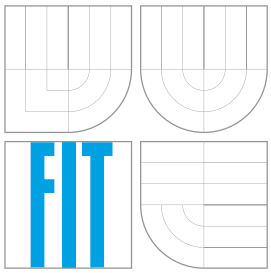
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D LOGICKÁ HRA S PRINCIPY DEFORMACE PROSTORU

3D PUZZLE GAME BASED ON SPACE DEFORMATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JURAJ JOŠČÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL TÓTH
SUPERVISOR

BRNO 2016

Abstrakt

Táto práca sa zaoberá globálnou deformáciou 3D geometrie a jej využitím v logickej hre. Cieľom je preskúmať rôzne metódy pre deformáciu geometrie v reálnom čase, zvážiť ich použiteľnosť v kontexte počítačovej hry a vyriešiť špecifické problémy, ktoré vyvstávajú pri vývoji takejto hry. Získané znalosti sú následne použité k implementácii hry v jazyku C++ za použitia grafickej knižnice OpenGL.

Abstract

This thesis focuses on global deformation of 3D geometry and using it in a videogame. The aim is to explore different methods for deformation of geometry in real time, consider their usability in context of a videogame, and solve specific problems, which can arise while developing such game. This knowledge is then used to implemennt the game in C++ programming language using OpenGL graphics library.

Kľúčové slová

3D, OpenGL, hra, deformácia, 3D textúra, C++, detekcia kolízií

Keywords

3D, OpenGL, game, deformation, 3D texture, C++, collision detection

Citácia

JOŠČÁK, Juraj. *3D logická hra s principy deformace prostoru.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tóth Michal.

3D logická hra s principy deformace prostoru

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Michala Tótha. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Juraj Joščák
14. mája 2016

Poďakovanie

Na tomto mieste chcem poďakovať vedúcemu mojej bakalárskej práce, pánovi Ing. Michalovi Tóthovi za jeho čas a cenné rady.

Obsah

Kapitola 1

Úvod

Počítačové hry dnes nájdeme takmer všade. Sú jedným z najrýchlejšie rastúcich odvetví zábavného priemyslu. Za posledných päťdesiat rokov ich vzniklo obrovské množstvo, od jednoduchých jednofarebných kratochvíľ, cez pohlcujúce príbehy schopné konkurovať filmom až po obrovské virtuálne svety s takmer neobmedzenými možnosťami. V dnešnej dobe je možné hrať hry na konzolách uspôsobených práve na tento účel, ale aj na osobných počítačoch, či dokonca na mobilných telefónoch, televízoroch a iných elektronických zariadeniach.

Z tejto záplavy hier, podstatná časť využíva na reprezentáciu objektov herného sveta trojrozmernú počítačovú grafiku, spravidla dynamickú a počítanú v reálnom čase. Animácia je nedeliteľnou súčasťou, a k nej patrí aj deformácia objektov. Ak napríklad chceme aby hlavný hrdina hry pôsobil realisticky, nestačí aby sa len presúval z miesta na miesto. Musí hýbať rukami, nohami, meniť výraz tváre. Od auta očakávame, že pri náraze sa mu pokriví karoséria. Vo väčšine hier sa teda deformácia trojrozmernej geometrie používa hlavne na to, aby bola hra vizuálne príťažlivejšia, zatiaľ čo na iné aspekty hrateľnosti nemá veľký dopad. Hier, kde by deformácia objektov hrala významnejšiu rolu je silný nedostatok, čo sa napokon stalo podnetom pre vypracovanie tejto práce.

Cieľom tejto práce je vytvoriť počítačovú hru, v ktorej bude práve deformácia geometrie hlavnou hernou mechanikou. Deformácia bude úplne pod kontrolou hráča a bude jej podliehať všetka geometria v každej úrovni tak, aby hráč získal pocit, že deformuje samotný priestor okolo neho.

Práca je členená na niekoľko častí. Druhá kapitola sa zaoberá teóriou trojrozmernej grafiky a rôznymi možnými riešeniami procedurálnej deformácie geometrie. Zvažované sú výhody a nevýhody každého riešenia, zložitosť ich implementácie ako aj nároky algoritmov na zdroje počítača. Ďalej je v kapitole diskutovaná detekcia kolízií, ktorá je v hrách tohto typu nevyhnutná, a treba ju prispôsobiť tak, aby správne fungovala aj s deformovanými objektmi.

Tretia kapitola obsahuje návrh vlastného riešenia, použité herné mechaniky, ovládanie a použité technológie. Popisuje tiež niektoré problémy s ktorými som sa pri vývoji stretol a ich riešenia.

Štvrtá kapitola približuje implementáciu programu. Piata kapitola diskutuje možnosti rozšírenia tohto projektu v budúcnosti. Záver práce obsahuje zhrnutie získaných poznatkov, zhodnotenie výsledkov celej práce a v nej použitých implemetačných metód.

Súčasťou práce je plne funkčná hra pre systém Windows, využívajúca spomínané mechaniky. Je implementovaná v jazyku C++ a využíva grafickú knižnicu OpenGL.

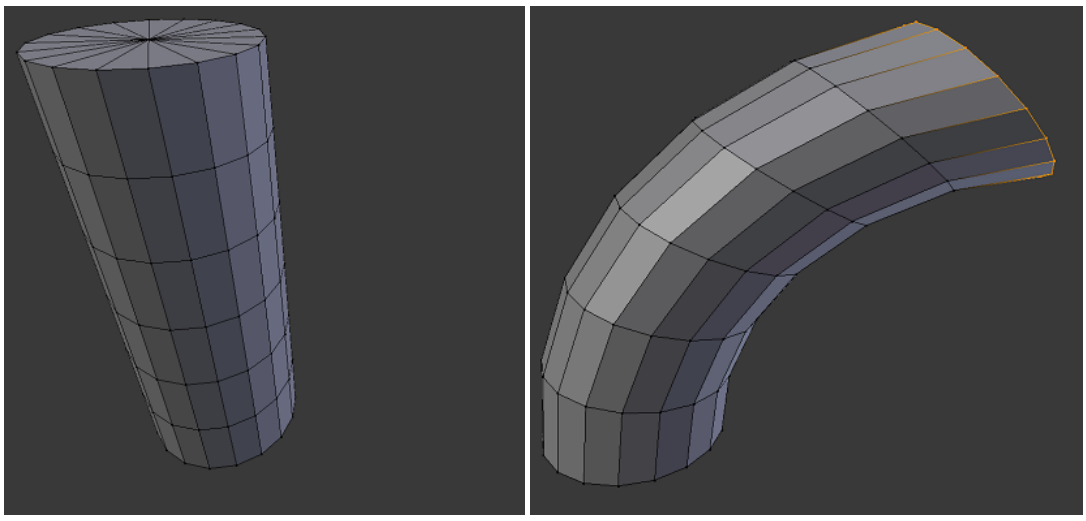
Kapitola 2

Možné riešenia

Táto kapitola uvádza možné riešenia dvoch hlavných problémov, s ktorými bolo potrebné sa vysporiadať pre úspešné dokončenie práce. Týmito problémami sú samotná deformácia a detekcia kolízií hráča s deformovanou geometriou hernej úrovne.

2.1 Deformácia

Deformáciu môžeme definovať ako zmenu tvaru daného objektu. Tvar môže byť v počítačovej grafike reprezentovaný rôznymi spôsobmi, napríklad mračnami bodov či Bézierovými plochami, ale zďaleka najpoužívanejšia v počítačových hrách je polygonálna reprezentácia, ktorú využíva aj zvyšok tejto práce (Obrázok 2.1). V tomto prípade je tvar objektu definovaný jeho povrchom, ktorý tvoria jednotlivé mnohouholníky – polygóny. Každý polygón tvorí skupina troch alebo viacerých vrcholov, z ktorých každý má presne určenú svoju polohu v priestore. Zmena polohy jednotlivých vrcholov v priestore tvorí základ akejkoľvek dynamiky.

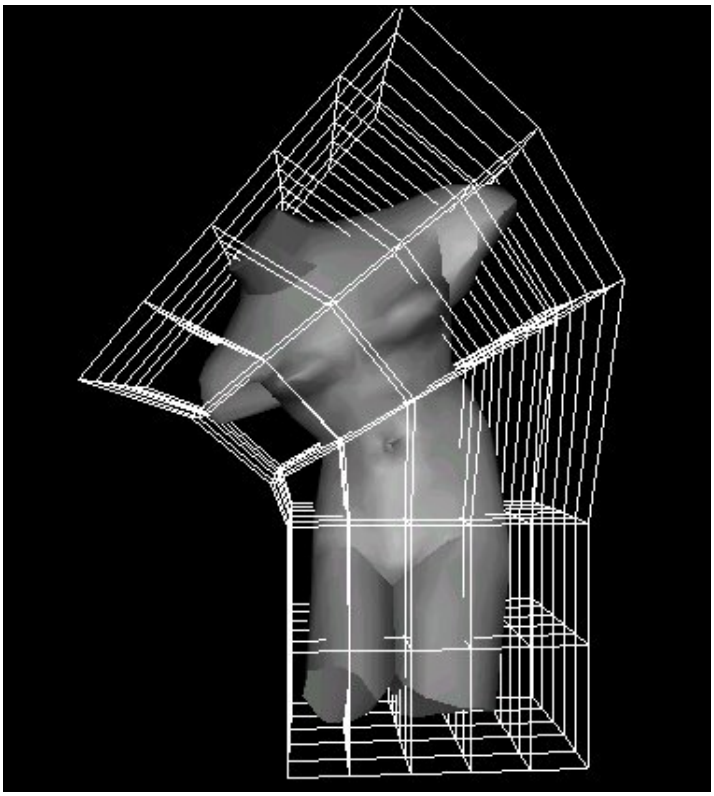


Obr. 2.1: Valec v polygonálnej reprezentácii a jeho zdeformovaná podoba

Deformácia je špecifická tým, že na rozdiel od jednoduchých transformácií, ako je posun alebo rotácia, pri nej dochádza nie len k zmene polohy vrcholov vzhľadom na počiatok súradnicového systému, ale aj vzájomne medzi jednotlivými vrcholmi. V podstate každú

transformáciu, ktorá nie je kombináciou posunu, rotácie a zmeny veľkosti môžeme označiť ako deformáciu. Pre každý objekt môže takýchto transformácií existovať prakticky nekonečno. Preto je hráčovi treba ponúknuť vhodný spôsob ovládania, tak aby mohol pohodlne využiť široké možnosti deformácie vo spoj prospech a zároveň sa nestratil v nekonečne možností.

Vhodnou abstrakciou od deformácie ako pohybu každého jedného bodu je metóda Free-form deformácie (ďalej len FFD)[8]. Táto metóda poníma deformáciu ako virtuálne sochárstvo. Deformovaný objekt je umiestnený do pravouhlej mriežky. Keď sú lokálne súradnice všetkých vrcholov objektu vyjadrené v priestore mriežky, presúvanie bodu mriežky sa prenáša na objekt prepočítaním súradníc objektu do globálneho priestoru (Obrázok 2.2). FFD možno prirovnať ku kvádru priehľadnej, pružnej hmoty v ktorej je zapustený objekt alebo objekty, ktoré chceme deformovať. Objekt sa taktiež pokladá za pružný, takže sa deformuje spolu s hmotou, ktorá ho obklopuje.



Obr. 2.2: Vizualizácia metódy Free-form deformation

Následne už stačí hráčovi len dať možnosť pomocou vhodného rozhrania touto mriežkou manipulovať, lokálne ju stláčať, roztahovať či ňou rotovať.

Nasledujúce podkapitoly rozoberajú niekoľko možností realizácie FFD v rámci celej úrovne v počítačovej hre.

2.1.1 FFD pomocou skeletálnej deformácie

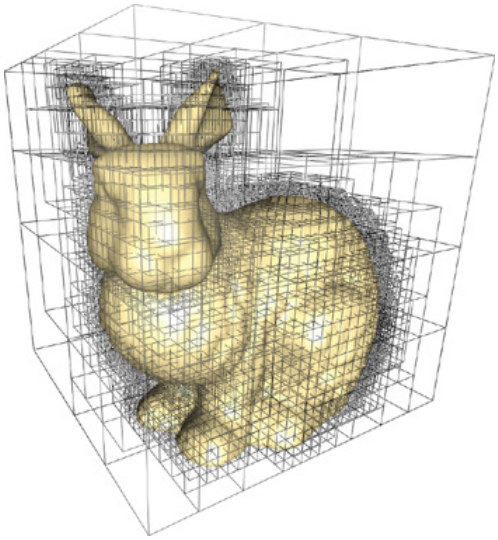
Skeletálna deformácia je veľmi populárna v odbore počítačovej animácie a je široko využívaná vo väčšine komerčných modelovacích a animačných programoch a počítačových hrách.

Jej popularita pramení z intuitívnej manipulácie, schopnosti rýchlo riešiť inverznú kinematiku na malom podpriestore (kostre zloženej z jednotlivých kostí) a efektívneho mapovania deformačných mechanizmov na grafický hardvér. Najzložitejšou časťou je interpolácia deformácií objektu v dostatočnej kvalite len z polohy kostry. Najrozšírenejšia je takzvaná metóda Skeletal Subspace Deformation (SSD)[9], ktorá transformuje každý vrchol geometrie váženým priemerom transformácií priradených ku kostiam ovplyvňujúcim tento vrchol.

FFD mriežku môžeme realizovať ako kostru zloženú z v priestore rovnomerne rozložených kostí. Váha podľa ktorej vrchol kopíruje transformácie kosti môže mať hodnotu od 0 – kosť nemá žiadny vplyv – po 1 – transformácia vrcholu je zhodná z transformáciou kosti. Každý vrchol objektu je ovplyvňovaný k nemu najbližšími kosťami, pričom celkový súčet váh je 1. Váha w je určená pomerom vzdialenosti kosti a vrcholu s a vzdialenosti medzi kosťami D v každej osi podľa vzorca 2.1.

$$w = \left(1 - \frac{s_1}{D}\right) \cdot \left(1 - \frac{s_2}{D}\right) \cdot \left(1 - \frac{s_3}{D}\right) \tag{2.1}$$

Pri tejto metóde je portrebné dobre zvážiť časové a pamäťové nároky. Už len mriežka s hustotou kostí 10 v každej osi by obsahovala celkom 1000 kostí. Pritom pre akákoľvek hernú úroveň rozumnej veľkosti by bolo 10 kostí naprieč príliš málo a nebolo by možné dosiahnuť dostatočnú presnosť deformácie. Zvyšovaním hustoty mriežky by sa počet kostí veľmi rýchlo dostal za rozumné hranice. Toto by sa dalo zmierniť použitím mriežky s variabilnou hustotou. Takáto mriežka by mohla mať napríklad podobu štruktúry Octree [5] (obrázok 2.3), kde v blízkosti vrcholov deformovanej geometrie by bola hustota kostí vyššia, než v miestach bez týchto vrcholov. Počiatočný výpočet váh by sa týmto skomplikoval, keďže D vo vzorci 2.1 by už nebolo konštantou.



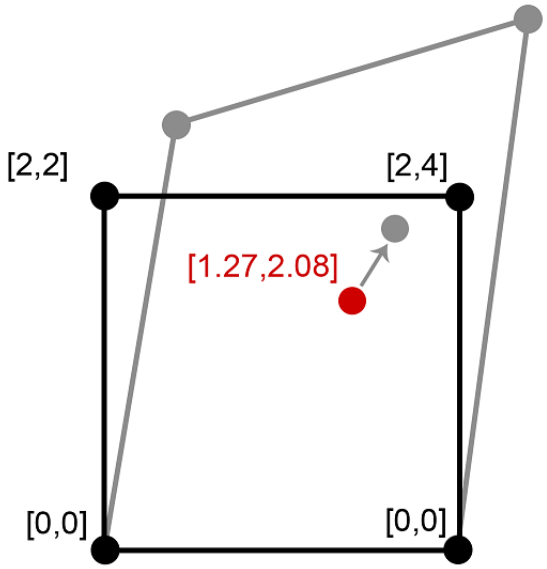
Obr. 2.3: Octree mriežka so zvýšenou hustotou v blízkosti geometrie

Na počet vrcholov geometrie je taktiež treba brať ohľad. Pre každý vrchol je potrebné si pamätať váhy jednej až ôsmich kostí, ktoré ho ovplyvňujú. Pri použití kvalitných modelov s veľkým množstvom vrcholov v kombinácii s veľkým počtom kostí by sa hlavne časové nároky počiatočného výpočtu váh mohli vymknúť spod kontroly.

2.1.2 FFD pomocou 3D textúry

Pri tejto metóde je FFD mriežka implementovaná ako trojkanálová 3D textúra. Každý pixel textúry predstavuje jeden kontrolný bod mriežky a jeho hodnota predstavuje zmenu oproti počiatočnej polohe v každej z troch osí. Absolútna poloha bodu mriežky sa teda vypočíta ako súčet súradníc daného pixelu a jeho hodnoty. Pohnúť bodom mriežky v niektorej z osí znamená zvýšiť/znížiť hodnotu príslušného kanálu daného pixelu. Poloha vrcholu geometrie sa následne vypočíta tak, že jeho počiatočná poloha sa namapuje do priestoru textúry a prečítaná hodnota v tomto bode sa aplikuje ako zmena polohy vrcholu oproti polohe počiatočnej.

Základom fungovania tejto metódy je spôsob, akým sa interpolujú hodnoty v miestach medzi texelmi. V tomto prípade sa ako najvhodnejšia javí trilineárna interpolácia (obrázok 2.4). Trilineárna interpolácia[3] sa používa na vypočítanie hodnoty na akejkoľvek pozícii pomocou váženého priemeru ôsmich pixelov najbližších k vstupným súradniciam. Nič viac na implementovanie dostatočne kvalitnej FFD nie je potrebné.



Obr. 2.4: Interpolácia hodnoty posunu vrcholu

Veľkou výhodou tejto metódy je vysoká podpora zo strany hardvéru. Veľmi dobre zoptimalizované funkcie pre prácu s 3D textúrou sú k dispozícií v ovládačoch väčšiny moderných grafických adaptérov, deformáciu je dokonca možné realizovať priamo pri vykresľovaní, čím sa prudko zníži záťaž na procesor. Pararelné výpočty na väčšom počte jadier grafického procesoru poskytne ďalšie zrýchlenie.

Nevýhodou je zvýšená závislosť na grafickom hardvéri. Posunutie procesu deformácie do fázy vykresľovania tiež znamená, že tvar objektu v pamäti sa vlastne nemení, deformácia sa prejaví až na obrazovke a s tým treba počítať pri návrhu ostatných častí programu. Niektoré funkcie, ktoré majú pracovať s už zdeformovanou geometriou, ju jednoducho nemajú k dispozícii. Táto práca používa práve túto metódu.

2.2 Detekcia kolízií

V počítačových hrách sa detekcia kolízií[2] stará o zachovanie ilúzie hmotného sveta. Zabraňuje napríklad hráčovi v prechádzaní cez steny či padaní cez podlahu. Umožňuje vystrelenej guľke zasiahnuť nepriateľa. Detekciu kolízií môžeme rozdeliť na dve podskupiny – detekciu a reakciu. Detekcia rieši kedy, kde a ako sa dva objekty stretnú. Reakcia určuje, ako kolízia ovplyvní chovanie týchto objektov.

Niektoré úkony, napríklad plánovanie trasy nevyžadujú detekciu kolízií v reálnom čase. Iné majú naopak mimoriadne nároky na jej efektívnosť. Počítačové hry zahŕňajú simulácie vyžadujúce veľký počet dotazov pri frekvenciách od 30 až do 60 snímkov za sekundu. Kvôli takejto časovej tiesni a pretože je to integrálna súčasť hry, detekcia kolízií dokáže zabrať veľké percento času, za aký sa dokončí snímok. Zle navrhnutý systém kolízií v počítačovej hre môže významne znížiť celkovú efektivitu.

Požiadavky na kolízny systém v tejto hre nie sú veľmi zložité. Kolízie je potrebé kontolovať len medzi dvoma objektmi – hráčom a geometriou hernej úrovne. Keďže pri použití pohľadu z prvej osoby hráč v podstate nemá žiadny tvar, môžeme ho pre účely detekcie kolízií nahradiť akýmkoľvek zjednodušeným objektom, napríklad kockou alebo elipsoidom. Takýto zástupný objekt nemusí byť tvorený polygónmi, môže byť definovaný ako jednoduchý objekt vyjadrený pomocou matematickej rovnice. Kocku môžeme napríklad vyjadriť ako množinu všetkých bodov x ktoré vyhovujú rovnici 2.2, kde c vyjadruje stred kocky a a dĺžku jej strany.

$$max\{|x_1 - c_1|, |x_2 - c_2|, |x_3 - c_3|\} \quad = \quad \frac{a}{2} \tag{2.2}$$

Keďže deformáciu geometrie, berieme ako "deformáciu priestoru", jediné čo sa v priestore pohybuje je hráč. Reakcia na kolíziu sa teda obmedzuje na jednoduché zastavenie pohybu hráča v smere, v ktorom došlo ku kolízii.

Zistiť kolíziu dvoch objektov, z ktorých jeden je tvorený polygónmi a druhý je definovaný iným spôsobom znamená zistiť kolíziu jedného alebo viacerých polygónov prvého objektu s druhým objektom. To znamená zistiť najmenšiu vzdialenosť objektu od plochy určenej vrcholmi polygónu, a ak je táto vzdialenosť nulová, rozhodnúť či sa dány bod nachádza vnútri polygónu.

Nemôžeme dopredu vedieť, kde sa objekty stretnú, eventuálne teda budeme musieť skontolovať každý jeden polygón. V dnešnej dobe môže jedna herná úroveň obsahovať stotisíce až milióny polygónov, čo predstavuje nezanedbateľnú záťaž na procesor, a môže negatívne ovplyvniť výkon. V nasledujúcich podkapitolách uvádzam niekoľko možností ako túto záťaž znížiť.

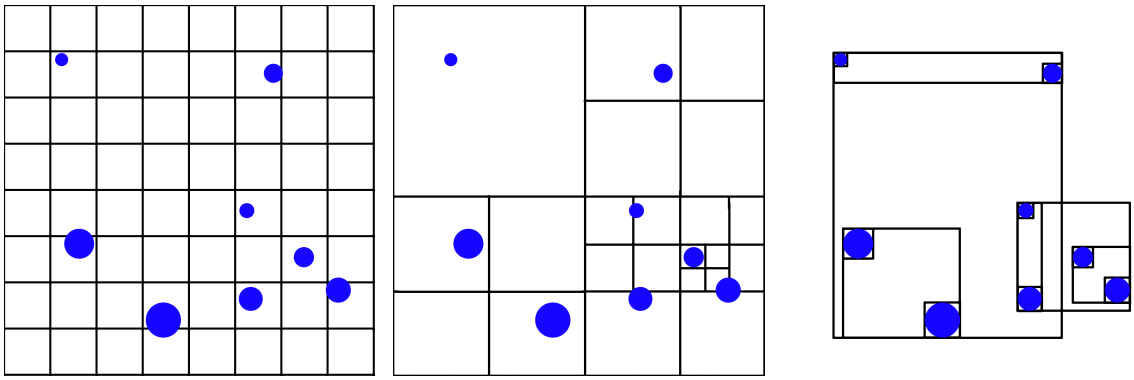
2.2.1 Delenie priestoru

Táto metóda spočíva v rozdelení priestoru na niekoľko menších oblastí a určení, do ktorej oblasti daný objekt patrí. Ak sa dva objekty nenachádzajú v určitom čase v tej istej oblasti, môžeme o nich povedať, že sú od seba príliš ďaleko a kolízia medzi nimi v blízkom čase nemôže nastať. Tým sa niekoľkonásobne obmedzí počet testovaní kolízií.

Priestor môžeme rozdeliť niekoľkými spôsobmi (Obrázok 2.5). Najjednoduchším z nich je pravidelná pravouhlá mriežka. Takáto mriežka je jednoduchá na implementáciu, ale trpí niekoľkými problémami. Nie je možné zabrániť tomu, aby sa niektorý objekt nachádzal na hranici viacerých oblastí. V takom prípade sa s ním musí počítať v kazdej z nich. Tento

prípad nastáva častejšie, ak sú objekty príliš veľké alebo oblasti príliš malé. Naopak ak sú oblasti príliš veľké, nachádza sa v nich priemerne viac objektov, počet testovaní kolízií sa zvyšuje a celá táto metóda stráca zmysel. Je teda nutné zvoliť vhodnú veľkosť oblastí.

Ak priestor rozdelíme pomocou stromovej štruktúry, nemusíme vhodnú veľkosť oblastí riešiť, pretože je v podstate variabilná. Takouto štruktúrou môže byť napríklad Octree[5] (viď strana 5) alebo BVH (bounding volume hierarchy)[4]. Tieto štruktúry možno vypočítať dopredu a znížiť tak zátaž pri práci so statickými objektmi.



Obr. 2.5: Rôzne spôsoby delenia priestoru: (zľava) jednoduchá mriežka, octree, BVH strom

Hlavnou výhodou delenia priestoru je zníženie počtu porovnávaní medzi veľkým počtom objektov. V našom prípade síce testujeme len kolízie hráča proti geometrii úrovne, netestujeme jednotlivé polygóny proti sebe, no aj tak pri desaťtisícoch polygónov táto metóda poskytuje markantné zrýchlenie. Vo finálnej verzii nebola implementovaná.

2.2.2 Zástupný objekt

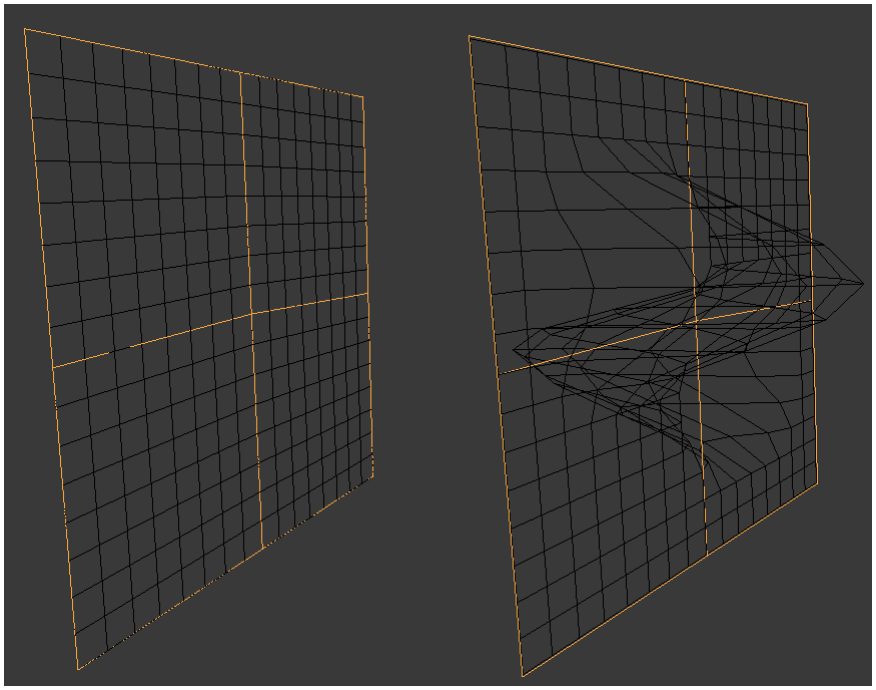
Ďalšou spôsobom ako zjednodušiť detekciu kolízií je použitie zástupných objektov. Vychádzame z toho, že človek si veľmi rýchlo všimne príliš jednoduchý model na obrazovke, ale fyzikálne interakcie nedokáže odhadnúť veľmi presne, hlavne v rýchlosti. Každý objekt teda môžeme pre účely fyzikálnych simulácií nahradiť zjednodušeným zástupným objektom. Na strane 7 už bolo spomenuté nahradenie objektu hráča matematickou rovnicou. Polygonálny model môžeme nahradiť iným, podobným polygonálnym modelom, ale s menším počtom polygónov. Tento zástupný model za použije pri detekcii kolízií, zatiaľ čo pôvodný model bude zobrazovaný na obrazovke.

Je však veľmi dôležité voliť zástupný objekt tak, aby jeho fyzikálne vlastnosti približne odpovedali tomu, čo človek očakáva od pôvodného, zobrazeného objektu. Nemôžeme napríklad nahradiť hranatý objekt guľou, pretože od hranatého objektu zvyčajne neočakávame, že sa bude kotúľať.

Zástupné modely nedeformovateľných objektov sú väčšinou dopredu vytvorené ručne, prípadne jednorázovo automaticky vygenerované a v priebehu času sa nemenia. Pri objektoch schopných deformácie je situácia komplikovanejšia. Ak objekt mení tvar, očakávame aj zmenu jeho fyzikálnych vlastností. Zástupný objekt sa teda musí aktualizovať po každej deformácii a ručne vytvorené modely neprichádzajú do úvahy.

Jednou možnosťou je aplikovať na zástupný model rovnaký algoritmus deformácie ako na pôvodný model. Toto však nemusí vždy fungovať. Zástupný model niekedy ani nemusí byť možné deformovať tak, aby odpovedal novému tvaru pôvodného objektu. Na obrázku

2.6 oranžový zástupný model dokonale odpovedá čiernemu modelu pred deformáciou, avšak po deformácii je už prakticky takmer nepoužiteľný.



Obr. 2.6: Zástupný model je po deformácii nepoužiteľný

Generovať zástupný model automaticky pri každej deformácii by vyžadovalo algoritmus, ktorý by bol schopný produkovať dostatočne kvalitné modely, a to dostatočne rýchlo. Vytvorenie takého algoritmu by presahovalo rozsah tejto práce.

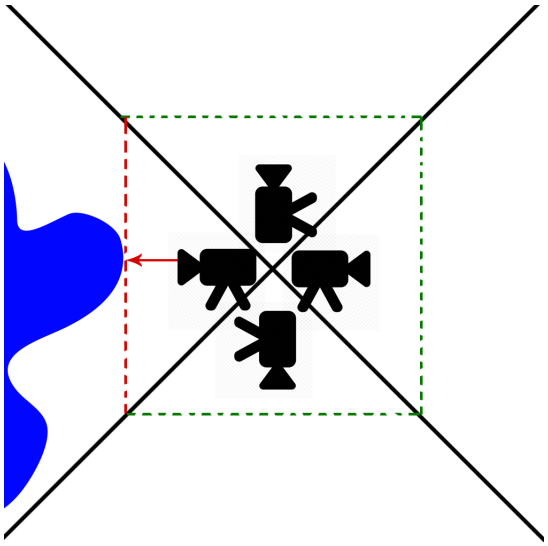
2.2.3 Využitie Z-bufferu

Grafické karty sú veľmi dobre optimalizované na prácu s veľkým množstvom polygónov, bolo by teda veľmi výhodné tento výkon využiť aj na detekciu kolízií.

Jednou z hlavných činností vykonávaných pri detekcii kolízii je zisťovanie najmenšej vzdialenosti. Po rasterizácii máme k dispozícii hĺbku každého fragmentu každého polygónu v Z-bufferi. Táto hĺbka je v podstate vzdialenosť fragmentu od pomyselnej kamery, a nájdenie najnižšej hodnoty je už triviálnosťou. Rozdiel tejto hodnoty a nami nastavenej minimálnej vzdialenosti vlastne určuje, o koľko sa ešte môže hráč pohnúť v smere, v ktorom je kamera namierená, predtým než narazí na prekážku. Ak túto metódu použijeme vo všetkých šiestich smeroch (smer nahor, nadol, vpravo, vľavo, vpred a vzad), môžeme zaručiť, že zaregistrujeme kolíziu hráča s prekážkou, nech už nastane v ktoromkoľvek smere (obrázok 2.7). V každom smere pritom môžeme nastaviť inú minimálnu vzdialenosť.

Pri použití tejto metódy treba pamätať na to, že každá kamera musí vykresliť celú scénu predtým, než získame akékoľvek relevantné informácie. Našťastie pre potreby detekcie kolízií netreba vykresľovať v takej kvalite ako to, čo sa má zobraziť na obrazovke. V podstate potrebujeme len informácie o hĺbke, nepotrebujeme textúry, osvetlenie ani podobné estetické prvky, rozlíšenie môžeme niekoľkonásobne znížiť. Nie však príliš, pretože pri príliš nízkom rozlíšení sa nemusia niektoré malé polygóny vôbec vykresliť, a teda kolízia by nebola detekovaná.

Výhodou tejto metódy je to, že väčšina výpočtu prebieha na grafickej karte a procesor sa zatiaľ môže zaoberať inými vecami. Navyše funguje veľmi dobre aj s geometriou deformovanou až pri vykresľovaní, ako bolo spomenuté v podkapitole 3.3. Práve preto bola táto metóda použitá v tejto práci.



Obr. 2.7: Pomocou systému kamier vieme presne určiť vzdialenosť a smer k najbližšej prekážke

Kapitola 3

Vlastné riešenie

Táto kapitola osahuje postupy, ktoré som použil pri návrhu a realizácii vlastného riešenia. Podrobnejšie sa venuje niektorým metódam opísaným v predchádzajúcej kapitole a uvádza niektoré problémy, s ktorými som sa pri riešení stretol.

3.1 Koncept hry

Konceptualizácia je nevyhnutnou súčasťou procesu počítačovej hry. V tejto fáze sa rozhodne o čom hra bude, ako sa bude hrať a aký celkový dojem v hráčovi zanechá. Táto podkapitola sa venuje práve týmto témam.

3.1.1 Základné princípy a mechaniky

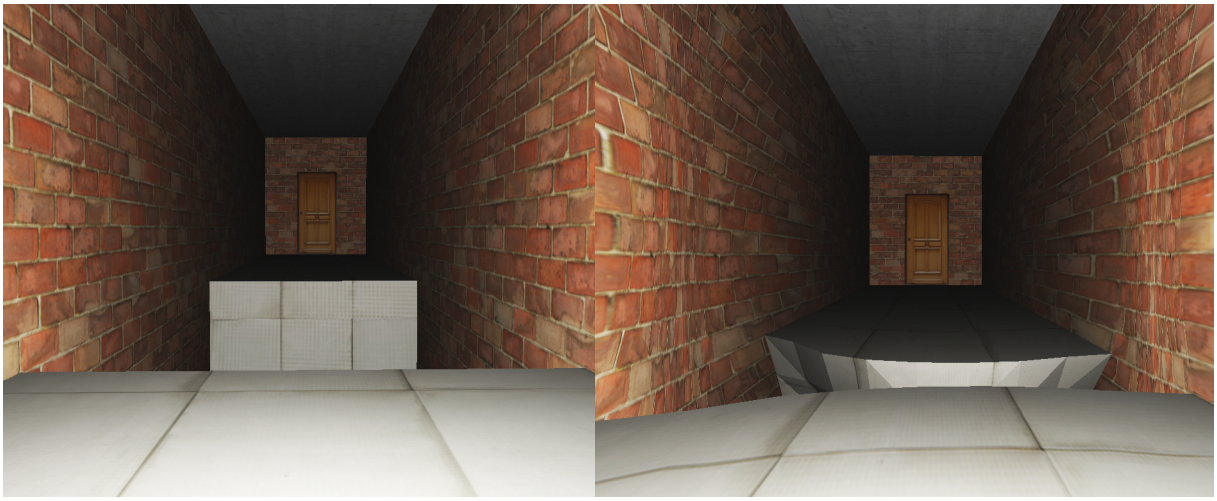
Základný koncept hry vychádza z hier, ako sú *Portal*, *Magrunner* či *Quantum Conundrum*. Hra sa odohráva v trojrozmerných úrovniach, ktorým nechýba vertikalita. Hráč nazerá na hru s pohľadu prvej osoby a jeho jediným cieľom je dostať sa v každej úrovni na dopredu určené miesto, čím úroveň končí a hra pokračuje v nasledujúcej. Pri tomto úsilí musí hráč využiť svoje orientačné schopnosti, reflexy a precíznosť pri skákaní medzi platformami.

Jadro hry však tvorí možnosť deformovať priestor. Hráč je schopný priestor "stlačiť", teda skrátiť vzdialenosť medzi jednotlivými časťami úrovne vo vertikálnom, pozdĺžnom alebo priečnom smere vzhľadom na jeho polohu. V podobnom duchu môže tiež priestor "natiahnuť", teda predĺžiť vzdialenosti v danom smere. Plne pod hráčovou kontrolou je epicentrum deformácie, jej intenzita ako aj rádius, kam až deformácia siaha. Deformáciu hráč môže využiť napríklad na skrátenie vzdialenosti kam treba doskočiť, na zväčšenie otvoru, ktorý by bol ináč príliš úzky, a na iné účely, ktoré pomôžu otvoriť pôvodne neprístupnú cestu na koniec úrovne (obrázok 3.1).

Ak to hráč s deformáciami preženie, má tiež možnosť hocikedy všetky deformácie v úrovni globálne negovať a postupne približovať celú úroveň jej pôvodnej podobe bez toho, aby musel začínať od začiatku.

V hre sa nevyskytujú žiadni nepriatelia, smrtiace pasce alebo čokoľvek iné, čo by mohlo hráča ohroziť. Tempo hry je teda dostatočne pomalé na to, aby si hráč mohol každý svoj krok dôkladne premyslieť. Jediná možnosť ako skončiť úroveň neúspešne je, ak sa hráč sám vzdá. V prípade že sa hráčovi podarí vypadnúť mimo priestor úrovne, úroveň sa automaticky reštartuje.

Hru môžeme žánrovo zaradiť ako 3D plošinovku s logickými prvkami.



Obr. 3.1: Deformácia otvoru v podlahe umožní hráčovi preskočiť na opačnú stranu

3.1.2 Užívateľské rozhranie a ovládanie

Ovládanie hry vyžíva kombináciu klávesnice a myši a do veľkej miery odpovedá štandardom zaužívaným v hrách z pohľadu prvej osoby. Pohyb hráča dopredu, dozadu a do strán ovládajú klávesy `[W]` `[S]` `[A]` a `[D]`. `[Medzerník]` ovláda skok. Pohyb hráča možno ovládať len pokiaľ sa dotýka zeme, vo vzduchu podlieha iba zotrvačnosti a gravitácii, prípadne nárazom o steny a strop. Takéto správanie viac odpovedá realite pridáva hre na náročnosti, pretože takto treba skoky dobredu dobre naplánovať.

Deformácia je ovládaná myšou. Pri stlačení a držaní pravého tlačidla myši pohyb myši ovláda deformáciu v pozdĺžnej a priečnej osi, vo zvislej osi deformáciu ovláda koliesko myši. Epicentrum deformácie určuje 3D kurzor, ktorý sa vždy nachádza priamo pred hráčom vo vzdialenosti, ktorú je možné nastaviť tlačidlami `[LShift]` a `[LControl]`. Vždy sa však musí nachádzať v dohľade hráča, nemožno ho umiestniť za stenu. Maximálna vzdialenosť kurzoru od hráča je teda vzdialenosť k najbližšej prekážke v smere priamo dopredu. Minimálna vzdialenosť je pochopiteľne nulová. Rádius deformácie sa nastaví kolieskom myši pri uvoľnenom pravom tlačítku.

Užívateské rozhranie je minimalistické, na obrazovke sa zobrazuje len výhľad do herného sveta. Vyššie spomenutý 3D kurzor má podobu vrstevnatej gule svetla, dopadajúceho všade tam kde sa má niečo deformovať. Okrem toho sa už hráčovi nezobrazujú žiadne ďalšie informácie, okrem okrem hlášky "CLEARED"po dokončení úrovne.

Na predčasné ukončenie úrovne a vstup do hlavného menu slúži klávesa `[ESCAPE]`. Hlavné menu umožnuje načítať inú úroveň alebo ukončiť program.

3.1.3 Audiovizuálne spracovanie

Deformácia priestoru môže na hráča pôsobiť vcelku psychedelicky. Audiovizuálnym spracovaním som sa rozhodol tento pocit ešte umocniť a vytvoriť svet na hranici reality, kde veci nedávajú zmysel tak ako sme zvyknutý.

Pri pohybe hráča nie je použitý viditeľný head bobbing, pohyb hráča je plynulý a navádza pocit nehmotnosti.

Povrch objektov je tvorený fotografickými textúrami, takže vyzerá vcelku realisticky, avšak tieto textúry sú použité neočakávaným spôsobom a na neočakávaných miestach.

Každá úroveň je osvetlená ambientným svetlom a jedným slabým bodovým svetlom, ktoré sa pohybuje spolu s hráčom. Toto osvetlenie spolu s nízkym kontrastom vytvára pocit šera, cez ktoré sa hráč snaží svojim zrakom preniknúť. 3D kurzor vydáva naopak ostré modrasté svetlo, takže je ho dobre vidieť a možno ho použiť ako akúsi baterku. Hra používa iba difúzny osvetľovací model bez akejkoľvek variability, čo spôsobuje jemný pocit senzorickej deprivácie a pridáva na celkovej tiesnivej atmosfére.

Hra neobsahuje žiadne zvuky, hráč teda môže počúvať hudbu podľa vlastného výberu, alebo sa vďaka pomalému tempu hry môže zapájať do koverzácie s ľuďmi v jeho okolí.

3.2 Výber technológií

Pri vývoji počítačových hier je výber vhodných technológií kritický. Je treba vybrať technológie, ktoré dostatočne ovládame, inak by vývoj mohol trvať príliš dlho a výsledný produkt by nedosahoval vysokú kvalitu, ale tiež treba dbať na to aby boli tieto technológie dostatočne rýchle pre hranie v reálnom čase. Tiež je dôležitá prenositeľnosť, minimálne medzi rôznymi hardvérovými konfiguráciami tej istej platformy, čo je v našom prípade PC s operačným systémom Windows.

Základom je zvoliť správny programovací jazyk. V tejto práci je použitý jazyk C++. Tento jazyk je na písanie počítačových hier najvhodnejší, čomu svedčí aj fakt, že 95%[1]. hier v dnešnej dobe bolo napísaných práve v C++. Ďalej je dôležitá voľba grafickej knižnice na zobrazovanie scény, a knižnice užívateľského rozhrania.

3.2.1 Grafická knižnica

Grafická knižnica umožňuje programu komunikovať s grafickou kartou. Poskytuje akúsi abstrakciu od hardvéru, takže programátor sa nemusí zaoberať problémami špecifickými pre každý jeden typ grafickej karty. Ak je program napísaný s použitím grafickej knižnice, bude fungovať na každej grafickej karte, ktorá túto knižnicu podporuje. Grafická knižnica obsahuje dátové štruktúry a funkcie potrebné na vykonávanie zložitých priestorových výpočtov, rasterizáciu geometrických útvarov, mapovanie textúr a iné. V niektorých prípadoch môže byť grafická knižnica použitá aj na výpočty, ktoré s grafikou nesúvisia. Vzhľadom na charakter tejto práce sa na výber ponúkajú dva grafické knižnice – OpenGL a DirectX.

DirectX bolo vyvinuté firmou Microsoft priamo pre potreby hier. Okrem grafiky zvláda aj zvuk, video, či spracovanie vstupu. To však znamená, že je potrebné napísať veľké množstvo kódu predtým než nám program ponúkne prvé výsledky. Taktiež má DirectX občas problémy s kompatibilitou a je závislý na platforme Windows.

Knižnica OpenGL (Open Graphics Library)[7] byla navrhnutá firmou Silicon Graphics ako aplikačné programové rozhranie (Application Programming Interface – API) k akcelerovaným grafickým kartám, respektíve celým grafickým subsystémom. OpenGL bola navrhnutá s dôrazom na to, aby bola použiteľná na rôznych typoch grafických akcelerátorov a aby ju bolo možné použiť aj v prípade, že na určitej platforme nie je žiadny akcelerátor nainštalovaný – v tom prípade sa použije softvérová simulácia.

Knižnica OpenGL (narozdiel od DirectX) bola vytvorená tak, aby bola nezávislá na použitom operačnom systéme, grafických ovládačoch a správcoch okien. Preto tiež neobsahuje žiadne funkcie na prácu s oknami, na vytváranie grafického užívateľského rozhrania ani ani na spracovanie vstupu. Na to je potrebné použiť knižnicu grafického užívateľského rozhrania (viď podkapitola 3.2.2).

Programátorske rozhranie knižnice OpenGL bolo vytvorené tak, aby bolo použiteľné takmer v ľubovoľnom programovacom jazyku. primárne je k dispozícii hlavičkový súbor pre C a C++. Pomocou funkcií poskytovaných OpenGL možno vykresľovať obrazce a telesá vytvorené zo základných grafických primitiv, ako sú bod, úsečka, trojuholník, polygón a iné. Vykresľované primitíva môžu byť osvetlené alebo pokryté textúrou.

3.2.2 Knižnica užívateľského rozhrania

Užívateľské rozhranie sa stará o komunikáciu medzi programom a užívateľom, v tomto prípade hráčom. Umožňuje hráčovi ovplyvňovať stav systému tým, že spracúva jeho vstupy a prezentuje mu výstupy. Grafické užívateľské rozhranie komunikuje pomocou okien, na vstup sa používajú vstupne zariadenia ako klávesnica, myš alebo špeciálne herné ovládače. Knižnice grafického užívateľského rozhrania obsahujú funkcie a dátove štruktúry, ktoré rozširujú zvolený programovací jazyk o možnosť vytvárať okná, ovládacie prvky, starajú sa o zbieranie informácií zo vstupov a od systému a zobrazovanie výstupov užívateľovi zrozumiteľnou formou.

Táto práca potrebuje knižnicu užívateľského rozhrania na vytvorenie okna, OpenGL kontextu, na spracovanie vstupu a časovanie. Na tento účel som zvolil knižnicu SDL (Simple Directmedia Layer).

Počiatky knižnice SDL[6] ukazujú k spoločnosoti Loki Entertainment Software, ktorá sa zaoberá portovaním hier do operačného systému GNU/Linux. Bola navrhnutá ako nízkoúrovňové API pre tvorbu hier a všeobecne multimediálnych aplikácií. Z veľkej časti zastrešuje funkcie operačného systému, a tým umžňuje takmer stopercentú prenositeľnosť zdrojového kódu. SDL obsahuje funkcie na vytvorenie okna (vrátane fullscreenu) a správu udalostí. Dvojrozmerná grafika je zahrnutá priamo, 3D grafika je realizovaná pomocou OpenGL, ktoré má priamu podporu. SDL ďalej umožnuje prácu s audiom, CD-ROM a časovačmi, podporuje tiež viacvláknove programovanie.

Táto knižnica je relatívne malá. V jadre obsahuje iba základnú funkcionalitu, vďaka čomu je prehľadná a nezahlcuje programátora žiadnym gigantickým API. Ostatné funkcie poskytujú rôzne nadstavby. V najhoršom prípade musí programátor všetko potrebné dotvoriť sám.

Samotné SDL býva štandardnou súčasťou drvivej väčšiny linuxových distribúcií, pri iných operačných systémoch je ho možné stiahnuť. V súčasnosti je SDL portované do operačných systémov GNU/Linux, Windows, BeOS, OS X, FreeBSD, Solaris, IRIX, QNX a iných.

SDL je napísané v jazyku C a a samozrejme funguje aj v C++. Môže sa však použiť aj v ďalších jazykoch, napríklad C#, Java, Lisp, Objective C, Pascal, PHP, Python alebo Ruby.

3.3 Deformácia

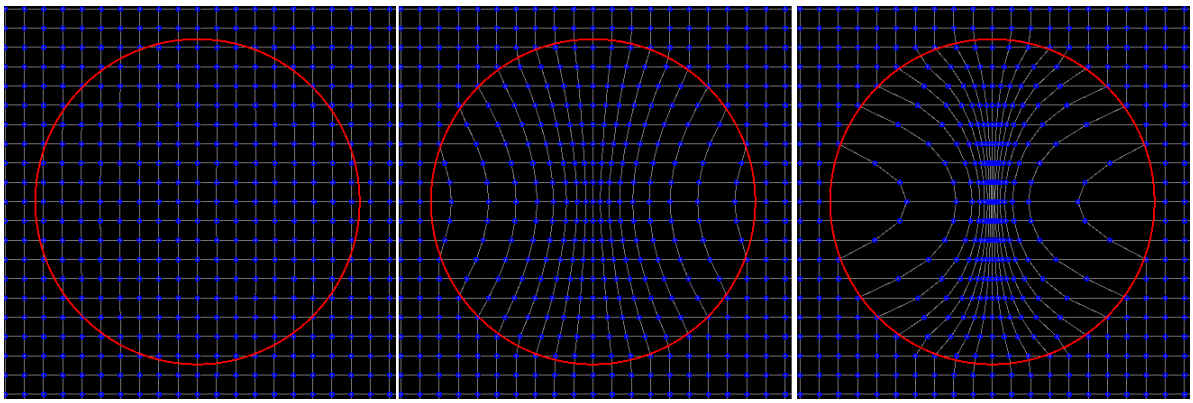
Pre deformáciu geometrie úrovne hra využíva metódu FFD pomocou 3D textúry, popísanú v podkapitole 2.1.2.

Hráč s FFD mriežkou interaguje pomocou 3D kurzora, ktorý má jasne určený stred a rádius, má teda tvar gule. Ovplyvňované sú tie kontrolné body FFD mriežky, ktoré sa nachádzajú vnútri tejto gule. Aby bol vyvolaný dojem stlačenia priestoru, musia sa všetky tieto body pohnúť k stredu kurzora po určenej osi.

Algoritmus, ktorý toto zabezpečuje, nepracuje s absolútnymi vzdialenosťami, ale so zlomkami vzdialenosti daného bodu od stredu. Vzdialenosť S, o akú sa má bod posunúť je určená vzorcom 3.1 kde P je vektor smerujúci od stredu kurzora k dotyčnému bodu, D je jednotkový vektor určujúci smer deformácie, R je rádius kurzoru a A je konštantný násobič ovládajúci intenzitu deformácie v jednom kroku.

$$S = P \cdot D \cdot \left(1 - \frac{|P|}{R}\right) \cdot A \tag{3.1}$$

Tento vzorec zabezpečuje, že čím bližšie je bod k stredu, tým pomalšie sa pohybuje až sa v strede úplne zastaví (obrázok 3.2). Keďže sa takto menia len vzdialenosti medzi jednotlivými bodmi a nie ich vzájomná poloha v priestore je zaručená integrita mriežky, nedôjde k jej pretínaniu a v konečnom dôsledku ani k pretínaniu polygónov geometrie, ktorej deformáciu mriežka ovláda. To je samozrejme podmienené presnosťou výpočtu. Pri použití nedostatočne presných čísel v pohyblivej rádovej čiarke sa môžu objaviť drobné artefakty, nemali by však byť veľmi nápadné.



Obr. 3.2: Deformácia mriežky v rôznych štádiách

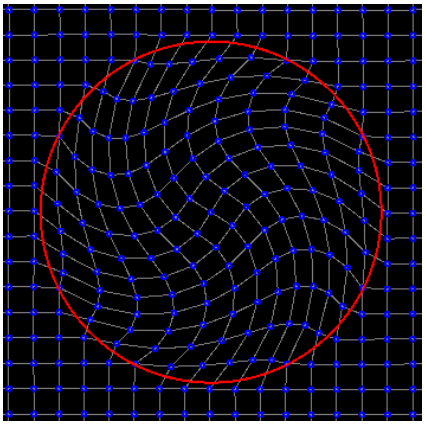
Okrem stlačenia mriežky je možné ju aj natiahnuť. Pri tom je takisto použitý vzorec 3.1, konštanta A je v tomto prípade záporná. Pri naťahovaní mriežky kontrolné body namiesto stredu konvergujú smerom k okraju kurzora.

Okrem stláčania a naťahovania je ďalším možným spôsobom deformácie zatočenie (obrázok 3.3). Každý kontrolný bod rotuje okolo stredu, pričom body blízko stredu rotujú rýchlejšie než vzdialenejšie body. Tým sa však mriežka stáva náchylnou na stratu integrity. Toto sa mi bohužiaľ z dôvodu nedostatku času nepodarilo ošetriť.

Keďže výpočet deformácie mriežky sa skladá z množstva nezávislých výpočtov pre každý jej bod, je tento algoritmus výborným kandidátom na pararelizáciu. Tým by sa významne znížil čas potrebný na výpočet, preto som sa rozhodol na počítanie deformácie mriežky využiť grafickú kartu.

3.4 Detekcia kolízií

Pre detekciu kolízií som použil Z-buffer, ako je popísané v podkapitole 2.2.3. Táto metóda nie je vhodná na prácu s veľkým množstvom objektov, ale pre detekciu kolízií hráča s komplikovanou geometriou úrovne je ideálna.



Obr. 3.3: Zatočenie mriežky

Na hráčovej pozícii je umiestnených šesť kamier, každá namierená do jedného zo šiestich smerov: +x, -x, +y, -y, +z a -z. Kamery používajú ortografickú projekciu, ich zorné polia majú tvar kvádra. Kamera zaregistruje kolíziu v smere do ktorého je namierená vždy, keď sa niečo vykreslí v jej zornom poli v hĺbke menšej než je určitá hodnota. Pre zjednodušenie je táto hodnota vždy rovná hĺbke celého zorného poľa kamery. Kamera teda zaregistruje kolíziu, vždy keď sa čokoľvek vykreslí v jej zornom poli.

Reakcia na kolíziu spočíva v jednoduchom posunutí hráča späť proti smeru kamery, ktorá zachytila kolíziu, o vzdialenosť takú, ktorá sa rovná rozdielu hĺbky zorného poľa a hĺbky, v ktorej bola hĺbka zachytená. Tým sa uveriteľne nasimuluje náraz do prekážky v tomto smere.

Správanie celého systému môžeme ovplyvniť nastavením rozmerov zorných polí všetkých kamier. Pri vhodnom pomere môžu napríklad dokopy tvoriť kváder, pričom zachytíme kolíziu s ktoroukoľvek jeho stenou. Obrázok 3.4 ukazuje túto možnosť v 2D.

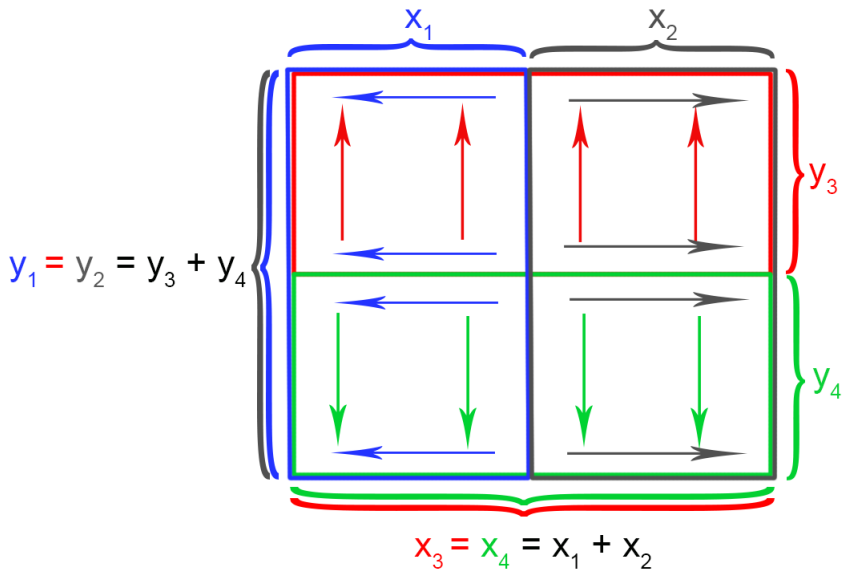
Pri testovaní však tento systém pôsobil značné problémy. V prípade že kolízia nastala na hrane alebo dokonca na rohu tohto kvádra, zareagovalo na ňu viac kamier súčasne a smer, kam sa hráč posunul bol teda dielom náhody. To spôsobovalo neočakávané chovanie a hra sa stala takmer neovládateľnou.

Riešením bolo zornému poľu každej kamery nechať pôvodnú hĺbku, ale šírku a výšku zmenšiť na polovicu. Takto vytvárajú dohromady tvar trojrozmerného znamienka plus a pravdepodobnosť, že kolízia nastane na rozhraní dvoch z nich je zanedbateľná.

Pri testovaní sa tento systém správal podobne ako kváder so skosenými hranami. Bol dokonca schopný kĺzania po šikmej ploche. V niektorých špecifických situáciách spôsobil osciláciu hráča medzi dvoma polohami, avšak toto nenastáva často, preto som tento systém zhodnotil ako dostačujúci pre túto prácu.

3.5 Návrh aplikácie

Finálna aplikácia bola realizovaná v podobe niekoľkých spolupracujúcich jednotiek. Nasledujúca podkapitola obsahuje ich stručný popis.



Obr. 3.4: Zorné polia v správnom pomere dokopy vytvoria kváder

3.5.1 Základné triedy

Názvy všetkých mnou vytvorených tried začínajú písmenami WP (prvé a posledné písmeno slova warp = deformácia/zakrivenie), aby sa dali ľahko odlíšiť od ostatných tried pochádzajúcich z externých knižníc. Názvy som sa snažil voliť tak, aby viac-menej popisovali zmysel každej triedy. Názvy sú v angličtine s dôvodu konzistencie so zvyškom programu.

Trieda `WPVector` reprezentuje trojrozmerný matematický vektor. Obsahuje metódy ma manipuláciu s vektorom, zistenie jeho dĺžky a podobne. Túto triedu využívajú takmer všetky ostatné triedy.

Trieda `WPMatrix` predstavuje matematickú maticu 4x4. Obsahuje metódy na sčítanie, násobenie a podobne.

Trieda `WPWarpable` reprezentuje geometriu schopnú deformácie. Obsahuje informácie o polohe jednotlivých vrcholov, textúru objektu a zapuzdruje OpenGL pri vykresľovaní. Dáta dokáže trieda sama načítať zo súboru.

Trieda `WPSpace` reprezentuje FDD mriežku. Obsahuje 3D textúru, informácie o rozmeroch a hustote mriežky, a metódy na prácu s ňou.

Trieda `WPCamera` predstavuje kameru, ktorou možno pohybovať v priestore. Obsahuje informácie o polohe kamery, o spôsobe projekcie a pri vykresľovaní tieto informácie ponúka triede `WPWarpable` v podobe objektu triedy `WPMatrix`

Trieda `WPLight` predstavuje bodové svetlo v hernom svete. Obsahuje informácie o polohe, farbe a intenzite svetla.

Trieda `WPCollider` sa stará o detekciu kolízií. Reprezentuje objekt v tvare kvádra schopný zaregistrovať kolíziu ktorejkoľvek jeho steny s `WPWarpable`. Obsahuje informácie o rozmeroch tohto kvádra

Trieda `WPPlayer` simuluje hráča v hernom svete. Obsahuje informácie o jeho polohe a o smere a rýchlosti jeho pohybu. Využíva triedu `WPCamera` pre vytvorenie pohľadu z prvej osoby, triedu `WPCollider` na simuláciu nárazov hráča do prekážok a triedu `WPLight` pre osvetlenie v okolí hráča. Obsahuje metódy na ovládanie hráča, jednoduché fyzikálne

simulácie a dokáže komunikovať s objektom triedy `WPSpace` za účelom deformácie priestoru.

Trieda `WPLevel` zahŕňa do jedného celku všetko potrebné pre vytvorenie jednej hernej úrovne, vrátane geometrie a hráča. Umožňuje načítanie úrovní zo súborov aj ich úspešné a neúspešné ukončovanie.

Kapitola 4

Implementácia

Táto kapitola popisuje niektoré detaily implementácie programu, problémy na ktoré som narazil a ich riešenia.

4.1 Implementačné nástroje

Ako primárne prostredie pre vývoj slúžil Code::Blocks (dostupný zdarma na stiahnutie na http://www.codeblocks.org/) vo verzii 12.11. Z dôvodu kompatibility bol celý projekt neskôr preportovaný do prostredia Microsoft Visual Studio 2015.

Z ďalších programov som použil Blender vo verzii 2.75b na tvorbu trojrozmernej geometrie. Na tvorbu 2D grafiky poslúžil GIMP vo verzii 2.8.16

4.2 Načítanie externých súborov

Aplikácia načítava 3D modely zo súborov vo formáte OBJ. V súbore OBJ sú uvedené najprv priestorové súradnice jednotlivých vrcholov, potom textúrové súradnice vrcholov a nakoniec sú definované jednotlivé polygóny podľa indexov. Ak by aj aplikácia používala na vykresľovanie indexované buffery, načítanie modelu by bolo jednoduché a spočívalo by len vo vkladaní hodnot do bufferov za sebou tak ako sa postupne čítajú zo súboru.

Indexované buffery však prinášajú jedno obmedzenie a tým je nemožnosť použiť v rôznych polygónoch rôzne textúrové súradnice pre ten istý vrchol. Preto ak polygóny zdieľajú nejaký vrchol, v pamäti musí byť reprezentovaný ako dva samostatné vrcholy so zhodnými súradnicami.

Preto pri načítavaní OBJ súboru sa hodnoty najprv ukladajú do pomocných polí a až keď sú všetky hodnoty načítané, vytvoria sa jednotlivé polygóny v bufferoch podľa načítaných súradníc a indexov. Toto môže trvať pomerne dlho – až sekundy na slabších počítačoch – ale pretože k tomu dochádza jednorázovo pri spustení danej úrovne, nie je to podstatné. Napokon hráči sú zvyknutí na niekoľkosekundový loading pri spustení väčšiny hier.

Okrem geometrie sa z OBJ súboru načítava ešte jedna informácia, a tou sú súradnice, na ktoré sa treba dostať pre dokončenie úrovne. Keďže ide iba o tri čísla, zdalo sa mi zbytočné načítavať túto informáciu zo samostatného súboru. OBJ súbor však štandardne takéto informácie neobsahuje, musel som ich tam teda napísať ručne. Nachádzajú sa vždy na prvom riadku súboru.

Okrem toho ešte aplikácia načítava textúry vo formáte BMP. To sa deje jednoducho pomocou funkcie *SDL_LoadBMP* a následným vložením do pamäte OpenGL.

4.3 Prispôsobenie rýchlosti

Hra beží v nekonečnom cykle, pričom pri každom prechode týmto cyklom sa vykoná niekoľko základných akcií. Tými sú ošetrenie vstupu, aktualizácia stavu hráča a vykreslenie všetkej grafiky na výstup. Toto však trvá na výkonovo sa líšiacich počítačoch rôznu dobu a celý tento cyklus beží s rozdielnou frekvenciou. Dokonca aj na tom istom počítači sa môže táto frekvencia počas behu meniť. To spôsobuje, že celá hra beží o poznanie pomalšie alebo rýchlejšie, mení sa rýchlosť hráčovej chôdze a pocitová senzitivita myši.

Na riešenie tohto problému bolo potrebné v prvom rade zistiť, aký čas trvá jedna iterácia. Na to som využil funkciu *SDL_GetTicks*, ktorá vracia čas v milisekundách uplynutý od štartu aplikácie. Od tejto hodnoty sa odčíta hodnota zistená v minulej iterácii, čím dostaneme čas celej jednej iterácie v milisekundách. Pri tomto prístupe hrozí pretečenie premennej, určujúcej čas od štartu aplikácie, avšak pri 32-bitovej hodnote by to trvalo päťdesiat dní, a tak dlho zrejme aplikácia nikdy nepobeží.

Keď už je známy čas trvania jednej iterácie, táto hodnota sa vydelí priemernou hodnotou, ktorú som meraním určil na 16 milisekúnd, a výsledok sa použije ako násobič pre všetky akcie, ktoré treba zrýchliť alebo spomaliť na udržanie konzistentnej rýchlosti. Ak je teda čas jednej iterácie menší ako 16 milisekúnd, násobič bude menší ako 1 a za jednu iteráciu sa teda zmení stav hry v menšej miere, ako by sa zmenil za 16 milisekúnd.

Vložiť do cyklu oneskorenie, a tým vynútiť rovnakú frekvenciu bez ohľadu na výkon by bolo tiež možné riešenie, avšak menej efektívne.

4.4 Beh úrovne

Pre úspešné začatie a ukončenie úrovne je potrebné poznať súradnice, z ktorých hráč štartuje a súradnice kam sa má dostať. Pre zjednodušenie sú úrovne navrhnuté tak, aby startovacia pozícia bola vždy v počiatku súradnicového systému. Tým odpadá nutnosť na začiatku nastavovať pozíciu hráča, inicializuje sa automaticky na nulu.

Súradnice koncovej pozície sú zadané v súbore(viď 4.2), ale nie je možné ich brať úplne presne. Je totiž takmer nemožné aby sa hráč dostal presne na tieto súradnice, ktoré su k tomu určené číslom s pohyblivou rádovou čiarkou. Testuje sa teda vzdialenosť hráča od týchto súradníc, a ak klesne pod určitú hranicu, dá sa povedať že hráč je približne v cieli a úroveň skončí úspešne.

Level končí neúspešne ak hráč vypadne mimo všetku geometriu. Mali by teda porovnávať súradnice hráča s maximálnymi súradnicami vrcholov v každom smere. V praxi sa však testuje len to, či sa hráč nachádza nižšie ako najspodnejší vrchol. Ak totiž hráč naozaj vypadne mimo geometriu, vplyvom gravitácie padá pomerne rýchlo nadol. Testovať ostatné maximá je teda zbytočnosť. Ak by sa hráčovi podarilo dostať nad geometriu, čo je veľmi nepravdepodobné, spadol by znovu do vnútra a mohol pokračovať v hre.

Pri testovaní, či je hráč pod geometriou sa uplatňuje istá tolerancia. To preto aby si hráč stihol ešte pred tým ako úroveň skončí uvedomiť, čo sa vlastne stalo.

4.5 Osvetlenie

Hra používa difúzne osvetlenie, intenzita svetla pre každý fragment teda závisí od vzdialenosti fragmentu od zdroja svetla a od uhlu dopadu. Tieto informácie však štandardne vo fragment shaderi nie sú k dispozícii.

Na vypočítanie vzdialenosti fragmentu a zdroja svetla potrebujeme vedieť jeho pozíciu po deformácii ale ešte pred transformáciou do súradníc obrazovky. Vertex shader teda musí na výstupe poskytnúť súradnice každého vrcholu po projekcii, aby sa správne vykreslila jeho poloha, ale aj pred ňou kvôli výpočtu osvetlenia. K tomu som použil vlastnú premennú.

Na vypočítanie uhlu dopadu potrebujeme vedieť normálu povrchu, na ktorom sa fragment nachádza. Tú získame deriváciami polohy fragmentu v dvoch smeroch pomocou funkcií *dFdx* a *dFdy* a ich následným vektorovým súčinom. Uhol dopadu sa potom jednoducho vypočíta skalárnym súčinom.

4.6 Detekcia kolízií

Detekcia kolízií používa metódu uvedenú v podkapitole 3.4. Namiesto šiestich kamier je však v systéme len jedna, ktorá sa postupne otáča po 90-stupňových krokoch do každého smeru. Pohľad kamery sa vykresľuje do textúry o rozmeroch 64x64.

4.7 Deformácia

Deformácia sa prepočítava na grafickej karte. Pri samotnom ovplyvňovaní FFD mriežky sa za účelom pararelizácie využíva compute shader, ktorý načítava údaje o bodoch mriežky z 3D textúry. Poloha každého bodu sa určí ako súčet súradnic daného texelu v textúre a jeho hodnoty. Táto poloha sa potom použije na výpočty podľa podkapitoly 3.3 a nové hodnoty sa zapíšu späť do textúry.

Textúra potrebuje tri kanály na vyjadrenie posunu po každej z troch osí. Z neznámeho dôvodu však OpenGL nepodporuje pri trojkanálovú textúru RGB ako platnú štruktúru, z ktorej možno v compute shaderi čítať aj do nej zapisovať. Preto bola použitá štvorkanálová textúra RGBA, v ktorej alfa hodnota je vždy nula. Tým sa zvýšili pamäťové nároky o štvrtinu.

Táto textúra sa ďalej používa pri vykresľovaní vo vertex shaderi. Podľa počiatočnej polohy vrcholu sa navzorkuje hodnota posunu z textúry pomocou trilineárnej interpolácie, posun vrcholu teda odpovedá váženému priemeru posunu uzlov FFD mriežky v jeho okolí.

Kapitola 5

Možnosti rozšírenia

Hra popísaná v tejto práci má podľa môjho názoru veľký potenciál, bohužiaľ rozsah tejto práce a nedostatok času mi nedovolil implementovať všetky funkcie, ktoré by som v tejto hre rád videl. Tieto funkcie by som v hre rád dokončil v budúcnosti, napríklad vo forme diplomovej práce. Sú to hlavne tieto:

- **Väčší počet úrovní:** V časovej tiesni som bol schopný vytvoriť len malý počet úrovní, ktoré sú použiteľné skôr na demonštráciu, ako na skutočnú zábavu. Bolo by dobré vytvoriť rozsiahlejšiu sériu úrovní s rozumnou krivkou obtiažnosti, ktoré by mohli oveľa lepšie potrápiť mozgy hráčov. Dokonca by bolo možné pri postupe hrou rozpovedať aj nejaký príbeh.
- **Viac možností deformácie:** Pri deformácii priestoru má hráč zatiaľ na výber len stlačenie alebo natiahnutie priestoru. V podkapitole 3.3 som spomenul možnosť zatočenia, ktorú som však kvôli niekoľkým problémom nestihol včas implementovať. Pri dostatku času by bolo určite možné vymyslieť ešte viac možných spôsobov deformácie čím by sa znížila jednotvárnosť a zvýšila obtiažnosť.
- **Pokročilešia fyzikálna simulácia:** Zatiaľ jediná fyzikálna simulácia v hre je hráč narážajúci do stien a padajúci pod vplyvom gravitácie. S lepším systémom na detekciu kolízií by bolo možné do hry pridať viac dynamických objektov podliehajúcich fyzikálnym zákonom, napríklad platformy, ktoré by sa dali presúvať z miesta na miesto, kamenné gule ktoré by sa mohli kotúlať po hráčom zdeformovanej podlahe a iné. Samotná deformácia by mohla byť do určitej miery fyzikálne simulovaná a dodať tak objektom akúsi pružnosť.
- **Nepriatelia a pasce:** Vypadnutie mimo priestor úrovne je jediná možnosť ako prehrať. Nepriatelia s určitou úrovňou umelej inteligencie by mohli ohrozovať hráča a nútiť ho myslieť za pochodu. Pasce umiestnené v hernom svete by mohli výrazne skomplikovať riešenie hádaniek. Smrť len na krok od cieľa by mohla dať hráčom presne ten pocit frustrácie, ktorý majú niektorí ľudia radi.
- **Hra pre viac hráčov:** Možnosť zahrať si hru s kamarátmi by urobila hru niekoľkonásobne zábavnejšou. Hráči by mohli spolupracovať pri riešení hádaniek s ktorými by si jeden hráč neporadil alebo naopak súťažiť medzi sebou. Navyše každý hráč by mohol mať vlastnú verziu FDD mriežky, takže hoci by sa nachádzali všetci v tej istej úrovni, každý by ju videl zdeformovanú inak, a tým by sa aj ovplyvnil jeho pohyb po nej.

- **Podpora virtuálnej reality:** Aj keď sa to volá virtuálna realita, nemusí simulovať len realitu. Ocitnúť sa v surrealistickom svete tejto hry by mohol byť zaujímavý zážitok. Bolo by zaujímavé zistiť, ako by sa so všemožne deformujúcim sa svetom vysporiadal žalúdok hráča. V každom prípade stereoskopický náhľad na hru by určite pomohol pri riešení niektorých hádaniek.

Kapitola 6

Záver

Cieľom tejto práce bolo vytvoriť počítačovú hru založenú na deformácii priestoru. Tento cieľ som splnil. Najväčší dôraz bol pri tom kladený na samotnú techiku deformácie a na detekciu kolízií s deformovanou geometriou, než na iné aspekty hry.

Pri písaní práce boli preskúmané rôzne spôsoby riešenia a bola zhodnotená ich vhodnosť. Hra vo veľkej miere využíva výpočetný výkon moderných grafických kariet.

Výsledná aplikácia dokáže načítavať otextúrované modely vo formáte OBJ a zobrazovať ju v reálnom čase s dynamickým osvetlením pomocou knižnice OpenGL. Umožňuje hráčovi pohybovať sa v hernom svete v pohľade s prvej osoby s voľným rozhliadaním, dokáže detekovať kolízie s prekážkami a obsahuje jednoduchú simuláciu gravitácie. Ďalej umožňuje hráčovi ovplyvňovať geometriu úrovne, stláčať a rozťahovať ľubovoľné oblasti pomocou FFD mriežky.

Hra obsahuje tri plne funkčné herné úrovne, ktoré je možné dokončiť za použitia princípov známych zo žánru skákačiek a vyššie spomenutej deformácie.

Ako podklady boli využité knihy a články uvedené v použitej literatúre, konzultácie s vedúcim práce a vlastné skúsenosti mňa ako programátora a hráča počítačových hier. Aplikácia bola naprogramovaná v jazyku C++ za použitia knižníc OpenGL a SDL.

Literatúra

[1] Dawson, M.: *Beginning C++ Through Game Programming*. Delmar Learning, Čtvrté vydání, 2014, ISBN 1305109910, 9781305109919.

[2] Ericson, C.: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology, Elsevier Science, 2004, ISBN 9780080474144.
URL https://books.google.cz/books?id=0MvuykjoW_IC

[3] Fadnavis, S.: Image Interpolation Techniques in Digital Image Processing: An Overview. *International Journal of Engineering Research and Applications (IJERA)*, ročník 4, č. 10, 2014: s. 70–73, ISSN 2248-9622.

[4] Klosowski, J. T.; Held, M.; Mitchell, J. S.; aj.: Efficient collision detection using bounding volume hierarchies of k-DOPs. *Visualization and Computer Graphics, IEEE Transactions on*, ročník 4, č. 1, 1998: s. 21–36.

[5] Meagher, D.: High-speed image generation of complex solid objects using octree encoding. Srpen 28 1985, eP Patent App. EP19,850,100,151.
URL http://www.google.com/patents/EP0152741A2?cl=en

[6] Michal Turek: SDL: Hry nejen pro Linux. [Online; citované 22.4.2016].
URL http://www.root.cz/clanky/sdl-hry-nejen-pro-linux-1/

[7] Pavel Tišnovský: Grafická knihovna OpenGL. [Online; citované 22.4.2016].
URL http://www.root.cz/clanky/graficka-knihovna-opengl-1/

[8] Raffin, R.: Free Form Deformations or Deformations Non-Constrained by Geometries or Topologies. In *Deformation Models*, ročník 7, editace M. G. Hidalgo; A. M. Torres; J. V. Gómez, kapitola 2, Springer Netherlands, první vydání, 2013, s. 49–74.

[9] Weber, O.; Sorkine, O.; Lipman, Y.; aj.: Context-Aware Skeletal Shape Deformation. *Computer Graphics Forum*, ročník 26, č. 3, 2007: s. 265–274, doi:10.1111/j.1467-8659.2007.01048.x.

Prílohy

Zoznam príloh

Príloha A

Obsah CD

source	zdrojové súbory programu
Warp	spustiteľná hra
bakalárska práca.pdf	elektronická verzia tejto technickej správy
src dokumentácia	zdrojové súbory technickej správy
video.avi	prezentačné video
README.txt	základné informácie o inštalácii a ovládaní