem.

2.4.1 Bézierovy křivky

V počítačové grafice se nejčastěji používá parametrický tvar pro vyjádření křivek. Bézierovy křivky jsou aproximační křivky. Platí pro ně následující vztah [10]:

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{2.21}$$

Bézierova křivku stupně n je daná počtem $n+1$ bodů řídicího polygonu P_i. B_i^n jsou Bernsteinovy polynomy stupně n. Při parametrickém vyjádření se pohybujeme v intervalu $t \in \langle 0,1 \rangle$. Dosadíme-li okrajové hodnoty do rovnice (3.2), zjistíme, že křivka prochází prvním a posledním bodem řídicího polynomu. Tuto vlastnost využijeme při navazování dvou a více těchto křivek. Pro spojování křivek budeme vyžadovat pouze spojitost C^0 tj. identitu posledního bodu řídicího polygonu první křivky a prvního bodu řídicího polygonu druhé křivky. Výsledná křivka leží v konvexní obálce bodů řídicího polygonu. Pro naše účely je přínosem, že jsou Beziérovy křivky invariantní vůči lineární transformaci (otáčení, posunutí, atd.).

2.4.1.1 Rasterizace

Pro zobrazení na rastru je třeba vypočítat body křivky. K tomu použijeme rekurzivní algoritmus de Casteljau. Platí vztah [12]:

$$P_{j,i}(t) = (1-t)P_{j-1,i-1} + tP_{j,i-1} \tag{2.22}$$

Algoritmus pracuje s body řídicího polygonu. Postupně se z nich vypočítávají body, které se opět použijí pro výpočet dalších bodů. Pokud chceme, můžeme řídicí body rozdělit na pravé (R_i) a levé (L_i) a vypočítávat aproximaci křivky pomocí vzniklých úseček. V každém rozdělení vznikají dva nové řídicí polygony. Pro řídicí body platí:

$$L_i = \sum_{j=0}^{i} \binom{i}{j} \frac{P_j}{2^i} \tag{2.23}$$

$$R_i = \sum_{j=i}^{n} \binom{n-i}{n-j} \frac{P_j}{2^{n-i}} \tag{2.24}$$

Změna polohy jednoho řídicího bodu má vliv na tvar celé křivky. Kvůli této vlastnosti je lepší skládat složité křivky z více menších částí.

2.4.2 NURBS

Druhým typem křivek, které můžeme při vizualizaci křivek využit, jsou neuniformní racionální B-spline křivky. Jednou z výhod oproti Bézierovým křivkám je možnost přidání bodů řídicího polygonu místo zvyšování stupně polynomu. Díky tomu lze snadno ovlivňovat jen tu část křivky, kterou chceme. Pro NURBS křivku Q(t) platí [13]:

$$Q(t) = \sum_{i=0}^{n} P_i R_{i,k}(t) \tag{2.25}$$

Podobně jako Bézierovy křivky je i NURBS křivka určena *n+1* body řídicího polygonu P_i. Dále také řádem B-spline (označuje se písmenem *k*) a uzlovým vektorem. $R_{i,k}$, který označuje racionální B-spline bázi, pro kterou platí:

$$R_{i,k} = \frac{w_i N_{i,k}(t)}{\sum_{j=0}^{n} w_j N_{j,k}(t)} \tag{2.26}$$

Váhy bodů řídicího polygonu jsou označeny písmenem *w*. $N_{i,k}(t)$ jsou normalizované B-spline bázové funkce. Platí pro ně rekurentní vztah:

$$N_{i,1}(t) = \begin{cases} 1 & pro\ t_i \le t < t_{i+1} \\ 0 & jinde \end{cases} \tag{2.27}$$

2.4.2.1 Rasterizace

Při rasterizaci NURBS křivek jsou body získávány postupným dělením úseček části řídicího polygonu, kam daný bod náleží. Jedná se o Cox-deBoorův algoritmus.

NURBS jsou invariantní vůči transformacím a křivka leží v obálce řídicího polygonu. Pro naše účely je však nejdůležitější výhoda oproti Bézierovým křivkám - změna polohy nebo váhy řídicího bodu ovlivňuje tvar křivky pouze lokálně.

3 Implementace

V této kapitole se budeme zabývat návrhem a vlastní implementací. Nejprve se podíváme na použité nástroje a technologie, poté na popis herních úrovní, dále práci se shadery, implementaci herní logiky a fyziky. V závěru jsou dvě kapitoly věnovány implementaci zvukového a hmatového vjemu. V rámci kapitoly jsou také zmíněny problémy vzniklé při implementaci a jejich řešení.

Použité zdroje pro psaní této kapitoly jsou následující. Průvodce programováním v OpenGL [14], online dostupný dokument [15] zabývající se kolizemi a prezentace firmy Nvidia z conference o framebuffer objektech a jejich rozšíření [16].

3.1 Použité technologie

Práce byla implementována s použitím programovacího jazyka C++. Pro vykreslování grafiky bylo použito OpenGL ve verzi 3.0 a odpovídající verze GLSL 1.30. Byl použit dopředně kompatibilní kontext. Využito je také Windows API. K vývoji bylo použito vývojové prostředí MS Visual Studio 2012. Hra byla otestována na sestavě s grafickou kartou Nvidia GeForce GTX 285 s nainstalovanými ovladačemi ve verzi 320.57.

Pro práci s vektory a jejich předávání GLSL byla použita matematická knihovna GLM. Pro práci s texturami ve formátu PNG byla použita knihovna libpng. Pro práci s projekční maticí bylo použito OpenGL transform utils a Vector math v2.0 vyvinutá vedoucím této práce.

3.1.1 OpenGL

OpenGL specifikuje API pro vykreslování grafiky [14]. OpenGL bylo navrženo jako hardwarově nezávislé multiplatformní rozhraní. Implementace knihoven OpenGL je součástí ovladačů grafických karet. OpenGL neposkytuje funkce pro animaci, časování, práci se soubory, uživatelské rozhraní apod. Stará se pouze o vykreslování. K tomu nabízí funkce pro práci se základními primitivy jako jsou body, úsečky nebo mnohoúhelníky. OpenGL je stavový, to znamená, že programátor pomocí různých příkazů mění vnitřní nastavení stavů stroje.

Starší verze OpenGL disponovala pouze fixní pipeline. Programátor byl tak závislý na funkcích, které byly do OpenGL vestavěné. Mohl pouze nastavit určité hodnoty stavů stroje použitím funkcí OpenGL. Ale nemohl definovat vlastní funkce uvnitř pipeline. V moderním OpenGL je fixní vestavěná pipeline nahrazena použitím jazyka The OpenGL Shading Language (GLSL). GLSL poskytuje kromě primitivních datových typů jako jsou celá a desetinná čísla také datové typy vektorů a matic. Jako v jazyku C je také možné používat cykly, větvení apod. Lze použít funkcí pro absolutní hodnotu, goniometrické funkce apod. Tento způsob umožňuje programátorovi definovat si vlastní operace a výpočty prováděné v shaderech a nespoléhat na omezenou nabídku vestavěných funkcí.

Shadery jsou dvojice programů, které se syntaxí podobají jazyku C. První z programů, vertex shader, pracuje se souřadnicemi vrcholů, textur a také například velikostí bodů. Druhý program, fragment shader, aplikuje textury, barvy a provádí se zde široká škála dalších operací. Například vyhlazování, testování hloubky, průhlednost, maskování barev a další. Simulace zvukového vjemu probíhá převážně ve fragment shaderu. Naopak hmatový vjem je implementován z větší části ve vertex shaderech.

3.2 Struktura programu

Program je strukturován do několika částí. Hlavní smyčka programu se nachází v `Main`. Kromě inicializace zde také probíhá vykreslování hry, obsluha stisknutí kláves, změna herních úrovní. Ve třídě `Hero` je vykreslování herní postavy v závislosti na vnitřním stavu (chůze, skok). Zpracování kolizí, tvorba geometrie herních úrovní, výpočet pohybu na základě času snímku, vykreslování hmatového vjemu a pomocné vykreslování geometrie pro účely ladění se nachází ve zdrojovém souboru Physics. Třída `Shader` načítá po dvojicích (vertex a fragment) shadery z jejich zdrojových souborů a kompiluje je. Potomky této třídy jsou konkrétní shadery. Funkce pro práci s texturami jsou v souboru `Texture`. Instance struktury `Collision` jsou tvořeny, když je detekována kolize. Jednotlivé herní úrovně mají všechny informace uloženy ve struktuře `Level`. Informace o površích objektů jsou ukládány do struktury `Surface`. Jednoduchá struktura `Box` udržuje informace o geometrii objektů (kostky herních úrovní, herní postava, východ).

Jednotlivé textury ve formátu PNG jsou uloženy ve složce Textures. Soubory shaderů se nacházejí ve složce Shaders.

3.3 Herní úrovně

Jedná se o 2D hru, viděnou hráčem z boku, někdy také označovanou hovorovým termínem „plošinovka“. Hráč ovládá postavu na obrazovce, která se pohybuje doprava a doleva po herních úrovních. Postava skáče přes překážky. Cílem je dostat se v každé úrovni k východu, který vede do další úrovně. Herní postava je slepá. K navigaci v herních úrovních používá sluchu a hmatu.

K udržování informací o jednotlivých herních úrovních byla implementována struktura `Level`. Následuje kód struktury s vynecháním některých členů.

```
struct Level {
 glm::vec2 heroStartPosition;
 std::string mapFilepath;
 Box exit;

 std::vector<glm::vec2> soundSourcePositions;

 Level();
 Level(glm::vec2, std::string, Box);
 void addSoundSource(glm::vec2);
};
```

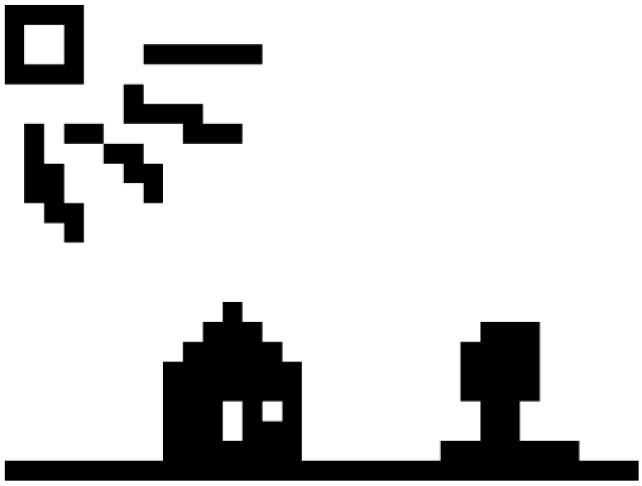
Příklad 3.1: Struktura Level

Najdeme zde následující položky. Souřadnice, kde se na začátku úrovně nachází herní postava, východ, vektor souřadnic zdrojů zvuku a cesta k textuře herní úrovně, která se použije jako hloubková mapa při simulaci vlnění a ke generování geometrie.

Změna úrovně vyžaduje načtení textury pro hloubkovou mapu, vygenerování geometrie, vyčištění textur s vlněním a přepsání souřadnic herní postavy.

3.3.1 Textura herní úrovně

Textury herních úrovní jsou uloženy ve formátu PNG. Jedná se o černobílé obrázky, kde bílá reprezentuje volné místo a černá překážky. Použité rozlišení je 32x24 pixelů. Tyto textury jsou použity ke dvěma účelům. Prvním využitím je načtení bitmapy do textury `depthmap`, která se používá při simulaci vlnění k udržování informací o překážkách. Textura je roztažena na použité rozlišení obrazovky. Herní svět se tak skládá z kostek a obdélníků odpovídajících pixelům z textury. Každá obrazovka herní úrovně odpovídá jedné textuře. Není třeba řešit simulaci na nekonečné ploše a pohyb kamery s herní postavou. Větší a komplexnější úrovně již budou vyžadovat simulaci na nekonečné ploše a pohyb kamery s herní postavou, proto je tato problematika vhodným budoucím rozšířením. Druhým účelem textur úrovní je jejich použití ke generování geometrie pro fyziku (kolize, pohyb herní postavy).



Obrázek 3.1: Textura herní úrovně po roztažení na rozlišení obrazovky (pro tyto účely byla změněna barva pozadí)

3.3.2 Systém souřadnic

V herních úrovních se pracuje se souřadným systémem, který má počátek v levém horním rohu obrazovky (viz. obrázek 3.2). Ypsilonová osa je vertikální a x roste horizontálně vpravo. Maxima odpovídají rozlišení textur herních úrovní tj. 32 a 24.

Obrázek 3.2: Použitý souřadný systém

K přepočtu na souřadný systém OpenGL s počátkem souřadnic ve středu obrazovky dochází až před vykreslením. K přepočtu je použit následující kód:

```
float s = (2.0f * x)/MAP_SIZE_X - 1.0f;
float t = 1.0f - (2.0f * y)/MAP_SIZE_Y;
```

Příklad 3.2: Přepočet souřadnic

3.4 Manipulace se shadery

Pro manipulaci se shadery je implementována struktura `Shader` a pomocné funkce. Zdrojové kódy vertex a fragment shaderů jsou uloženy v samostatných souborech *.vx.shader respektive *.fr.shader. K načítání řetězců kódu z těchto souborů je implementována funkce `getTextFileString()`. Struktura `Shader` obsahuje obecnou funkci pro kompilaci shaderu do programu spustitelného na GPU. Následuje kód struktury s vynecháním některých členů.

```
struct Shader {
 bool compile(GLuint *, GLuint *, GLuint *);
};
```

Příklad 3.3: Shader

Všechny konkrétní shadery jsou pak potomky této struktury. Protože každý shader potřebuje jiné parametry, tak se i liší parametry funkcí `bind()` jednotlivých shaderů. Tyto funkce volají OpenGL funkci `glUseProgram` s daným programem shaderu a nastavují různé proměnné `Uniform`.

Během implementace se vyskytl problém s předáváním souřadnic vrcholů a textur, kde byla vždy použita souřadnice textury. Řešením je při vytváření pole vrcholů nastavit správnou hodnotu indexu ve funkci `glEnableVertexAttribArray()` respektive `glVertexAttribPointer()`.

3.4.1 Použité shadery

Zda krátce představíme shadery, které byly implementovány a použity v programu:

- `ShaderBlur` – první implementovaný shader, určený pro testování, provádí rozmazání textury
- `ShaderPass` – shader vykreslující texturu simulace zvukového vjemu na obrazovku
- `ShaderHero` – stará se o vykreslení herní postavy na obrazovku
- `ShaderVelocity` – provádí první krok jedné iterace simulace zvukového vjemu, tj. výpočet rychlostí vln
- `ShaderHeight` – provádí druhý krok jedné iterace simulace, tj. výpočet výšky vln
- `ShaderPoint` – přidává počáteční vlnu zdroje zvuku do textury simulace zvukového vjemu

- `ShaderPointSimple` – shader implementovaný kvůli testování vykreslí krajní body kolize
- `ShaderBoxes` – vykreslí geometrii, určeno k testování
- `ShaderSurface` – vykreslí hmatový vjem

3.5 Implementace herní logiky a fyziky

V této kapitole se budeme zabývat některými problémy při implementaci a jejich řešení, které souvisí s herní logikou a fyzikou. To zahrnuje generování geometrie, detekci kolizí, pohyb a ovládání.

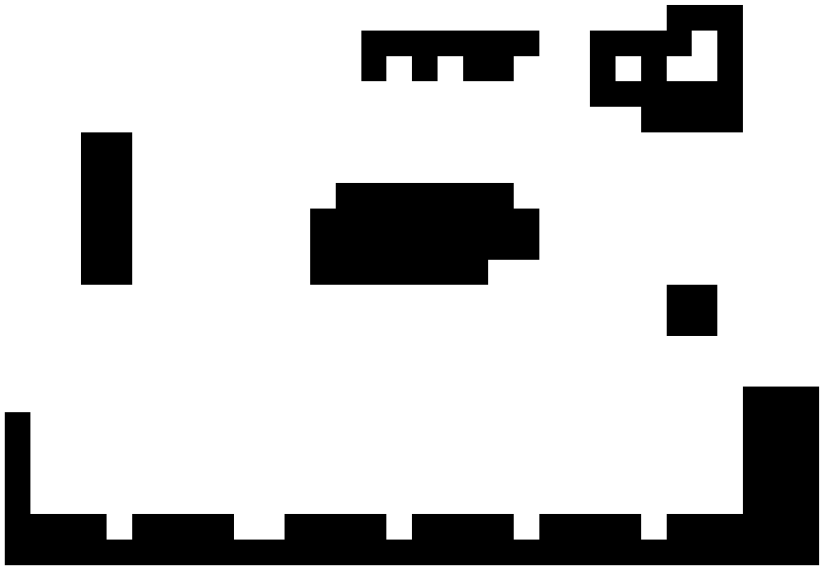
3.5.1 Generování geometrie

Pro generování geometrie je použito textury herní úrovně. Obrázkem je iterováno jako polem a tam, kde se nachází černé pixely, je vytvořena instance struktury `Box` s odpovídajícími souřadnicemi. Tato struktura udržuje informace o souřadnicích levého horního a pravého spodního vrcholu každé kostky resp. pixelu. Následně jsou některé kostky, které vhodným způsobem sousedí s dalšími kostkami, sloučeny do obdélníků. Sloučení probíhá ve dvou fázích. Nejprve na délku, poté na výšku. Výsledná geometrie (množina instancí struktury `Box`) je umístěna do vektoru. Následuje kód struktury `Box`.

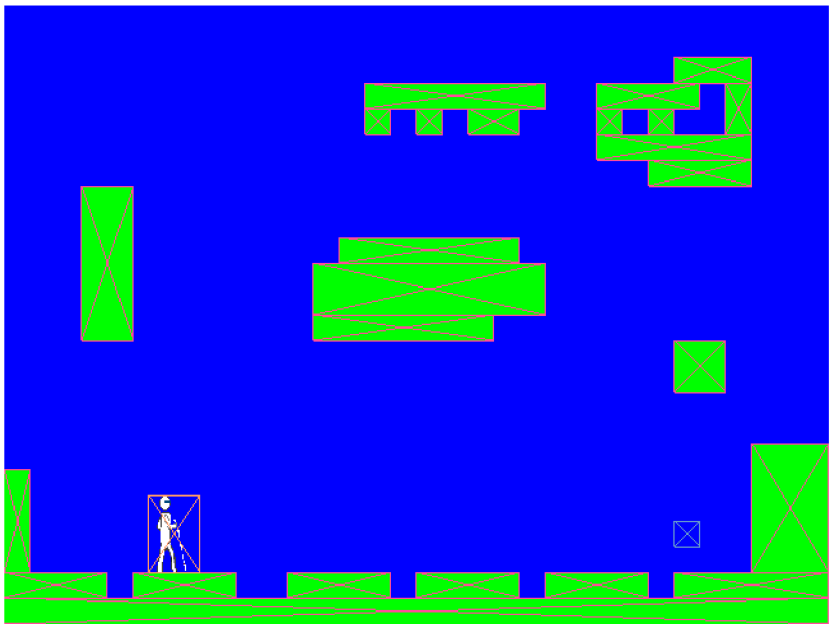
```
struct Box {
 glm::vec2 vMin, vMax;
 Surface surface;
};
```

Příklad 3.4: Struktura Box

Kromě souřadnic vrcholů obsahuje struktura také instanci struktury `Surface`. Následují dva obrázky s ukázkou vygenerování geometrie.



Obrázek 3.3: Černobílá textura herní úrovně



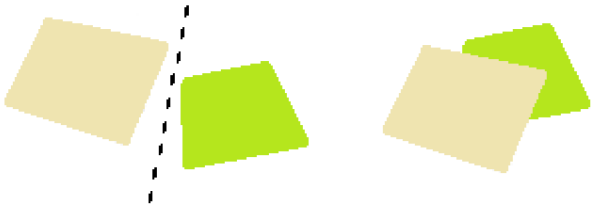
Obrázek 3.4: Stejná herní úroveň se zobrazenou vygenerovanou geometrií (zeleně) včetně herní postavy

Na obrázku 3.4 je dobře vidět způsob spojování vhodných kostek. Například vpravo nahoře jsou kostky nejprve spojeny horizontálně a až poté je utvořen jediný vyhovující obdélník orientovaný na výšku. Pro komplexnější herní úrovně bude vhodné budoucí rozlišení optimalizace spojování kostek, aby jich bylo co nejméně. Algoritmus tak bude procházet menším množství objektů a tím dojde ke zrychlení detekce kolizí.

3.5.2 Kolize

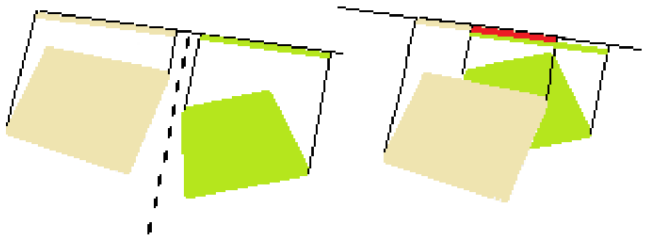
K detekci kolizí je implementována zjednodušená varianta věty o dělící ose (Separating Axis Theorem [15] – SAT). Kolize jsou zjišťovány pro instanci struktury `Box` patřící herní postavě se zbytkem geometrie (množinou struktur `Box` vygenerovaných z textury herní úrovně).

Věta o dělící ose říká, že pokud máme rovinu a v ní dvě konvexní množiny, pak se tyto množiny neprotínají (nekolidují), pokud existuje přímka, která dělí rovinu tím způsobem, že v každé polovině leží jen jedna z množin. Konvexní množinou lze vést úsečku mezi body okrajů. Jinými slovy, jedná se o objekt bez vykrojení a výřezů.



Obrázek 3.5: Vlevo existuje dělící přímka (bez kolize), vpravo neexistuje (kolize)

Varianta SAT používaná pro detekci kolizí pracuje s projekcemi objektů na množinu vhodných os. Osy jsou rovnoběžné normálami stěn objektů. Každý objekt je promítnut na tyto osy. V každé ose se testuje překrytí průmětů objektů. Pokud se průměty objekty překrývají ve všech dělících osách, tak došlo ke kolizi. Nalezne-li algoritmus alespoň jednu osu, kde se průměty nepřekrývají, tak končí a závěrem je, že objekty nekolidují.



Obrázek 3.6: Vlevo průměty bez průniku (bez kolize), vpravo se průměty protínají (kolize)

Pro účely této práce lze detekci kolizí pomocí SAT zjednodušit. Není nutné z objektů zjišťovat normály, protože máme pouze kostky a obdélníky rovnoběžné s osami x a y. Místo promítání objektů na osy také pouze stačí kontrolovat překrytí souřadnic jejich vrcholů.

Výpočet minimálního posuvného vektoru (Minimum Translation Vector – MTV), který je důležitá součást SAT, je zachován. Použije pro posunutí kolidujících objektů o nejmenší možný vektor, kde už nedochází ke kolizi. MTV je rozdíl souřadnic překrývajících se vrcholů v ose, kde je délka průniku dvou objektů nejmenší. Tato hodnota je pak aplikována na posun souřadnic herní postavy. Po výskytu kolize je vytvořena instance struktury `Collision`. Najdeme zde, o jaký typ kolize se jedná (zprava, zleva, zvrchu, zdola), krajní body kolize a typ povrchu. Tyto informace jsou poté použity pro eventuální vykreslení hmatového vjemu.

Následuje kód struktury `Collision` s vynecháním některých členů.

```
struct Collision {
 collisionType type;
 glm::vec2 point1;
 glm::vec2 point2;
 Surface surface;
};
```

Příklad 3.5: Struktura Collision

Implementovaná varianta detekcí kolizí má problémy s detekcí rychle se pohybujících objektů (např. rychle padající herní postava). Je to dáno tím, že v každé smyčce vykreslení jsou kolize detekovány pouze jednou. Následně se mění pohyb aplikovaný na herní postavu. V další smyčce vykreslení je již herní postava posunuta mimo herní úroveň nebo dovnitř překážky. Vhodným budoucím rozšířením je tak implementovat detekci kolizí, aby běžela nezávisle na smyčce vykreslování.

3.5.3 Pohyb a ovládání

K vyčítání detekcí stisknutí kláves je implementována struktura `Keys`, která obsahuje pravdivostní hodnoty ke každé z kláves ovládající hru. Hodnoty v této struktuře jsou nastavovány asynchronně pomocí funkce pro obsluhu zpráv. Hra se ovládá směrovými klávesami. Pro orientaci v herní úrovni slouží možnost úderu hůlkou o zem. Od místa úderu se šíří zvukový vjem, který odhaluje překážky a geometrii herní úrovně. Úder je mapován na klávesu mezerníku. Restart herní úrovně se provede stisknutím klávesy 0 na numerické klávesnici.

Na základě stisknutých směrových kláves je určen směr pohybu, uražená vzdálenost se vypočítá jako rozdíl času vykresleného snímku a rychlosti v dané ose. Pro skákání je použito dočasné zapínání gravitace. To znamená, že tato síla působí pouze, když se postava nachází ve vzduchu, tj. nekoliduje se zemí. Pro pády z větších výšek bylo přidáno omezení na maximální rychlost v ose y, protože implementovaný systém kolizí nebyl schopen tak rychlý objekt detekovat. Je to zřejmě způsobeno tím, že kolize se detekuji pouze jednou za každý snímek. Zde je prostor pro zlepšení – např. detekce kolizí běžící na fixních časových úsecích asynchronně ke smyčce s vykreslováním.

Informace o herní postavě jsou uloženy ve struktuře `Hero`. Najdeme zde definici rozměrů, geometrii, rychlost, pozici, informace o pohybu, proměnné určené pro vykreslování (vrcholy, vertex buffer objekty, shader, atd.). Následuje kód struktury `Hero` s vynecháním některých členů.

```
struct Hero {
 static const int height = 3;
 static const int width = 2;

 ShaderHero heroShader;
 GLuint heroVBO, heroVAO, heroIBO;
 GLuint hero_tex, hero_tex2;

 Box box;

 glm::vec2 velocity;
 glm::vec2 position;
```

```
    static bool jumping;
    static bool moving;
    bool facingLeft;

    static float p_hero_vertices[];
    static unsigned int p_hero_indices[];

    void init();
    void draw();
};
```

Příklad 3.6: Struktura Hero

Při inicializaci jsou načteny textury a vygenerovány všechny potřebné OpenGL objekty. Vykreslování používá jednoduchou animaci chůze, také se kontroluje orientace postavy. V shaderu dochází k přepočtu souřadnic a následně se pouze aplikuje textura. Je také aplikován blending.

3.6 Implementace zvukového vjemu

V této kapitole se budeme zabývat implementací zvukového vjemu. To znamená tématy jako jsou datová reprezentace vln, překážky, implementace simulace, přidání dodatečného tlumení, algoritmus ping-pong, obarvování různých zdrojů vlnění a generování počátečních vln zdrojů zvuku. V kapitole 3.8 vyhodnotíme a změříme rychlost simulace implementovaného řešení v několika rozlišeních.

3.6.1 Datová reprezentace vln

Zvukové vlny musí být v paměti reprezentovány takovým způsobem, abychom s nimi mohli jednoduše pracovat. Spojité zvukové vlny musíme pro potřeby výpočtu diskretizovat. Použijeme dvourozměrnou matici. K reprezentaci na GPU použijeme texturu s více kanály. Budeme potřebovat tři. Jeden kanál sloužící k uložení výšky hladiny, další dva kanály potom k uložení rychlostí vln v osách x a y. Počáteční vlnu budeme předpočítávat v shaderu a přičítat k textuře zvukové vlny na souřadnicích zdroje.

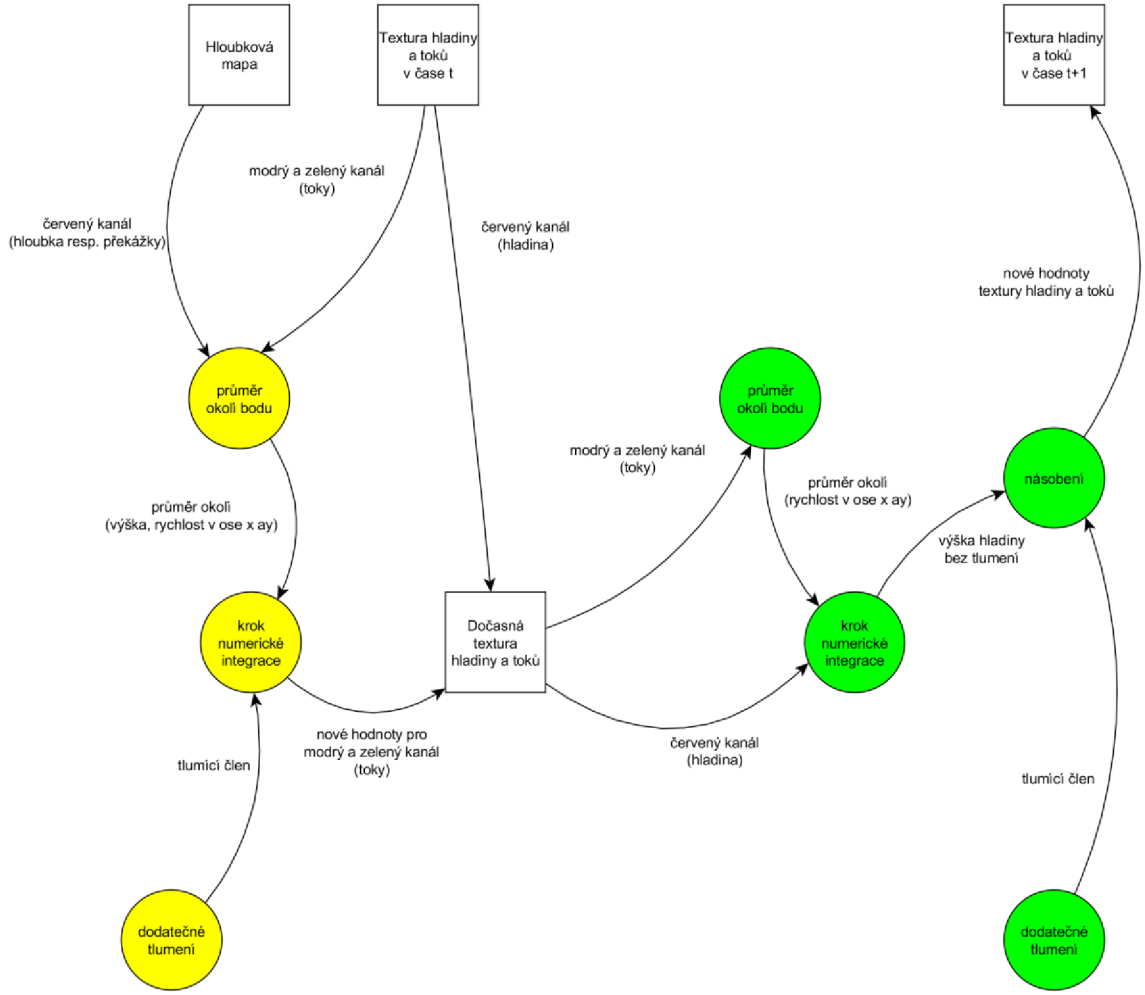
Pro další informace, které mohou zvukové vlny šířit, použijeme další vhodné datové typy. Uložíme si také souřadnice a další vlastnosti zdroje. Celou množinu dat jednoho zdroje zvukových vln umístíme do struktury.

3.6.2 Překážky

Informace o překážkách jsou při inicializaci nebo změně herní úrovně nahrány do textury `depthmap`. Na texturu je aplikováno filtrování `GL_NEAREST` kvůli zachování ostrosti při roztažení. Jako vstup je použit černobílý obrázek. Je možné použít barevný, ale odrazy jsou pak pro některé barvy slabé. Hodnoty v této textuře se vyhodnocují následujícím způsobem. Pokud má pixel hodnotu 0x00, jedná se o překážku, v opačném případě je nastavena hloubka na hodnotu odpovídající hodnotě pixelu. Pokud je pixel bílý, je výsledná hloubka 1.0. Pro každý bod vlny je pak vyhodnocena hloubka nejbližšího okolí. V případě, že se jedná o vlnu narážející do překážky, je mu nastavena odpovídající hodnota rychlosti vlny. Bodu vlny, který by se nacházel na překážce, je ostře nulována výška i rychlosti.

3.6.3 Vlastní simulace

Vykreslujeme pouze 1 čtverec složený ze dvou trojúhelníků roztažený přes celou obrazovku skládající se ze čtyř vrcholů. Simulaci provádí dvojice shaderů. První počítá rychlosti vlny, druhý výšku hladiny. Pro zrychlení simulace se shadery volají několikrát v cyklu za dobu vykreslení jednoho snímku. Shadery potřebují dvojici textur. První textura se třemi kanály obsahuje výšku hladiny v červeném kanálu, rychlost pohybu vln v ose y v zeleném kanálu a rychlost pohybu vln v ose x v modrém kanálu. Druhá textura je již dříve zmíněná hloubková mapa reprezentující překážky, od kterých se budou vlny odrážet.



Obrázek 3.7: Diagram simulace vlnění zvukového vjemu (VelocityShader žlutě, HeightShader zeleně)

Shader pro výpočet rychlostí interpoluje výslednou rychlost ze stavu okolí daného bodu a diskrétních časových kroků. Přidáno bylo také dodatečné tlumení, kvůli zpřehlednění vlnění pro hráče. Shader pro výpočet výšky také interpoluje výslednou výšku hladiny ze stavu okolí daného bodu. Opět je přidáno dodatečné tlumení. Výpočty probíhají ve fragment shaderech, vertex shadery pouze přeposílají souřadnice.

Během implementace se vyskytl problém s posunováním celé simulace po úhlopříčce. Řešením bylo použít správné okolí bodu. To znamená, místo bodů *(x+1, y)*, *(x, y+1)* a *(x+1, y+1)* body *(x-1, y)*, *(x+1, y)*, *(x, y+1)* a *(x, y-1)*.

3.6.4 Dodatečné tlumení

Do simulace bylo přidáno dodatečné tlumení, které působí pouze na okrajích obrazovky. Důvodem bylo příliš velké množství odrazů a starých vln, které se v herní úrovni simulovaly. Velké množství slábnoucích starých vln znepřehledňovalo situaci a znesnadňovalo orientaci hráče v herním prostředí. Během implementace se ukázalo, že je třeba, aby tlumení působilo pozvolna. Cílem je zamezit vzniku velkých rozdílů v hladinách a tak i vzniku následných artefaktů. Tlumení je implementováno v GLSL krom jiného pomocí funkce `smoothstep()`.

3.6.5 Ping-pong algoritmus

Ping-pong algoritmus je technika, která se používá v počítačové grafice pro práci se vstupními a výstupními texturami [14] resp. pro vykreslování do textury. Pokud má shader pracovat s texturou, mění její hodnoty a v další iteraci potřebuje pracovat se změněnou texturou. Proto je nutné použít dvojici textur, nelze totiž zároveň číst a zapisovat do stejné textury. V praxi existují tři přístupy [15], jak implementovat algoritmus ping-pong. Lze připojovat různé framebuffer objekty nebo mít více připojených textur pomocí OpenGL příkazu `glDrawBuffer`. V naší práci použijeme třetí způsob, kde se připojuje v každé iteraci různá textura k framebuffer objektu. V případě implementace zvukového vjemu použijeme techniku ping-pong pro práci s texturou udržující informace o výšce hladiny a rychlostech vln v osách x a y.

Následuje zjednodušený pseudokód techniky ping-pong, který je v práci implementován:

```
bind framebuffer(fbo)
framebuffer texture(tex[1-src])
viewport
clear
bind texture(tex[src])
draw
bind framebuffer(0)
bind texture(tex[1-src])
generate mipmap
src=1-src
```

Příklad 3.7: Ping-pong algoritmus

Nejprve je nastaven správný shader a framebuffer. Poté je k nastavenému framebufferu připojena cílová textura. Následuje nastavení obdélníku viewport (zde na rozměry použité textury) a čištění bufferů. Následně se nastaví vstupní textura pro shader a volá se příkaz pro vykreslení. Poté je zrušeno připojení framebuffer objektu. Pro vytvoření mip map je připojena textura, do které bylo zapisováno a nakonec se prohodí indexy zdroje a cíle.

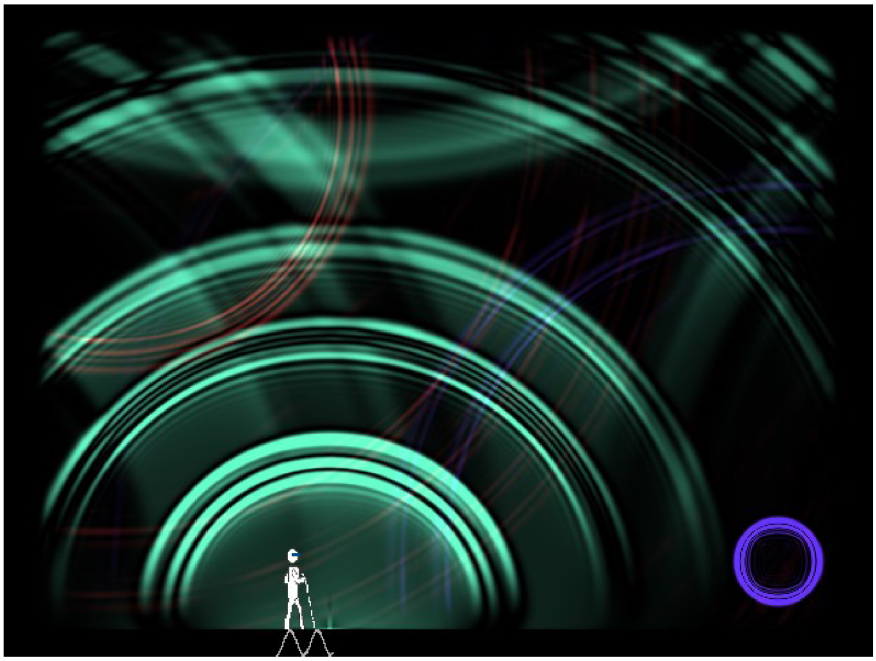
Pro správnou funkčnost tohoto algoritmu je nutné při inicializaci vygenerovat framebuffer a specifikovat seznam bufferů pro výstup barev s použitím OpenGL příkazu `glDrawBuffers`, kde použijeme `GL_COLOR_ATTACHMENT0`.

3.6.6 Obarvení vlnění

Pro účely hry je třeba rozlišit různé zdroje zvukových vjemů. Existují dva přístupy jak obarvení dosáhnout. První možností je použít další texturu s třemi kanály, kde každý bod reprezentuje barvu hladiny na dané souřadnici. Během simulace pak vzájemně míchat tyto barvy. Výhodou tohoto způsobu je jediná textura, kterou simulujeme a interakce vln od různých zdrojů. Tento způsob ale

není pro naše účely vhodný, protože smícháním barev nám vzniknou nové, které nebudou mít pro hráče vypovídající význam. Chceme ostře rozlišit typy zdrojů zvuku. Druhou možností, kterou v práci implementujeme, je vytvoření samostatné simulované textury pro každý typ zdroje. Obarvení je aplikováno až při vykreslování simulované textury na obrazovku. K nevýhodě této metody patří paměťová náročnost, kde pro každý typ zdroje s jinou barvou potřebujeme přidat texturu, kde budeme simulovat vlnění. V našem případě se ale jedná jen o pár jednotlivých typů zdrojů.

Během implementace se také ukázalo, že množství starých slabých vln vytváří nepřehlednou situaci a nové vlny jsou mezi ostatními špatně vidět. Do shaderu pro zobrazování je tedy přidáno zesílení a zvýraznění nových vln. Slábnoucí starší vlny jsou naopak zeslabeny. Tento způsob velice pomáhá usnadňovat hráči orientaci v úrovních a navíc neovlivňuje hodnoty v textuře pro simulaci. Na následujícím snímku je vidět, že se jednotlivě zbarvené vlny neovlivňují, protože se každá simuluje v samostatné textuře. Texturu s hloubkovou mapou mají však společnou.



Obrázek 3.8: Různě barevné vlny se neovlivňují (samostatné textury)

3.6.7 Generování zdrojů zvukových vjemů

Vytvoření zdroje zvuku se skládá ze dvou částí. Nejprve jsou určeny souřadnice zdroje a poté je pomocí shaderu vygenerována počáteční vlna.

Souřadnice zdroje lze určit z několika různých hodnot. V případě zdrojů, které jsou součástí herní úrovně, jsou jejich souřadnice uloženy ve vektoru souřadnic. Souřadnice zdroje zvuku východu z úrovně jsou vypočítány ze struktury geometrie východu. Dalším zdrojem zvuku jsou akce herní postavy, kterou hráč ovládá. Souřadnice těchto zdrojů zvukových vjemů jsou získány z geometrie herní postavy.

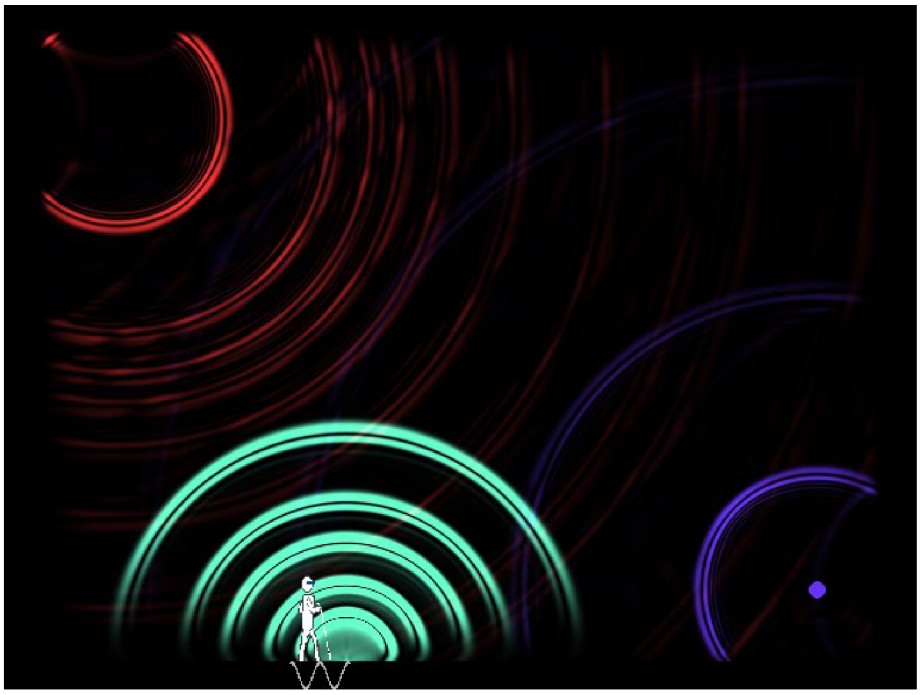
Generování počáteční vlny probíhá následujícím způsobem. Nejdříve proběhne inicializace objektů, nastavení správné cílové textury, připojení shaderu a samotné vykreslení. Použit je také blending k přičtení počáteční vlny k textuře vlnění. V shaderu se pouze počítá vzdálenost od středu zdroje zvuku a dle této vzdálenosti je použita odpovídající intenzita barvy. Jedná se o výpočetně nenáročnou aproximaci tvaru vycházející z Gaussovy funkce pro dvě dimenze z kapitoly 2.2.4.

Zvažována byla také možnost předpočítat si jednu texturu s malými rozměry obsahující tvar počáteční vlny zdroje zvuku a tuto pouze připočíst na požadovaných souřadnicích k textuře hladiny

simulace vlnění. Tato možnost byla zavržena kvůli velkému množství přístupu do paměti pro jedinou počáteční vlnu zdroje zvuku.

Během implementace byla nejprve vytvořena CPU verze vkládání počáteční vlny zdroje zvuku ve funkci `n_GLTextureRGBFloatFromData()`. Tato funkce byla zprvu použita pro vytvoření textury obsahující jednu počáteční vlnu zdroje zvuku, která byla následně nahrána do GPU a rozpohybována použitou simulací vlnění.

Počáteční vlny jsou v místě zdrojů umísťovány neustále, míru frekvence vkládání ovlivňuje generování náhodných čísel. Následuje ukázka různých zdrojů.



Obrázek 3.9: Ukázka zdrojů zvuků, vlevo nahoře zvuk prostředí, vpravo dole východ, uprostřed zvuk od úderu slepeckou holí

3.7 Implementace hmatového vjemu

V této kapitole se budeme zabývat implementací hmatového vjemu. Vysvětlíme použití hmatového vjemu. Objasníme strukturu definic povrchů a použití shaderů povrchů. Vysvětlíme, jak funguje vizualizace hmatu v praxi.

3.7.1 Chování

Hmatový vjem je zobrazen na podlaze pouze při dotyku (kolizí) herní postavy s touto podlahou. Šířka hmatového vjemu odpovídá šířce dotyku herní postavy se zemí. Krajní body vjemu jsou tedy nejlevější a nejpravější místo kolize. Pokud se herní postava nachází na okraji podlahy, je hmatový vjem kratší. Toto chování hmatového vjemu má hráči usnadnit pohyb v úrovních, kde je třeba opatrnosti. Snadno tak pozná, když se blíží k okraji.

3.7.2 Definice povrchů

K uchovávání informací o jednotlivých typech povrchů jsme definovali strukturu `Surface`. Obsahuje informace o barvě lomených čar, které budou reprezentovat hmatový vjem. Dále zde máme frekvenci a amplitudu, které slouží k modifikaci tvaru těchto čar. Důležitá je také teplota, která určuje rychlost pohybu čáry. Každá instance struktury má také přiřazen vhodný shader. Jedním ze způsobů, jak můžeme dosáhnout různých křivek, je větvení uvnitř jednoho obecného shaderu, který dle typu křivky vykreslí výsledný hmatový vjem. V této práci je ale použito několika různých shaderů pro každý různorodý typ povrchu. Následuje kód struktury `Surface` s vynecháním některých členů.

```
struct Surface {
 glm::vec3 color;
 float frequency;
 float amplitude;
 float temperature;

 ShaderSurface surfaceShader;
};
```

Příklad 3.8: Struktura Surface

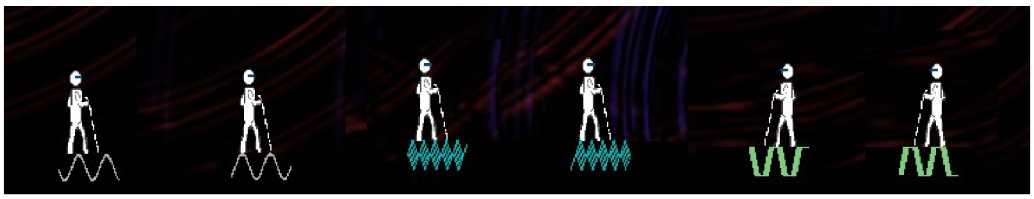
3.7.3 Shadery povrchů

`ShaderSurface` pro vykreslování povrchů se mírně liší od ostatních použitých shaderů, protože při kompilaci je nutné předat cestu k souboru. V každém z různých souborů shaderů se však používají stejné parametry, takže použití je již stejné.

3.7.4 Vizualizace hmatu

Implementovány jsou tři různé typy povrchů. Prvním je vykreslení hmatové vjemu jako funkce sinus. Dalším typem povrchu je hmatový vjem v podobě klikaté lomené čáry. Poslední je potom vjem připomínající digitální signál. Použití se neliší, každý shader implementuje následující algoritmus.

Vizualizace hmatu funguje následujícím způsobem. Jako vrcholy se do vertex shaderu posílají pozice v desetinných číslech v rozsahu od 0 do 1. Pro každý vrchol se pak interpoluje z krajních bodů kolize jeho pozice. Každý shaderu hmatového vjemu svým způsobem nastavuje výšku bodů v ose y. V každé iteraci se také podle dané teploty body posunují po ose x. Při vyšších teplotách se tak hmatový vjem pohybuje rychleji. Ve fragment shaderu se pouze nastaví požadovaná barva. Následuje ukázka hmatového vjemu.



Obrázek 3.10: Ukázka vizualizace hmatového vjemu

Máme tedy množství kombinací, jakých hmatových vjemů chceme docílit. Můžeme si vybrat jeden z typů, nastavit libovolnou barvu, rychlost kmitání, amplitudu. K dispozici je celá škála variant.

Původně byla zamýšlená varianta implementace hmatového vjemu pomocí rasterizace Bézierových a NURBS křivek. Použitý přístup je však snazší na implementaci a výsledek je vizuálně uspokojivý.

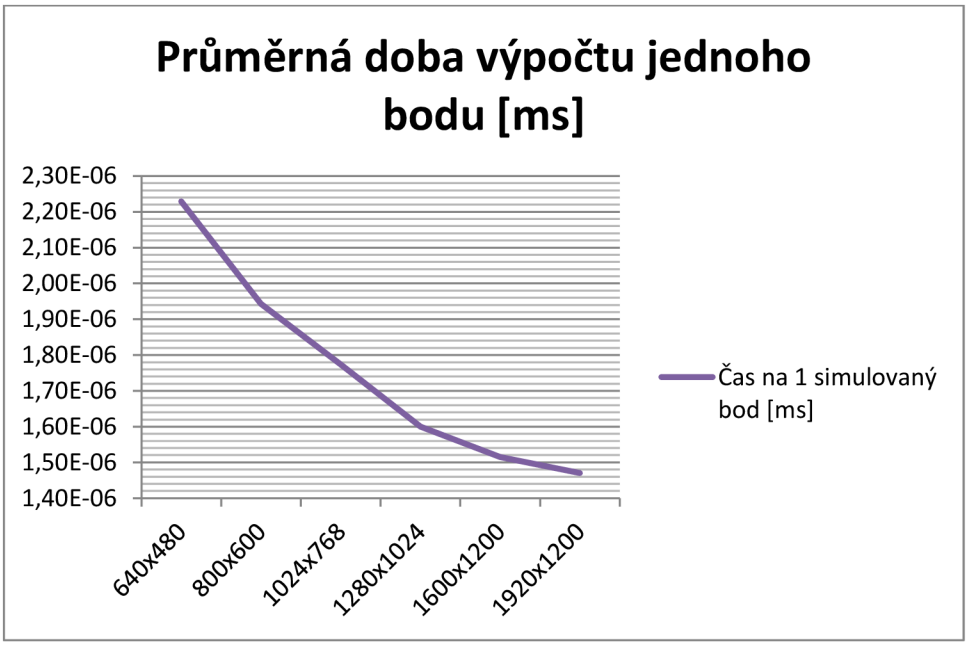
3.8 Vyhodnocení

Měření simulace zvukového vjemu bylo provedeno pro 10 000 iterací pomocí funkce `simulateSoundMeasurement()`. Pro měření byla použita hardwarová konfigurace popsaná v kapitole 3.1. Následuje tabulka s naměřenými hodnotami.

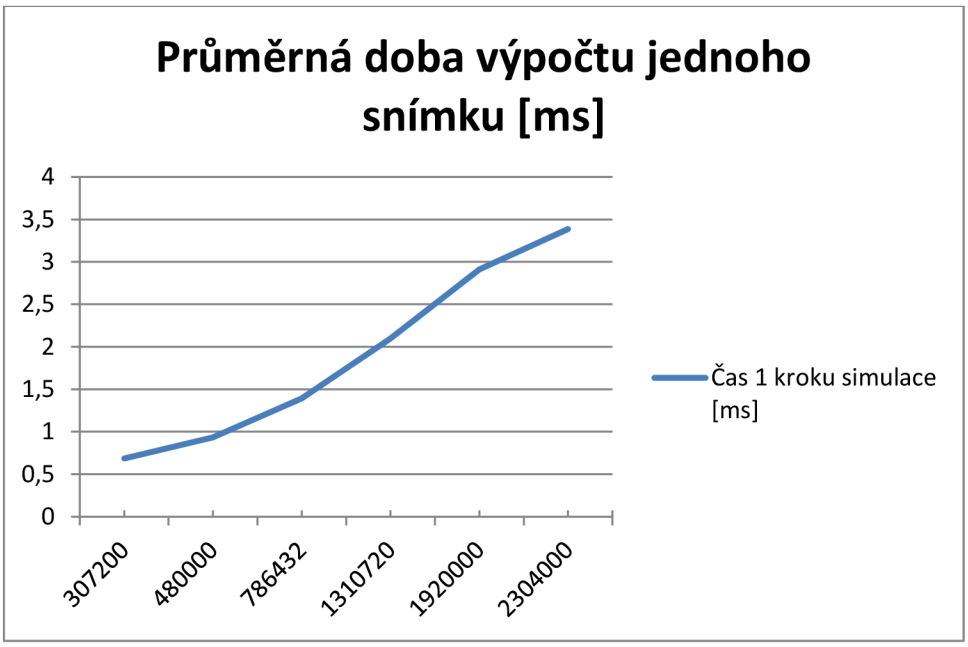
Rozlišení [px]	Počet bodů	Průměrná doba výpočtu jednoho snímku [ms]	Průměrná doba výpočtu jednoho bodu [ms]
640x480	307200	0,6849	2,22949E-06
800x600	480000	0,9329	1,94354E-06
1024x768	786432	1,3946	1,77333E-06
1280x1024	1310720	2,0967	1,59966E-06
1600x1200	1920000	2,9094	1,51531E-06
1920x1200	2304000	3,3868	1,46997E-06

Tabulka 3.1: Tabulka s výsledky měření simulace zvukového vjemu

V tabulce je dobře vidět, že s narůstajícím počtem bodů roste i čas jednoho kroku simulace. Pro rozlišení používaná na dnešních moderních monitorech jsou to již více než 3 milisekundy. Čas potřebný na výpočet 1 simulovaného bodu se však při zvětšování rozlišení mírně snižuje (viz. obrázek 3.11). Z grafu na obrázku 3.12 také vyplývá, že čas jednoho snímku simulace roste úměrně s počtem bodů.



Obrázek 3.11: Graf měření s časem jednoho bodu (použité rozlišení na horizontální ose)



Obrázek 3.12: Graf měření s časem jednoho kroku (počet bodů 1 snímku na horizontální ose)

4 Závěr

Cílem této práce bylo vytvořit jednoduchou 2D hru s vizualizací hmatu a sluchu. V úvodu jsme položili základní představu o vizualizaci zvukových a hmatových vjemů a nastínili strukturu této práce. Práce byla rozdělena na dvě části. Nejprve teoretický úvod k vjemům, poté část věnující se implementaci těchto vjemů.

V první kapitole se nejprve věnujeme zvukovému vjemu. Uvedli jsme požadavky na hotové řešení. Dále jsme vyvodili numerické řešení z fyzikálních vztahů pro popis vln. Poté jsme uvedli metody, které se používají pro simulaci vlnění. K implementaci jsme vybrali poslední jmenovanou metodu, která je explicitní. Na začátku podkapitoly o hmatovém vjemu jsme se zabývali vlastnostmi povrchů (struktura, tvrdost a teplota) a jejich vlivem na vizualizaci vjemu. Poté jsme uvedli teoretický základ Bézierových a NURBS křivek.

V druhé kapitole jsme se věnovali návrhu implementace a také problémům vzniklým při implementaci a jejich řešení. Nejprve jsme popsali použité technologie, stručně uvedli vlastnosti OpenGL a jazyka GLSL a nastínili strukturu programu. Následovala část o implementaci herních úrovní (textury herní úrovně a systému souřadnic). Dále jsme se zabývali manipulací se shadery, implementací herní logiky a fyziky (generování geometrie, detekci kolizí) a implementací vjemů. V případě zvukového vjemu jsme se zabývali datovou reprezentací vln, implementací vlastní simulace, zmínili jsme dodatečné tlumení, popsali jsme a pomocí pseudokódu vysvětlili techniku ping-pong, objasnili obarvování vlnění a tvorbu zdroje zvuku. O hmatovém vjemu jsme mluvili v souvislosti s chováním, použitými shadery a definicí vlastností jednotlivých povrchů. Změřili jsme rychlost simulace a vyvodili přímou závislost doby trvání simulace na počtu bodů.

V průběhu práce jsme zmínili některá budoucí rozšíření. Patří k nim rozšíření simulace na nekonečnou plochu a pohyb kamery s hráčem. Důvodem jsou rozměrnější a komplexnější herní úrovně. Další možné budoucí rozšíření je obohacení vizualizace hmatu o tlak, který vzniká v místě dotyku. Pro vylepšení detekce kolizí jsme uvedli dvě možná rozšíření. Prvním je snížení množství objektů geometrie optimalizací algoritmu pro slučování kostek po načtení textury herní úrovně. Druhým je implementace detekci kolizí pro běh nezávisle na smyčce vykreslování. Obě rozšíření mají detekci kolizí urychlit, protože současná implementace má problémy s rychlými objekty. Dalším vhodným rozšířením může být implementace pokročilejší varianty SAT, která počítá i s dalšími tvary objektů. Vzhledem k použitému core OpenGL profilu se také nabízí možnost portování hry do OpenGL ES a distribuce programu na mobilní zařízení.

5 Literatura

[1] PLETCHER, Richard H, John C TANNEHILL a Dale A ANDERSON. *Computational fluid mechanics and heat transfer*. 3rd ed. Boca Raton: CRC Press, c2013, xx, 753 s. Series in computational and physical processes in mechanics and thermal sciencies. ISBN

[2] SUNDQVIST, Peter. *Interactive Real-time Simulation of Fluid Surfaces*. Umeå, 2007. Master's Thesis in Computing Science. Umeå University.

[3] FALSTAD, Paul. Ripple Tank Simulation. *Paul Falstad's Home Page* [online]. 2009 [cit. 2014-01-22]. Dostupné z: http://www.falstad.com/ripple/

[4] KASS, Michael a Gavin MILLER. Rapid, stable fluid dynamics for computer graphics. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH '90*. New York, New York, USA: ACM Press, 1990, roč. 24, č. 4, s. 49-57. DOI: 10.1145/97879.97884. Dostupné z: http://portal.acm.org/citation.cfm?doid=97879.97884

[5] VREUGDENHIL, C. *Numerical methods for shallow-water flow*. 1st ed. Dordrecht: Kluwer Academic Publishers, 1998, 261 s. ISBN 07-923-3164-8.

[6] LAYTON, Anita T. a Michiel VAN DE PANNE. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*. 2002, vol. 18, issue 1, s. 41-53. DOI: 10.1007/s003710100131. Dostupné z: http://link.springer.com/10.1007/s003710100131

[7] HALLIDAY, David, Robert RESNICK a Jearl WALKER. Fyzika: vysokoškolská učebnice obecné fyziky. Vyd. 1. Překlad Jana Musilová, Jan Obdržálek, Petr Dub. Brno: VUTIUM, 2003, xvi, 328, [28] s. ISBN 80-214-1868-0.

[8] AMES, William F. *Numerical methods for partial differential equations*. 2d ed. New York: Academic Press, 1977, xiv, 365 p. ISBN 01-205-6760-1.

[9] FAJMON, B., RŮŽIČKOVÁ, I. Matematika 3. UMAT FEKT, 2004.

[10] WEISSTEIN, Eric. Making MathWorld. The Mathematica Journal [online]. 2007 [cit. 2014-01-21]. Dostupné z: http://mathworld.wolfram.com/GaussianFunction.html

[11] WEISSTEIN, Eric. Making MathWorld. The Mathematica Journal [online]. 2007 [cit. 2014-01-21]. Dostupné z: http://mathworld.wolfram.com/BezierCurve.html

[12] MALINA, J. Vytvoření interaktivních studijních pomůcek pro výuku počítačové grafiky. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010

[13] JIŘÍ, Žára a Jiří ŽÁRA. Moderní počítačová grafika. Vyd 1. Brno: Computer Press, 2004, 609 s. ISBN 80-251-0454-0.

[14] SHREINER, Dave. OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1. 7th ed. Upper Saddle River, NJ: Addison-Wesley, c2010, l, 885 p. ISBN

[15] EBERLY, David. Intersection of convex objects: The method of separating axes. *Magic Software Inc. http://www. magic-software. com*, 2001.

[16] The OpenGL Framebuffer Object Extension. In: GREEN, Simon. *The OpenGL Framebuffer Object Extension* [online]. 2005 [cit. 2014-05-04]. Dostupné z: http://download.nvidia.com/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_FrameBuffer_Object.pdf