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

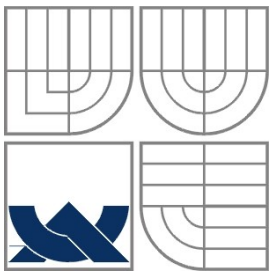
DEMONSTRACE POKROČILÝCH TECHNIK VYUŽÍVAJÍCÍCH OPENGL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

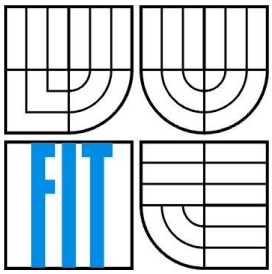
AUTOR PRÁCE
AUTHOR

Bc. ANTONÍN BUČEK

BRNO 2008



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

DEMONSTRACE POKROČILÝCH TECHNIK VYUŽÍVAJÍCÍCH OPENGL

DEMONSTRATING OF ADVANCED TECHNIQUES USING OPENGL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ANTONÍN BUČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2008

Abstrakt

Cílem této práce je v OpenGL demonstrovat pokročilé zobrazovací techniky. Práce je pojata jako demo bez omezení velikosti. Zabývá se vytvořením 3D světa, konkrétně komplexu vnitřních prostor, jeho obohacením o různé prvky, použitím osvětlování a stínů. Je popsána technika částicových systémů, pomocí níž byly vytvořeny výbuchy, dým a další částicové efekty. Děj v demu vytvářejí 3D modely strojů, které mezi sebou bojují laserovými paprsky. Důležitou částí je pohybový model, který je využit jak pro pohyb modelů, tak avatara. Práce je obohacena i o zvukovou stopu.

Klíčová slova

demo, OpenGL, 3D svět, částicové systémy, 3D modely, laserové paprsky, osvětlení, stíny, pohyb, zvuk

Abstract

This thesis demonstrates advanced techniques using OpenGL. The work is conceived as a demo without any size restrictions. It focuses on creating 3D world, which consists of inner rooms complex. Some additional interesting objects, lighting and shadows enrich the rooms. Particle systems, which were used to create explosions, smokes and some other particle effects are described in the thesis, too. 3D machine models fight between themselves using laser beams and form the story of the demo. Important part of the work is a movement model which is applied to both models and avatar. Sound track completes the demo.

Keywords

demo, OpenGL, 3D world, particle systems, 3D models, laser beams, lighting, shadows, movement, sound

Citace

Antonín Buček: Demonstrace pokročilých technik využívajících OpenGL. Brno, 2008, diplomová práce, FIT VUT v Brně.

Demonstrace pokročilých technik využívajících OpenGl

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Adama Herouta Ph.D. a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Antonín Buček
19.5.2008

Poděkování

Chtěl bych zde poděkovat především Ing. Adamu Heroutovi, Ph.D. za informace poskytnuté k této práci a Bc. Petře Kohoutkové za jazykovou korekturu.

Obsah

1 Úvod

Počítačová grafika dosáhla takového stupně dokonalosti, že jsou pomocí ní ve filmech generovány celé scény, aniž by divák poznal, že se nejedná o skutečnost, ale pouze o počítačem vytvořenou fikci. Modelování a zobrazování vizuálně dokonalých scén umožňují především fyzikální simulace jistých fenoménů (osvětlení, stínování, amorfní vlastnosti látek, simulace kouře, plazmy,...), které sice poskytují realistické výsledky, ale jejich výpočet je časově velmi náročný. Renderování takových scén se proto neprovádí v reálném čase, což ovšem ve filmovém průmyslu není kritické. Tam jde o konečný, co nejlepší, výsledek.

Naproti tomu v jiných oblastech počítačové grafiky, a zejména v oblasti počítačových her, je právě real-time výpočet a renderování jedním z nejdůležitějších kritérií. Raději jsou tolerovány vizuální nedokonalosti a chyby v zobrazování než časové zpomalení. Přesto už v dnešní době lze jen těžko odlišit, jedná-li se o nový hollywoodský trhák nebo zbrusu novou počítačovou hru. Místo fyzikálních simulací se pro real-time zobrazování většinou hledají jiné, rychlejší algoritmy a postupy, které by dosáhly uspokojivých výsledků v akceptovatelném čase. Buď se vychází z fyzikálních modelů, které se zjednodušují a aproximují výpočetně méně náročnými algoritmy, nebo jsou vytvořeny zcela odlišné techniky.

Tato práce se zabývá právě real-time vykreslováním a jejím cílem je demonstrovat několik pokročilých technik zobrazování, které by realisticky napodobovaly skutečnost. Jejím výstupem bude demo bez omezení velikosti, které je právě pro takové účely vhodné. V demu bude pozorovatel procházet podzemním komplexem chodeb a místností a pozorovat různé zajímavosti. Čím hlouběji bude, tím více efektů bude moci vidět. Stěžejní část efektů je spojena s létajícími stroji z budoucnosti, které pozorovatel při své procházce v podzemí objeví. Tyto stroje však nejsou jen statickou kulisou, ale budou se pohybovat a navzájem mezi sebou bojovat. To dává prostor k vytvoření laserů, výbuchů a dalších efektů.

V první kapitole bude jen velice stručně a přehledově popsáno prostředí, ve kterém bude demo vyvíjeno. Čtenář si tak bude moci utvořit představu o implementačních nárocích, složitosti a apriori znalostech, potřebných pro takovou práci.

Následně bude popsán model 3D světa, komplex místností a chodeb dále označovaný jen jako podzemí. Představena bude struktura uložení, postup vykreslování a řešení viditelnosti. K tomu se váží i objekty rozmístěné v podzemí a další zvláštnosti, které bude možné v demu vidět.

První skutečnou modelovací technikou zmíněnou v této práci budou částicové systémy. Kapitola obsahuje jak obecný nástin částicových systémů, tak i konkrétní příklad využití, kdy budou pomocí nich modelovány realistické výbuchy, kouře, šlehající plameny a další.

Podstatnou částí práce je vytvoření modelů. Kromě dvou hlavních modelů, raket a vznášedel, jsou vymodelovány i další drobné objekty, např. světla nebo pochodně. Kapitola se nezabývá příliš detailně samotným vytvářením modelů, což je záležitost spíše konkrétního modelování v konkrétním nástroji, ale především problematikou načtení popisujícího formátu, zde VRML, a zobrazení v aplikaci.

K modelům se úzce váže problematika jejich střel, což jsou laserové paprsky. Protože jsou vytvořeny precizním způsobem, je jim věnována samostatná kapitola.

Dále bude čtenář seznámen s využitím světel v demu. Pomocí nich bylo dosaženo několika pěkných efektů. Kromě obecných postupů bude zmíněno i několik konkrétních nastavení a typů na řízení osvětlení. K osvětlení se váže i problematika stínů, která bude také rozebrána.

Zásadní kapitolou pro tuto práci bude část popisující problematiku pohybu avatara[1] a bojujících strojů v podzemí. Kromě popisu základního modelu pohybu budou zmíněny i některé důležité aspekty a také problémy vycházející z tohoto konceptu.

Krátká část bude věnována i zvuku. Přestože tvoří důležitou kulisu k demu a dodává mu napětí a dynamičnost, je programátorsky naprosto triviální a práce potřebná k vytvoření hudebního pokladu byla spíše uměleckého a tvořivého charakteru.

Na závěr je umístěn popis několika dalších problémů a efektů, kterými je možno stávající práci vylepšit či obohatit.

1 Avatar je virtuální postava ovládaná uživatelem, často se vyskytující například ve hrách. Je to ten, za koho bojujeme a jehož očima vidíme. Avatar může být reprezentován jen umístěním kamery, ale může mít také velikost, váhu či jiné fyzikální vlastnosti, které interagují s prostředím, ve kterém se pohybuje.

2 Použité nástroje a knihovny

Nelehkým úkolem před započetím každé práce je zvolení implementačního nástroje, popř. prostředí, či jejich kombinace. Pokud budou vybrány špatné nástroje, může se vývoj zbytečně protáhnout, mohou nastat nečekané komplikace nebo nemusí být implementace vůbec možná. Příčinou toho bývá například nekompatibilita zvolených nástrojů.

Naopak volbou vhodných nástrojů se může vývoj velice urychlit a je možné se vyhnout mnoha nepříjemnostem. Je proto dobré se vždy nejdříve seznámit s požadavky na aplikaci, její kompatibilitu, rozšiřitelnost, či s návazností na další práce. Teprve potom je možno vybrat vhodný implementační jazyk, prostředí a další programy, nutné pro vývoj.

Poslední část kapitoly je věnována implementačním detailům, týkajících se celého programu jako celku. Ten je napsán v jazyce C++ a využívá grafickou knihovnu OpenGL. O vytvoření okna a obsluhu událostí se stará přímo WinAPI.

2.1 Grafická knihovna OpenGL

V oblasti grafiky všem programátorům dobře známá knihovna OpenGL je přenositelnou alternativou k DirectX. V podstatě se jedná o programové vybavení pro grafické akcelerátory. Lze pomocí něj vykreslovat jen základní primitiva, na rozdíl od DirectX nepodporuje načítání ani ukládání různých grafických formátů a nepodporuje ani práci s okny v systémech. Přesto je OpenGL velice rozšířeno a hojně používáno, a to zejména díky své nezávislosti na platformě.

Přes zmíněná omezení OpenGL ve svých schopnostech nezaostává za DirectX, ale je s ním naprosto rovnocenné. Umožňuje texturování, blending, výpočet hloubky a mnoho dalšího. Nejnovější vlastnosti, umožňující pokročilé zobrazování, jsou dostupné přes rozšíření OpenGL.

Protože je OpenGL známé, není třeba o něm dále diskutovat. Pro podrobné informace je možné nahlédnout do [1] nebo na internetu navštívit [2].

2.2 Blender

3D modelovacích nástrojů je na dnešním trhu velké množství. K nejznámějším patří 3D Studio MAX, Cinema 4D, Maya, Lightwave atd. Bohužel jsou to všechno komerční programy a za jejich pořízení je třeba zaplatit nemalé částky.

Z tohoto hlediska je snad jediným vhodným 3D modelovacím nástrojem Blender, který si ve svých možnostech s placenými nástroji vůbec nezadá. Poskytuje široké modelovací techniky, texturování, nastavení materiálu, osvětlení, práci s částicovými systémy a v neposlední řadě i tvorbu

animací. Přínosem pro tuto práci je, že umí exportovat modely do VRML. Je tedy možné si v Blenderu vytvořit vlastní model, uložit ho do VRML a pak už jen načíst a použít v aplikaci. Není potřeba model zdlouhavě ručně vytvářet, což by zejména v případě složitých modelů ani nebylo možné. Více o práci v Blenderu je možné nalézt v [3] nebo na internetu [4], kde je i zdarma ke stažení.

2.3 Další programy

Při vytváření rozsáhlejších prací je většinou nutné použít i několik dalších podpůrných programů, které jsou vhodné pro specifické účely nebo na konkrétní úkoly. V této práci to bylo zejména Flux Studio [18], které doplňovalo Blender při exportu a úpravách modelů a jeho konkrétní využití je popsáno podrobněji níže. Stejně tak bylo potřeba pro vytvoření modelů nakreslit jejich UV textury, k čemuž byl s výhodou využit free kreslící multiplatformní program Gimp [19]. Ten byl také použit při tvorbě stínových map. Audacity [20] je program pro zpracování zvuku a při vytváření této práce byl využíván hlavně k výslednému mixu jednotlivých zvukových efektů a stop a k úpravě jejich hlasitosti.

Jistě by bylo možné jmenovat další programy, které byly použity. Ty ale měly minoritní zastoupení, takže nebyly pro vývoj práce esenciální, nebo se jednalo spíše o běžné rutinní úkoly v klasických programech.

2.4 Knihovny

Přehledově by zde bylo vhodné uvést i knihovny, kterých tato práce hojně využívá. První dvě jsou free, volně stažitelné a použitelné knihovny pod licencí GPL. Jsou to knihovna Colony pro práci s částicovými systémy [5], ze které se kromě samotných částicových systémů hojně využívá i ovládání kamery a práce s vektory, a knihovna pro načítání VRML souborů [17].

Knihovna obsahující jedinou třídu pro práci s bodovým fontem, je využita pro vytvoření ohnivého písma. Je převzata z projektu, který byl určen do předmětu Pokročilá počítačová grafika a na jehož realizaci se podílel tým řešitelů.

Poslední důležitou knihovnou použitou v této práci je multiplatformní knihovna Glut, o níž se lze mnoho dozvědět na [25]. Základem této nadstavbové knihovny je podpora pro práci s okny (včetně zpracování událostí), pop-up menu a fonty a to z toho důvodu, že tyto činnosti nejsou v knihovně OpenGL kvůli přenositelnosti přímo podporovány.

3 Vytvoření prostředí dema

Žádná počítačová hra ani filmová scéna se neobejde bez prostředí, ve kterém se odehrává děj nebo příběh. Může to být příroda, les, poušť, může se jednat o město nebo jinou zástavbu, o vnitřní prostory, domy nebo skladiště. Dokonce se může děj odehrávat ve vesmíru, na kosmické lodi nebo na jiné planetě. V tomto případě bylo jako prostředí 3D světa zvoleno podzemí, přesněji několik místností propojených chodbami.

3.1 Metody modelování

Vymodelovat podzemí přímo v aplikaci by sice šlo a bylo by to nejpřímější řešení, ale ne zrovna nejšťastnější. Dobrou volbou je podzemí vytvořit v některém 3D modelovacím programu, uložit ho do vhodného formátu a potom načíst do aplikace. Tím se ušetří mnoho času při vytváření modelu, protože není nutné zadávat souřadnice ručně a starat se o návaznost ploch, přímek atd. Také není potřeba kontrolovat nepřehledné množství čísel, ale vše je jasně vidět. Je možné si model prohlédnout již v části návrhu, otáčet s ním nebo zkoušet různé změny a to všechno s vynaložením minimálního úsilí. Na všechny potřebné operace jsou lehce dostupné nástroje programu, kterými lze docílit deformací povrchů, texturování, nasvícení a nepřeberného množství dalších funkcí. Neocenitelnou výhodou je znovupoužitelnost modelu kdekoliv jinde díky možnostem exportu a importu 3D formátů. Kdykoliv se pak rozhodneme model změnit, není potřeba znovu překládat celý program, ale stačí jen vyměnit soubor s modelem.

Přes veškeré klady toho přístupu byl v práci použit ještě jiný. Model podzemí je uložen v externím souboru, ale ručně vytvořený. Důvodů pro to je hned několik. V prvé řadě byl použit základ popsaný v [6], kde již byla navržena struktura uložení, parser i vykreslovací část. Stačilo tedy jen několik vhodných úprav kódu, aby jej bylo možné použít v projektu. V souboru je také uloženo jen to, co je opravdu potřeba. Soubor není zbytečně velký, neobsahuje světla, kameru a jiné zde nepotřebné záležitosti a jeho načítání i uložení tak trvá nejkratší možnou dobu. A konečně bylo možné si tak editaci vyzkoušet ručně, což vedlo k získání cenných zkušeností, které se budou hodit i při práci s 3D modelovacími programy a jejich formáty. Nicméně tento přístup bylo možné použít především díky tomu, že model podzemí nebyl velký. Při vytváření větších 3D světů je rozhodně vhodné některý z modelovacích programů použít.

3.2 Model podzemí

Základ struktury uložení i algoritmus načítání a vykreslování je převzat z [6]. Bylo jej však třeba rozšířit a to hlavně kvůli řízení vykreslování, které je popsáno v samostatné kapitole 4. Zde stačí konstatovat, že je podzemí rozděleno na určité sektory, vetšinou odpovídající místnostem či chodbám. Celý model je tedy uložen v podobě jednotlivých trojúhelníků, které náleží do konkrétních sektorů. Tato struktura je uložena v textovém souboru `World.txt`. Každý trojúhelník navíc obsahuje informace o textuře, texturovacích souřadnicích a normále.

3.2.1 Formát uložení dat

Na začátku souboru je krátká hlavička, která udává, kolik sektorů je v souboru celkem uloženo. Tato informace slouží pro alokaci paměti v době načítání modelu. Dále jsou už za sebou uloženy jednotlivé sektory. Každý sektor začíná opět hlavičkou s počtem trojúhelníků v tomto sektoru, aby bylo možné alokovat paměť, a následuje odpovídající počet trojúhelníků, náležejících aktuálnímu sektoru.

Ve struktuře trojúhelníku je *ID_textury* číslo textury, která se má na daný trojúhelník nanést. *smer_normaly* je pomocný údaj, jehož význam je popsán v části o osvětlení. V podstatě se jedná o identifikaci směru normály, tudíž je ihned známo, jak je trojúhelník v aplikaci natočen a není potřeba to složit zjišťovat z hodnot jednotlivých souřadnic normály. *n* určuje normálu trojúhelníku ve 3D (má tedy tři složky x, y, a z). *v1*, *v2* a *v3* jsou vrcholy trojúhelníku a *t1*, *t1* a *t3* jim odpovídající texturovací souřadnice. Struktura vypadá následovně:

NUMSECTORS *pocet_sektoru*
NUMPOLLIES *pocet_trojuhelniku*

ID_textury smer_normaly
n.x n.y n.z
v1.x v1.y v1.z t1.x t1.y
v2.x v2.y v2.z t2.x t2.y
v3.x v3.y v3.z t3.x t3.y (1)

//další sektory a trojúhelníky

Důležité pro formát je oddělení jednotlivých hodnot minimálně jedním bílým znakem a zachování řádkování. Vynechané volné řádky mají jen optický význam. Také je možné používat komentáře, klasicky označené //.

3.2.2 Načítání a vykreslování

Po spuštění aplikace se najednou načte celý model do paměti. Vzhledem k jeho malé velikosti je to výhodnější, než jej načítat v průběhu animace, což by ji mohlo zdržovat. Při načítání se dynamicky vytvoří pole sektorů o odpovídající délce. Data každého trojúhelníku se vloží do jedné struktury a ty pak do prostého dynamického pole v sektoru, do kterého trojúhelník patří. Při vykreslování sektoru se toto pole jednoduše projde a jednotlivé položky struktury trojúhelníku se přímo vloží do parametrů vykreslovacích funkcí OpenGL. Sektory se vykreslují v závislosti na pozici avatara, což je podrobněji popsáno v kapitole 4.

Celá struktura je tedy uložena v dvojrozměrném poli, jejíž oba rozměry jsou dynamické. První rozměr je dán počtem sektorů a druhý rozměr počtem trojúhelníků v tomto sektoru. Typem prvku pole je struktura trojúhelníku popsaná výše.

3.3 Statické objekty, předměty

Naprosto totožné zpracování probíhá i se statickými objekty, umístěnými do podzemí pro jeho oživení. Jedná se o barely a tři typy beden. Jejich modely jsou tentokrát uloženy v Display listu přímo v programu. Aby bylo možné objekty po podzemí libovolně rozmístit, byl opět vytvořen konfigurační soubor.

3.3.1 Formát uložení dat

Datový soubor nese název `Items.txt` a jeho struktura je následující:

NUMSECTORS *pocet_sektoru*
NUMITEMS *pocet_objektu*

typ_objektu rotace_objektu
x y z (2)

//následující sektory a objekty

Na začátku souboru je opět nejdříve specifikováno, kolik sektorů se v souboru nachází. Tento počet nemusí odpovídat počtu sektorů v modelu podzemí. Jednotlivé sektory jsou opět uvozeny hlavičkou a počtem objektů, které do daného sektoru patří. Za úvodní hlavičkou sektoru následují parametry pro jednotlivé objekty. Těch je podstatně méně než pro trojúhelníky. Prvním parametrem je typ objektu, podle nějž se rozhodne, který objekt se bude umisťovat (zda se jedná o barel, bednu, atd.). Protože jsou modely statické a navíc vytvořené přímo v kódu, mohly by působit by příliš jednotvárně. Tomu se lze vyhnout tím, že budou různě natočeny. Bylo by proto možné do

konfiguračního souboru ukládat stupně otočení. Tím by ale bylo dosaženo pouze rotace kolem jedné, pevně zvolené osy. Pro některé předměty je to dostačující, jiné je však vhodné natočit i kolem více os.

Pokud by měl konfigurační soubor postihnout i tento případ, bylo by nutné buď zadávat v mnoha případech mnoho zbytečných parametrů, nebo by vznikly volitelné parametry, což by načítání zkomplikovalo. Vzhledem k jednoduchosti celého konceptu byla situace vyřešena tak, že místo konkrétních hodnot natočení je do souboru uloženo identifikační číslo, reprezentující jeden typ obecně libovolné transformace. Toto číslo může zastupovat jak jednoduché natočení kolem jedné osy, tak jakoukoliv jinou transformaci nebo i jejich složení (například rotaci kolem obecné osy, posunutí, změnu měřítka a zkosení). To nám dává silný nástroj při zachování jednoduchosti konfiguračního souboru. Jedinou nevýhodou je to, že si musíme všechny transformace předpřipravit a používat pak jen tyto. Nelze jednoduše, bez nového překladu programu, přidat novou transformaci v konfiguračním souboru. Nicméně už i pouhých deset různých transformací dostačuje k tomu, aby se daly objekty rozmístit tak, že vypadají, jako by je do scény opravdu někdo náhodně poházel, a nepůsobí nijak uniformně a strojeně.

Neuspořádanost lze ještě podpořit tím, že bedny naskládané v řadě budou každá mírně posunutá do stran nebo i mírně vysunuty či zasunuty z řady. Toho lze dosáhnout přímo umístěním na požadované místo a slouží k tomu poslední tři parametry v konfiguračním souboru, což jsou prosté souřadnice objektů. Nejsou to ale souřadnice jejich středů, nýbrž středů jejich podstav. Proto už není třeba myslet na případné posunutí v ose y, aby nebyly objekty napůl „v podlaze“, ale stačí je umístit relativně k dané místnosti.

3.3.2 Načítání a vykreslování

Jedná se zcela o triviální záležitost. Podle identifikačního čísla objektu se zavolá příslušný Display list, na ten se aplikuje podle čísla ze souboru příslušná transformace a umístí se na zadané souřadnice.

3.4 Zvláštnosti v podzemí

Pro další oživení a doplnění podzemí bylo vytvořeno ještě několik speciálních objektů (které nejsou externí modely, o nichž bude pojednáno níže). První dva jsou mříž, přes kterou lze vidět do další místnosti, a plot. V obou případech se jedná o textury s definovanou průhledností. Správným zapnutím alfa míchání (viz [1]) v OpenGL jsou pak textury zobrazeny podle požadavků a přes části s průhledností je vidět za ně.

V jedné z místností je vytvořeno koryto, ve kterém teče voda. Ta je simulována prostou texturou vody s nastavenou průhledností a správně nastavenými parametry alfa míchání. Navíc se

textura pohybuje ve směru toku vody, čehož je dosaženo pravidelnou aktualizací texturovacích souřadnic na čtverci hladiny. Přes tento podzemní kanál vede jednoduchý mostek.

Jedna z místností má otevřený strop, kterým je vidět hvězdná obloha. Na ní je vykreslen měsíc, který je vytvořen jako billboard. To nám dává možnost umístit malou texturu přímo do místnosti a nepotřebujeme obří texturu na pozici skutečného měsíce, která by se i při pohybu a pohledu z různých úhlů zdála pořád statická jako měsíc na obloze. Takže i když je billboard kousek od kamery, díky tomu, že je natočen stále k pozorovateli a jeho okraje jsou průhledné, se zdá, jako by byl měsíc na své skutečné pozici daleko ve vesmíru.

4 Řízení vykreslování

Čím je model podzemí komplikovanější a polygonová síť jemnější, čím více různých efektů se v podzemí vyskytuje, čím více světel je zapnuto, a jednoduše čím složitější je aplikace, tím déle trvá její vykreslování. I při relativně malém komplexu místností se efekty brzy nahromadí a vykreslování začne být trhané a nepěkné. Mezi nejnáročnější efekty patří osvětlování a částicové systémy. Velmi brzy je tuto situaci nutné začít řešit a naprosto nejvhodnější je s tím počítat již od začátku a práci koncipovat nějakým vhodným způsobem.

Používaným a také efektivním způsobem řešení tohoto problému je řízení vykreslování. Jedná se o to, že jsou vykreslovány pouze ty části, které jsou zrovna vidět, a ostatní se nevykreslují. Základní algoritmy jsou popsány v [9] v kapitole Zobrazování rozsáhlých scén. Pro tuto práci byla navržena jednoduchá varianta Portálového řešení.

Kromě řešení viditelnosti lze urychlení vykreslování dosáhnout i vhodnou hustotou polygonové sítě 3D světa. I tato problematika byla v práci řešena a obecně je popsána také v [9].

4.1 Řešení viditelnosti

Viditelnost je v této práci řešena poměrně jednoduchým, ale efektivním způsobem. Celé podzemí je vhodným způsobem rozděleno na osmnáct oblastí a z předchozí kapitoly víme, že je model podzemí rozdělen na sektory. Není pak nic jednoduššího než zařídit, aby se v oblasti, kde se právě nachází avatar vykreslovaly jen ty sektory, které jsou vidět. Pokud máme tento základní koncept, není pak už složité pomocí aktuální oblasti řídit i vykreslování ostatních prvků. Víme, že také objekty jsou rozděleny do sektorů, které přímo korespondují s viditelností v dané oblasti. Dále tak můžeme vykreslovat vhodně jen viditelné modely objektů, stínové mapy, částicové systémy a všechno, co je potřeba zobrazovat. Stejným způsobem lze řídit například aktualizace částicových systémů, výpočet pohybu strojů, změnu osvětlení a všechny ostatní výpočty.

Známe-li již při návrhu rozmístění oblastí i pohyb avatara podzemím, můžeme toho také s výhodou využít a nevykreslovat ty části, které má za sebou a již se na ně ani pohledem nevrátí. Naopak pokud víme, že se bude rozhlížet do všech stran, necháme vykreslovat celé okolí. Vyhneme se tak složité struktuře rozmístění oblastí a režii spojené s častou inicializací nové oblasti.

Mimo to je dobré při návrhu počítat také s osvětlením, aby bylo umožněno vhodné přepínání světel při vstupu do nové oblasti. Blíže tuto problematiku popisuje kapitola Osvětlení a stíny.

Vzhledem k navržení jednotlivých sektorů modelu a oblastí bylo docíleno toho, že se v každé oblasti vykreslují maximálně čtyři sektory. Ve skutečnosti záleží více na velikosti jednotlivých sektorů než na jejich počtu. Nejednou se stalo, že vykreslování jednoho sektoru obsahujícího jedinou

velkou místnost bylo pomalejší než vykreslení oblasti se čtyřmi sektory. Podobně v případě, že některý z velkých sektorů je vidět v pozadí, přes okno nebo skrze chodbu, může na jeho zobrazení být potřeba nejvíce času, přestože ve scéně hraje sekundární roli a jen ji uzavírá, aby nebylo vidět černé pozadí. Řešením těchto situací je umožnit zobrazování různých sektorů s různým počtem detailů. Tomu se podrobněji věnuje následující podkapitola.

4.2 Řízení hustoty polygonové sítě

Při prvotní tvorbě podzemí stačily pro zobrazení jedné stěny dva trojúhelníky, na které se nanesla textura. Tímto způsobem šlo zobrazit i rozsáhlejší podzemí najednou, protože dohromady nebyl počet vrcholů nijak extrémní. Velmi záhy se však ukázalo, že dva trojúhelníky na jednu velkou stěnu je opravdu málo. Bylo to ve chvíli, kdy bylo zapnuto osvětlení. Pokud máme u velké stěny umístěný slabý zdroj světla, který má osvětlovat jen malé okolí, není toto osvětlení vůbec vidět, protože v okolí zdroje světla nejsou žádné vrcholy, na kterých by se osvětlení počítalo, a na vzdálené vrcholy v rozích stěn už nemá vliv. Bylo tedy nutné zajistit automatické rozdělení stěn na větší počet polygonů.

Samotné rozdělení nebylo příliš složité, stačilo vědět, v jakém pořadí jsou v souboru uloženy vrcholy trojúhelníků, z nich se vzaly rohové vrcholy a mezi ně se vložily iterativně další čtvercové polygony s určitým krokem, podle požadavku na jejich hustotu.

Mnohem složitější bylo zajistit mapování textur na tyto nové vrcholy a vedlo to k poměrně rozsáhlému kusu kódu. Problémem textur na rozdíl od polygonů je to, že na vrcholy mohou být namapovány mnoha způsoby, ať už se jedná o natočení textury či její opakování. Navíc je potřeba znát i orientaci normály vzhledem ke světlu pro správné osvětlení. Už i pouhé zjištění tohoto faktu ze souřadnic vrcholů trojúhelníků stěn není zcela triviální a stálo by drahocenný výpočetní výkon. To ale bylo vyřešeno tak, že se přímo ke každému trojúhelníku uložila i orientace normály, která je v souboru reprezentována jedním číslem *smer_normaly*. Lze ji tedy snadno zjistit a nemusí se počítat.

Pokud je nachystán tento aparát, stačí pak už jen v každé oblasti pro každý sektor určit, jak moc hustě mají být polygony v něm rozděleny. Nejdůležitější sektory tak mohou mít velice hustou síť vrcholů, zatímco sektory v pozadí nemusí být děleny vůbec.

V kapitole Osvětlení a stíny je popsán ještě jeden případ, kdy by bylo potřeba, aby byly různě hustě děleny i jednotlivé části stěn a zároveň šlo jednoduše určit, které části to mají být. Tento požadavek by bylo možné do stávajícího algoritmu jistým způsobem zakomponovat, ale dosažený výsledek a jeho klady by v tomto případě nepředčily jeho nevýhody. Problém byl nakonec vyřešen alternativním způsobem, jak je ve zmíněné kapitole popsáno. V případě větších a složitějších prací než je tato by však bylo vhodné přesnější řízení dělení polygonů umožnit. Jednou z možných cest by mohlo být použití a případná úprava některého ze známých algoritmů pro LOD sítě trojúhelníků.

5 Efekty s částicovými systémy

To, co scénám dodává realistický vzhled, co není jen osvětlená, vystínovaná místnost s pravoúhlými předměty a u čeho je nutné pro požadovaný výsledný efekt zachovat jistý stupeň realističnosti, jsou těžko simulovatelné přírodní jevy jako dým, oheň, sníh apod. Obecně jsou to všechny jevy nebo objekty, které nelze pro jejich velkou povrchovou členitost nebo způsob změny tvaru reprezentovat jako povrch.

Silnou, často používanou modelovací a zobrazovací technikou, která splňuje požadavky jak na real-time výpočet, tak i na zobrazování přírodních jevů, jsou částicové systémy (particle systems), které byly poprvé diskutovány v [7]. Jsou snadno implementovatelné a škála jejich využití je široká. Kromě již zmíněných jevů lze pomocí částicových systémů s výhodou generovat i hejna ryb či ptáků, padající sníh nebo déšť, trávu nebo dokonce celý model lesa, jak je popsáno v [8].

5.1 Co jsou to částicové systémy

Modelované jevy je někdy výhodnější nevytvářet jako jeden objekt, ale naopak jako množinu malých prvků (částic), jejichž vlastnosti jsou dány podle účelu částicového systému. Jsou to tedy systémy modelující jisté jevy mnoha malými částicemi s určitými vlastnostmi, které se v čase mění podle zadaných pravidel.

Mezi základní vlastnosti částic patří hlavně poloha, barva, rychlost, směr pohybu, případně zrychlení, tvar, hmotnost atd. Pokud je částicový systém dynamický, mění se některé tyto vlastnosti (např. směr či rychlost) v čase. Částice mohou v jistém čase a místě vznikat a po nějaké době zase mizet, mohou emitovat další částice, narážet do jiných částic nebo objektů a tak je dále ovlivňovat.

Jednotlivé částice mohou být reprezentovány různě, podle účelu částicového systému. Nejjednodušší, ale také nejméně využitelnou možností je částici reprezentovat jediným bodem. Mnohem častěji se však používá čtverec (billboard, impostor) s vhodně zvolenou texturou jako kapka deště, jiskra či sněhová vločka. Nic však nebrání tomu, aby byla částice reprezentována čímkoliv jiným, od koule až po libovolný geometrický útvar. Vždy záleží na jevu, který chceme částicovým systémem modelovat.

Klíčovou roli při práci s částicovými systémy hrají náhodná čísla. Většinu vlastností, které částice mají, lze charakterizovat jejich střední hodnotou a náhodným rozptylem. Pro zvolenou vlastnost x to názorně ukazuje následující rovnice:

$$x = s + rand * v \qquad (3)$$

kde s je střední hodnota hustoty pravděpodobnosti, v je rozptyl a $rand$ náhodné číslo. Veškeré parametry a nastavení systému provádí tvůrce systému nebo jsou vypočítány z jiných parametrů tak,

aby systém modeloval požadovaný jev. Globálně nelze vytvořit jeden obecný částicový systém, který by s minimem změn modeloval více jevů, protože se variabilita parametrů a jejich chování v čase může velice lišit podle účelu systému.

Jeden obrázek animace s částicovým systémem pak lze vytvořit podle následujících kroků (převzato z [9]):

1. Na základě parametrů každé existující částice se aktualizuje její poloha a ostatní atributy.
2. V celém systému se vytvoří nové částice. Ty vznikají buď v nějaké konkrétní oblasti, nebo mohou vznikat jako potomci jiných částic. Oblast, ve které částice vznikají, může být libovolná – mrak, ze kterého prší, děravá hadice, ze které stříká voda atp.
3. Každé nové částici se přiřadí její vlastnosti.
4. Částice, které překročily dobu svého života, nebo jiné hranice, zaniknou.
5. Systém se zobrazí.

Částice se však nepoužívají jen pro modelování obtížně tvarovatelných objektů, ale jsou také nástrojem pro zobrazování fyzikálních simulací. Metoda trasování částic (particle tracing) se používá v případech, kde je potřeba zobrazit složité fyzikální pole. Například pro zobrazení proudění ve větrném tunelu jsou do něj vypuštěny částice, které se v něm chovají podle stanovených pravidel, a lze tak snadno vizuálně zjistit potřebné vlastnosti modelu v tunelu.

Orientované částice jsou částice s vlastním lokálním souřadným systémem a jsou schopny se samy v prostoru orientovat podle zadaných pravidel. Mohou tak vytvořit i povrch, který pak lze zobrazit klasickými technikami (Phongovo stínování atd.).

Je nutné podotknout, že ačkoliv se částicové systémy používají pro real-time realistická zobrazení, mohou být stejně časově náročné jako fyzikální simulace. Jejich síla spočívá v tom, že se v každém kroku aktualizuje určitý počet částic, které dohromady tvoří vizuálně realistický celek. Pokud ale bude částic velké množství, bude jejich aktualizace trvat příliš dlouho a dojde ke zpomalení a ztrátě výkonu. Při návrhu systému je proto potřeba myslet na různé vazby a vztahy, které mohou ovlivnit rychlost systému. Příkladem budiž vztah velikosti textury částic a jejich počtu. Čím větší bude textura, tím méně částic bude potřeba a systém bude rychlejší, ale také bude větší riziko vzniku artefaktů. Čím více částic bude, tím může být textura menší a systém bude vypadat lépe, ale také bude pomalejší. Vždy je potřeba nalézt určitý kompromis uspokojující jak výkon, tak realističnost částicového systému.

5.2 Knihovna pro práci s částicovými systémy

Pro demonstraci částicových systémů byla využita knihovna volně dostupná na internetu [5]. Ta kromě samotného částicového systému poskytuje struktury vektoru a základní operace s nimi, sama se stará o načítání textur částic a poskytuje třídu pro práci s kamerou, protože ta je k částicovému systému úzce vázána. Podle směru pohledu kamery je totiž počítáno natočení impostorových částic (viz níže) a dále byla využita i pro pohyb ve scéně.

Kamera se ovládá voláním několika jednoduchých metod. Jedná se o posun kamery na určité místo, otočení o zadaný úhel kolem jedné ze tří os, posun oběma směry po ose pohledu a posun vpravo nebo vlevo v závislosti na směru pohledu. Doplněn byl i posun kamery ve vertikálním směru.

Částicový systém je v podstatě řídící jednotkou pro jednotlivé částice. K jeho základním parametrům patří umístění generujícího prostoru. Podle typu částicového systému to může být jeden bod, plocha nebo i celý objekt. Pro případ výbuchu stačí brát za generující prostor (emitor) právě jeden bod, pro déšť by to byl pravděpodobně kvádr, umístěný nad celou scénou. V knihovně je emitor určen svojí pozicí a rozptylem, tj. okolím, ve kterém budou částice vznikat.

Dalšími důležitými parametry jsou směr pohybu částic a jejich rozptyl. Pro klasický kruhový výbuch bude směr nulový, protože se výbuch nikam nepohybuje, ale jeho rozptyl bude velký do všech stran. Naopak pro déšť by byl směr zadán, nejspíše ve směru záporné osy y, s malým rozptylem.

Maximální a minimální rychlost určuje rozmezí, jak rychle se mohou částice daným směrem pohybovat. Pro ovládání rychlosti slouží ještě další dva parametry, které určují zpomalení a časový okamžik, od kterého má být zpomalení na částice uplatněno. Dále je určen směr zrychlení a rozmezí jeho velikosti, rozmezí rychlosti rotace a rozmezí zrychlení rotace.

Další parametry určují alfa kanál textur. To, že jiskra pohasíná nebo se rozpouští sněhová vločka, je simulováno právě postupným zvyšováním alfa hodnoty textury u částice. Je tedy potřeba nastavit rozmezí alfa hodnot pro částice v době vzniku a v době zániku (většinou maximální průhlednost). Opět pro lepší kontrolovatelnost průběhu je jedním parametrem nastavena doba, od kdy se má začít alfa hodnota měnit. Podobně se nastavují čtyři parametry (rozmezí při vzniku a zániku) pro velikost částice a její barvu. Poslední parametry týkající se jednotlivých částic udávají rozmezí délky jejich života.

Následující parametry se již týkají celého částicového systému. Důležitým údajem je jejich počet a počet aktuálně používaných částic, protože se pro ně musí alokovat potřebná paměť. Pokud se pro daný systém určí textura, která se má použít, automaticky se nastaví parametr indikující použití textury, jinak se nastaví použití bodu jako částice. Několik parametrů, starajících se o celkový průběh systému, zajišťuje to, jestli se mají částice obnovit ihned jakmile zaniknou, nebo se již neobnoví. Dále

určují, zda životní cyklus systému proběhne jednou, nebo se bude periodicky opakovat. Též lze nastavit dobu, ve které budou částice postupně vznikat, případně i varianci, takže se bude počet vznikajících částic v čase měnit.

Inicializace a aktualizace systému se děje tak, že částicový systém postupně volá příslušné metody všech jednotlivých částic, kterých se to týká. Nejsou aktualizovány zaniklé částice. Při inicializaci se všem částicím náhodně nastaví parametry z rozmezí zadaného v rodičovském systému a vypočítají se kroky jednotlivých parametrů, o které se budou vlastnosti měnit. Při aktualizaci je vlastnost změněna podle těchto kroků a uplynulého času. Všude se do výpočtů zahrnují náhodná čísla, takže výsledek pak nepůsobí strojově přesně, ale naopak napodobuje simulované přírodní jevy.

Významnou vlastností systému je to, že lze zapnout natáčení částic směrem ke kameře. Texturovaným plochám, které se automaticky natáčí na kameru, se říká impostory. Po zapnutí této volby jsou pak částice systému neustále otočeny na kameru, i když se pozorovatel pohybuje, takže se nemůže stát, že by obejitím systému pozorovatel místo částic viděl jen jejich boky, což by vedlo k degradaci systému. Lze zapnout natáčení ve všech směrech nebo jen ve směru horizontálním.

5.3 Výbuchy

Výbuchy jsou velice složité jevy a nejedná se pouze o ohnivý efekt, jak by se mohlo na první pohled zdát. Výbuch modelovaný pouze ohněm nebude nikdy působit věrně. Je to proces spalování a při tom vždy vzniká dým, který je nutné také modelovat. Navíc se v případě explozí většinou jedná o destruktivní proces, při němž jsou působícími silami do prostoru vymrštěny různé hmotné objekty, ať už se jedná o kusy nějakých předmětů, strojů nebo jen kusy zeminy či skály. I s tím je nutné pro věrnou simulaci počítat. V případě intenzivních výbuchů by bylo možné simulovat i světelný záblesk. Pro účely této práce je tento jev zanedbán, protože není pro realističnost kritický a také ne vždy je pozorovatelný ve skutečnosti.

Na tomto místě je potřeba se zamyslet nad tím, jak simulovat kouř. Existuje totiž více způsobů. V přírodě dým vzniká tak, že chladne hořící substance a po určité době, kdy je spotřebováno palivo, se promění na pohaslé částice - kouř. V částicových systémech by se dal tento proces napodobit tak, že v době zániku ohnivé částice by na jejím místě vznikla částice kouře. To je sice vhodný způsob, nicméně použitá knihovna není tak propracovaná, aby to umožnila, a byl by potřeba větší zásah.

Kouř je proto modelován jako samostatný částicový systém., což přináší i svoje výhody. Lze jej nastavit nezávisle na systému ohně a dosáhnout tak lepších výsledků a hlavně je možno použít jiný, menší počet částic, což zlepší výkon.

Výbuch je tedy složen ze třech samostatných částicových systémů ohně, kouře a úlomků, rozmetaných výbuchem do okolí. Dále jsou popsány jednotlivé částicové systémy tvořící základní kulový výbuch, které byly nastaveny na základě popisu v [10].

5.3.1 Oheň

Nejdůležitější částí každého výbuchu je ohnivá koule. Proto také tento částicový systém obsahuje největší množství částic. Při jeho inicializaci se všechny částice vytvoří na jednom místě a jsou velkou rychlostí vymrštěny do všech stran, přičemž však také rychle zanikají. Jako emitor je zvolen jeden bod, případně s malou odchylkou do všech stran. Je nastavena velká rychlost rozptylu částic, přičemž směr je nulový, protože se výbuch jako celek nepohybuje a život částic systému je nastaven na krátkou dobu v rozmezí do jedné sekundy. Navíc, čím jsou částice starší, tím se jejich rychlost snižuje, což simuluje ztrátu energie výbuchu v čase. Na rychlost mají vliv i další síly. V tomto modelu bylo použito gravitační zrychlení, které působí stejně na částice všech tří systémů. Stejně tak život těchto systémů proběhne jen jednou, takže se výbuch neopakuje pořád dokola.

Na začátku života jsou částice ohně obarveny na červeno a jsou neprůhledné, v době zániku mají bílo-šedou barvu a jsou téměř průhledné. Barva těchto částic je navíc upravena barvou textury, která je uprostřed bílá a směrem ke krajům přechází přes žlutou a oranžovou do červené. Částice tedy nejsou při vzniku celé červené, ale mají mnohem sytější a jasnější barvy než v době zániku. Velikost částic byla zvolena tak, aby bylo dosaženo požadované velikosti výbuchu a aby při daném počtu částic nevznikaly na okrajích artefakty. Částice se v čase zvětšují, což má za následek, že výbuch vypadá jako spojitá ohnivá koule, ne jako jednotlivé, odletující plameny. Nakonec je částicím ohně nastavena náhodná rotace.

5.3.2 Kouř

I když samotná ohnivá koule vypadá pěkně, teprve kouř jí dodá na přesvědčivosti. Stejně jako ohnivá koule vzniká na jednom místě a je vymrštěn do všech stran. Částic použitých pro kouř stačí méně, mohou být větší a textura vypadá víc jako mlžný opar než jako část plazmy. Podstatné je to, že kouř je vidět déle než oheň, tudíž má větší životnost. Zatímco oheň již dávno pohasl, kouř se teprve rozplývá. Dále jeho barva se mění od úplně černé po světle šedou. Na počátku spalování jde o hustý těžký kouř, který se postupně rozplývá a mizí a než zanikne, zbývají z něj už jen lehké rozptýlené částečky. To, že je kouř na začátku úplně černý, je výhodné také proto, že nad černým dýmem ohnivá koule lépe vynikne na jakémkoliv pozadí výbuchu. Bez kouře je ohnivá koule na světlém pozadí mdlá, s černým dýmem je však ohnisko výbuchu krásně viditelné. Rotace kouřových částic je pomalejší než u ohně.

5.3.3 Úlomky

Poslední částí je systém úlomků a střepin odletujících od výbuchu. Základní chování je stejné jako v předchozích dvou případech. Rozdíl je zejména v rychlosti, kdy jsou úlomky mnohem rychlejší a dolétnou dále než oheň, a také se nemění jejich velikost v čase. Úlomky jsou pevná tělesa, takže zůstávají stejně velká. V závislosti na použité textuře je možné docílit jejich různých druhů. Mohou to být kusy hlíny, dřeva, železa, střepy skla, třísky či jiskry. Důležité je také to, že tyto částice se většinou neztrácí, ale zůstanou pak ležet na zemi. Na takovou simulaci by však bylo potřeba model rozšířit, proto byly jako úlomky zvoleny jiskry, které nakonec také pohasnou. Výhodu to má i v tom, že jiskry mají pořád stejný tvar. V případě použití pevných částic by bylo vhodné použít i několik textur, aby nevypadaly všechny stejně. Jejich jednotný tvar by působil rušivě a kazil dojem celého výbuchu. Tomu se dá trochu vyhnout tím, že se částicím nastaví vysoká rychlost rotace, takže není jejich tvar tak dobře patrný, ale ani tak není výsledek úplně ideální. I počet úlomků je mnohem menší než počet částic ohně či kouře. Zatímco v prvních dvou případech je potřeba i několik set částic, pro úlomky stačí pár desítek.

5.3.4 Parametrizace dalších typů výbuchů

I když výsledný výbuch vypadá pěkně, což bylo cílem, má přece dvě zásadní nevýhody, které by bylo vhodné odstranit. První je, že exploze vypadá typově pořád stejně. I když plameny šlehají vlivem náhodných čísel pokaždé jinak a kouř se také rozplývá pokaždé jinak, je výbuch jako celek stejný. Jeho velikost, umístění, barva, rychlost zániku, to vše je pořád stejné a nemění se. V realitě se však exploze od sebe liší mnohem víc, jak tvarem, intenzitou barev, tak i směrem atd.

Tento problém lze vyřešit tak, že se určí několik parametrů, kterými se před spuštěním výbuch nastaví. Tyto parametry ovlivňují podle určitých kritérií všechny tři částicové systémy, ze kterých je výbuch vytvořen. Když se například zadá požadavek na změnu výbuchu, změní se velikost částic ohně i kouře, ale nemůže se změnit velikost úlomků (nebo alespoň ne ve stejném měřítku). Úlomky jsou pevné částice a jejich zvětšením by byl v podstatě na výbuch aplikován zoom efekt, což není požadovaný výsledek.

I na vzájemnou provázanost dalších vlastností systémů je potřeba brát zřetel, jinak by se mohlo stát, že kouř celý zmizí v ohni a podobně. Proto je parametry, které ovlivňují všechny tři částicové systémy a mění celý výbuch, potřeba pečlivě zakomponovat a dbát na to, s jakou váhou ovlivňují který systém.

Druhou zásadní nevýhodou je, že jeden objekt výbuchu vytvoří jen jednu explozi, kterou je možné vidět. Nelze jich spustit několik současně. Samozřejmě lze vytvořit více objektů a není potom s vytvořením několika současných výbuchů problém. Přesto, že na ně použijeme různou parametrizaci, jak bylo popsáno v odstavcích výše, budou působit podobně. Proto, když už je potřeba

současně zobrazit více výbuchů, je vhodné vytvořit zcela jiné typy. Není potřeba vytvářet znova celou třídu, ale stačí zdědit potomka a technikou virtuálních metod jen přepsat tu, která nastavuje parametry výbuchu, popřípadě provést některé další drobné úpravy. Zadarmo tak získáme i parametrizaci výbuchu vytvořenou pro hlavní explozi.

Tímto způsobem byl pro demonstraci vytvořen směrový výbuch nebo přesněji vyšlehnutí plamene do výšky. Bylo zděděn ze základního výbuchu a změněny jen některé parametry. Především mu byl nastaven směr pohybu podle osy y a zrušen rozptyl do všech stran. Ponechán byl jen rozptyl v malém kuželovitém tvaru ve směru pohybu. Barva byla nastavena více do červené, takže výsledek připomíná spíš plamen než výbuch. Kouř a jiskry jsou více zvýrazněny než v případě výbuchu.

Takovýmto způsobem by bylo možné vytvořit nepřeberné množství výbuchů, plamenů a dalších ohnivých efektů. Pro demonstraci však tyto dva efekty budou postačovat.

5.4 Plameny a oheň pochodně

Pokud již máme hotový výbuch, snadno z něj můžeme několika málo úpravami vytvořit i jiné ohnivé efekty. Ty jsou v této práci reprezentovány ohněm pochodní a plameny, šlehajícími z hořícího barelu. Oba tyto částicové systémy mají stejné nastavení, jen se liší parametrizací, která byla popsána u výbuchů. V podstatě se jedná jen o jinou velikost částic a emitoru. Ten je na rozdíl od výbuchů plošný a tvoří ho v případě plamenů pochodně elipsa, umístěná k přední části modelu hořící části pochodně, a v případě plamenů barelu kruh, umístěný kousek pod víko barelu.

Zásadním rozdílem, který je mezi plameny a výbuchem, je chování jednotlivých částic po dokončení svého života. V případě výbuchů tyto částice zůstanou mrtvé a již se nezobrazí. V případě plamenů se však částice po zaniknutí ihned obnoví, jsou znovu inicializovány a vypuštěny do koloběhu. Oheň tedy hoří neustále. Plameny mají v čase přibližně konstantní velikost, nezanikají jako v případě výbuchu a jejich náhodné chování vytváří realistický obraz hoření.

V částicovém systému, který je takto kontinuální a ne jednorázový, je možné nastavovat parametr, který udává počet vypuštěných částic za sekundu. Uveďme si dva příklady. Částicový systém má 100 částic, za sekundu jich emituje deset a život jedné částice má přibližně deset sekund. Takový částicový systém se bude jevit, jakoby neustále emitoval nové částice a je to příklad právě popisovaných plamenů. V druhém příkladu je má částicový systém opět 100 částic, za sekundu stihne vyslat všech sto částic a jejich život je dlouhý dvě sekundy. Tento částicový systém bude vypadat, jako by periodicky po dvou sekundách vysílal velké množství částic, přičemž mezi touto dobou nic neemituje. Dal by se tak modelovat například kouř z komínů parních lokomotiv v kreslených pohádkách, kde komíny vypouští jednotlivé obláčky kouře a mašinka při tom spokojeně

bafá. Při bližším zamyšlení přijde jistě každý na to, že lze tímto parametrem ovládat i hustotu výsledného částicového systému, tedy kolik částic bude právě zobrazováno.

S dýmem je to v případě těchto plamenných částicových efektů stejné. Řídí se stejnými pravidly jako v případě výbuchů a jeho částice jsou emitovány průběžně stejně jako částice ohně. Částicový systém úlomků v tomto případě není potřeba, pro efekt hoření stačí oheň a kouř. Systém úlomků by se dal využít v některých specifických případech například pro jiskry.

5.5 Dým

Dým použitý v demu je ze zmíněných efektů nejjednodušší. Obsahuje pouze částicový systém kouře, tedy ani oheň, ani úlomky. Jeho parametry se dají nastavovat stejně jako v předchozích případech a stejně jako u hoření se jedná o kontinuální systém, takže se jeho částice neustále obnovují. Liší se snad pouze velikostí emitoru a barvou částic.

V demu je možné vidět několik typů kouře. Prvním z nich je mírný kouř. Je spíše šedý a obsahuje málo částic, které mají dlouhou životnost. Ve výsledku tedy tvoří řídký vysoký dým. Druhý je naopak hustý černý dým, valící se z vybuchlého vznášedla, evokující dojem hořícího oleje. Toho je docíleno velkým počtem téměř černých částic s krátkým životem. Mohutnosti dýmu napomáhá i plošně velký emitor.

Třetím typem dýmu je kouř, valící se z vybuchlého barelu. Při souboji jeden z vystřelených laserů zasáhne barel, který vybuchne. Jakmile zanikne částicový systém reprezentující tento směrový výbuch, objeví se místo něj výše popsaný plamenný systém a systém kouře. Dohromady tak tvoří efekt hořícího barelu. Aby byl efekt ještě zajímavější, je dýmu v době zániku nastavena zelená barva, čímž je docíleno dojmu, jako by v barelu hořela nějaká chemikálie.

Dalším efektem vytvořeným pomocí tohoto částicového systému jsou výpary. V jedné z místností je na zemi kolem zničeného barelu rozlita chemikálie, z které stoupají obláčky jedovaté páry. Jediným podstatným rozdílem oproti dýmům je jiná barva, v tomto případě žlutá, která se po dobu života částic nemění. Dým je velice řídký, takže vypadá opravdu jako výpary, ne jako spojitý dým. Tomu napomáhá i fakt, že emitor je kruh o velkém poloměru, takže výpary stoupají z celého povrchu rozlité chemikálie, ne jen z jednoho bodu.

6 Externí modely

Důležitou složkou této práce a hlavním prvkem, ke kterému se váže dění v demu, jsou létající stroje. Jedná se konkrétně o dva typy, raketu a vznášedlo. Raketa má obyčejný šípovitý tvar a na konci těla má diagonálně připojené čtyři tryskové motory, dva menší nahoře a velké dva hlavní dole. Kokpit je umístěn na horní části těla rakety. Vznášedlo je podobné tanku s tím rozdílem, že místo podvozku má tělo tvaru elipsoidu s několika různými výčnělky. Věž i dělo mají přibližně klasický tvar.

Kromě těchto dvou stěžejních modelů bylo vytvořeno i několik dalších menších modelů. Jsou to zejména modely různých druhů světel - zářivky, lampy atd. Kromě toho byla vymodelována i pochodeň. Tato kapitola bude spíše než na samotné modelování zaměřena na popis různých drobností, se kterými je nutné při práci s modely počítat.

6.1 Vytvoření a export modelů

Všechny modely byly vytvořeny v programu Blender různými metodami modelování. Nejčastěji se jednalo o práci s mash objekty. Některé ze základních objektů jako koule či krychle byly pomocí subsurfingu rozděleny na více vrcholů a dále dotvarovány do požadovaného vzhledu. V případě světel byly použity i křivky a následná práce s nimi. Základní tvar pochodně vnikl rotováním křivky kolem svislé osy. Více informací o metodách modelování, popis konkrétních příkazů a tutoriály k Blenderu lze nalézt například v [3, 4, 15].

Oblíbeným a často používaným formátem pro uložení a přenos modelů je formát VRML (Virtual Reality Modeling Language). Jak již sám název napovídá, jedná se o celý obsáhlý jazyk, kterým je možné popisovat 3D světy. Byl určen hlavně pro WWW prostředí, ale ujal se i mimo ně, právě pro popis modelů a scén. Obsahuje mnoho nastavení a parametrů. Kromě popisu geometrie a transformací podporuje i osvětlení, texturování, mlhu a zvuky, hypertextové odkazy či časování. Poskytuje dokonce i dynamiku objektů. Formát VRML textový, takže jej lze jednoduše prohlížet a editovat a navíc je platformě nezávislý. Pro všechny platformy existuje celá řada free prohlížečů a editorů, navíc do něj lze exportovat z mnoha modelovacích programů (viz Blender). Více o formátu VRML lze najít na [16].

Právě díky rozšířenosti a podpoře, byl VRML zvolen jako vhodný formát pro popis modelů v této práci. Po dokončení práce na jednotlivých modelech tedy stačilo v Blenderu vyvolat exportní příkaz a model jednoduše uložit ve VRML.

Po prozkoumání samotného VRML souboru, ale nastaly první komplikace. Blender exportoval pouze souřadnice vrcholů mash objektů, neexportoval však textury a dokonce ani texturovací souřadnice. To byl ovšem zásadní problém, protože bez textur by modely nikdy nemohly vypadat

pěkně. Naštěstí řešení situace nebylo příliš složité. V tutoriálech k Blenderu je možné se dočíst, že Blender exportuje pouze texturovací souřadnice UV textur, přičemž samotné textury neexportuje.

Prvním krokem tedy bylo se naučit UV texturování a použít tuto techniku na modely. UV texturování spočívá v tom, že nezadáváme pouze čtyři souřadnice rohů textury, popřípadě nějaké snadno odvoditelné souřadnice, ale popis nanesení textury je mnohem komplexnější. Textura například může obsahovat obrázek obličeje a pomocí UV texturování tento obličej potom namapujeme na model hlavy tak, že určíme, které body textury odpovídají kterým vrcholům modelu. V UV textuře tedy může být uložen i velmi komplikovaný prostorový objekt.

Jak se ukázalo později, jednoduché texturování by stejně k vytvoření dokonalých modelů, vzhledem k jejich složitému tvaru, nestačilo. Protože modely nemají ani jednoduché rovinné plochy ani pravidelné kulovité tvary, nikdy by se nepodařilo texturu namapovat tak přesně, jak bylo třeba. Naproti tomu je mnohem lehčí si vytvořit UV texturu, například pláště rakety, a tu potom přesně nanést na model rakety. Hlavním problémem je zde samotné vytvoření textury, což předpokládá určitou zručnost a trpělivost. To ale platí při vytváření jakékoliv textury, nejen UV.

Po exportování modelů s UV texturou již Blender exportoval texturovací souřadnice správně. Stále ale neexportoval názvy či jiné identifikátory textur, podle kterých by po načtení VRML do programu bylo možné nanést požadované textury. Tento problém se v Blenderu nakonec nepodařilo vůbec vyřešit. Bylo by nejspíše třeba jiný exportní skript VRML, ale žádný z vyzkoušených to neuměl a většina z nich ani nefungovala. Proto byl pro export použit původní skript dodaný s Blenderem.

Řešením situace bylo editovat exportované VRML ještě před načtením do programu a doplnit jména textur. Bylo by možné to provést ručně, lepší však bylo zvolit nějaký jiný program na práci s VRML. Pro tento úkol se osvědčilo Flux Studio [18], free program na vytváření a editaci modelů, které také umožňovalo export a import VRML formátu. Tento program jednoduše rozpoznal, které vrcholy patří které textuře, takže stačilo pouze doplnit jména textur a po exportování již byl VRML kompletní, tak jak bylo třeba pro kompletní uložení modelů.

Vzhledem k jednoduchosti a názornosti práce ve Flux Studiu v něm bylo provedeno ještě několik dalších úprav modelů. Některé části modelů měly oblé tvary, jiné byly plošné s ostrými hranami. Kdybychom chtěli, aby se oblé tvary zobrazovaly opravdu jako oblé, musely by být vytvořeny z mnoha set vrcholů, což by vedlo na obrovské množství uložených údajů a vykreslování by bylo extrémně pomalé. Pro tuto práci tedy takový způsob nebyl nepoužitelný. Vhodnějším metoda je, modelovat i oblé tvary méně vrcholy. Je potom sice vidět polygonová podstata těchto částí a jsou nepěkně kostrbaté, ale pouze do doby, než zapneme vyhlazené stínování (Gouraudovo či Phongovo), které tento jev potlačí. Modely pak vypadají jako hladké a při tom jsou tvořeny jen několika málo vrcholy. Ovšem v případě, kdy model obsahuje i oblé tvary i tvary s ostrými přechody a máme

zapnuté vyhlazené stínování, jsou vyhlazeny i tyto ostré přechody a nejsou vidět. Pokud toto osvětlení vypneme, je opět vidět hranatost kulatých tvarů. Tento problém vyřešíme uložením oblých a hranatých částí modelu samostatně, přičemž klidně mohou mít přiřazenu jednu UV texturu. Flux Studio potom tyto části rozpozná a stačí jednoduše každé této části nastavit jiné stínování, kulatým částem hladké stínování a částem s ostrými přechody mezi plochami konstantní stínování. Kromě toho bylo ve Flux Studiu doladěno i materiálové nastavení modelů.

Flux Studio také poskytovalo k editaci stromovou strukturu VRML souboru a bylo tedy možné jednoduše a přehledně smazat ty části VRML, které nebyly pro uložení modelů potřeba a zbytečně zvětšovaly jeho obsah. To se týkalo zejména nastavení kamery a osvětlení. Tyto aspekty jsou řešeny samostatně v rámci programu a ve VRML by tedy byly uložené naprosto zbytečně.

Také je dobré se zamyslet nad tím, které objekty mají být uloženy v jednom VRML souboru. Na první pohled by se zdálo, že jeden model patří do jednoho VRML souboru. To ovšem nemusí být pravda. Například vznášedlo sestává ze tří částí, které se mohou pohybovat nezávisle na sobě. Základem je tělo vznášedla, na které je napojená věž. Ta se na tělu může otáčet kolem svislé osy. K věži déle náleží dělo, které se otáčí zároveň s věží, ale může mít různý náklon. Kdyby byl celý model vznášedla uložen v jednom VRML souboru, nemuselo by pak být jednoduché transformovat jednotlivé části zvlášť. Velice by potom záleželo na poskytnutých nástrojích knihovny, která by VRML načítala. Vhodnější je proto na toto myslet dopředu a uložit si každou pohyblivou část modelu zvlášť. Samozřejmě modely, které tvoří jeden celek a které se pohybují jako celek, budou většinou uloženy v jednom souboru.

Nakonec ještě jeden postřeh k uložení modelu do VRML souboru. Když v programu vytváříme model, máme nějaké základní objekty, například mashe či křivky. Na ty aplikujeme různé transformace, rotace, posun, změnu měřítka atd., čímž měníme vzhled výsledného modelu. Když se potom model exportuje do VRML, uloží se do něj základní objekty a všechny transformace, které s ním byly provedeny. Při vykreslování se tak vždy na vykreslovaný objekt nejdříve aplikují všechny transformace a teprve potom je správně zobrazen. Je tedy vidět, že i vykreslení jednoduchého modelu, na který bylo aplikováno hodně transformací, může trvat poměrně dlouho. Zvláště v případě změny měřítka může být toto zpomalení významné.

Tomuto nepříjemnému jevu se však dá snadno zamezit tím, že před uložením modelu do VRML necháme transformace na objekty skutečně aplikovat. V Blenderu to lze provést klávesovou zkratkou Ctrl+A. Všechny řídící body a vrcholy se přepočítají přímo v modelovacím programu a do VRML souboru se už uloží přímo konkrétní souřadnice bez transformací. Při vykreslování se pak berou přímo uložené vrcholy a není nutné je ještě všechny transformovat.

6.2 Načtení modelů do programu

Pro načítání VRML modelů do aplikace byla zvolena free knihovna dostupná z [17]. Ta podporuje základní vlastnosti VRML jako geometrii a vzhled, tedy materiály a textury, a některé další aspekty. Není pro účely této aplikace příliš složitá a rozsáhlá a pokrývá přesně to, co je od VRML požadováno.

Tuto knihovnu však bylo třeba rozšířit a upravit, aby fungovala přesně podle požadavků. V prvé řadě si, jako většina jiných knihoven, sama zajišťovala načítání a vytváření textur. To ale bylo zcela zbytečné a neodpovídalo to navrženému konceptu aplikace, která si načítání textur zajišťuje podle potřeby sama. Bylo tedy nutné odpojit stávající systému práce s texturami a napojit ho na vlastní. V podstatě se jednalo o to, že se do objektu udržujícího VRML objekt navíc předal ukazatel na texturovací objekt aplikace a identifikační číslo textury, která se má na model nanést.

Texturovací objekt obsahuje všechny potřebné informace a řízení pro práci s texturami, takže se tak vše zpřístupnilo i modelům. Identifikační číslo textury bylo potřeba k určení konkrétní textury pro mapování, protože texturovací objekt nemůže sám o sobě vědět, která textura patří kterému modelu.

Oba modely strojů jsou natolik komplikované, že jejich vzhled nebylo možné nakreslit do jedné textury, a proto je na každém modelu aplikováno textur více. K tomu bylo potřeba knihovnu upravit, protože zpočátku umožňovala mapování pouze jedné textury. Byl vytvořen krátký algoritmus, pomocí kterého si knihovna sama pro sebe uloží čísla textur v tom pořadí, jak jsou uloženy ve VRML souboru. Identifikační číslo textury předávané do knihovny potom udává první texturu ve VRML souboru a je snadné podle vnitřně uložených čísel textur určit skutečnou texturu pro namapování na model. Jedinou podmínkou je, aby byly textury v aplikaci uloženy v takovém pořadí, v jakém jsou postupně použity ve VRML souboru.

Pro úplnou dokonalost práce knihovny byla ještě doplněna obsluha pro spekulární a shininess složku materiálu, které nebyly úplně implementovány.

7 Laserové paprsky

Oslnivé efekty jsou důležitou složkou všech her, animovaných filmů i různých grafických upoutávek. Lasery patří k těm efektů, které se objevují pořád znovu a znovu a to nejen v podobě laserových střel vesmírných lodí, příručních zbraní z budoucnosti či různých magických střel. Neprůhledné nebo vhodně upravené paprsky lze použít například i jako stavební prvky budov. V této práci jsou paprsky použity právě pro laserové střely bojujících strojů.

Existuje několik metod, jak paprsky vytvořit. Každá z nich má své klady a zápory. Nejjednodušší způsob využívá několik spritů orientovaných na kameru, vykreslených podél paprsku, čímž ho mají vytvořit. To ale vyžaduje mnoho spritů na vytvoření malého počtu paprsků a dochází k plýtvání výkonem. Navíc tento způsob trpí mnoha vizuálními nedostatky.

Výkonově mnohem úspornější způsob je využití jednoho otexturovaného billboardu – polygonu, který se neustále natáčí na kameru, takže nikdy není vidět z boku. Toto je celkem vhodná volba, která v mnoha případech poskytuje uspokojivé výsledky. V perspektivním promítání však může být nápadná polygonová podstata paprsku.

Spojením těchto dvou metod však lze dosáhnout jak dobré výkonnosti, tak pěkného vzhledu ve většině případů. Jeden paprsek sestává z pouhých čtyř trojúhelníků. Dva trojúhelníky tvoří billboardy reprezentující konce paprsku a dva trojúhelníky tvoří jeho tělo. Tato metoda, jež bude níže popsána, byla prezentována v [14].

7.1 Matematický základ

Matice billboardů pro koncové body paprsku jsou určeny modelační maticí a pozicí každého z koncových bodů. Běžná matice bilboardu pro sprity orientované ke kameře využívá modelační matici, jen s jiným umístěním. Ale pro paprsek by měla být matice billboardů orientována ve směru paprsku na obrazovce. Dopředný směr proto zůstane stejný jako v modelační matici, ale pravý vektor a vektor udávající směr vzhůru se změní.

Nejdříve si z koncových bodů paprsku určíme vektor paprsku, což je jeho směrový vektor o skutečné délce paprsku.

$$\boldsymbol{B} = \boldsymbol{B}_1 - \boldsymbol{B}_2 \qquad (4)$$

Je vhodné pro výpočet **B** opravdu použít tento vzorec, protože pokud za **B** dosadíme směrový vektor paprsku vynásobený délkou paprsku, ořízneme si ho právě o podstatné ukončení.

Dále si vypočítáme takzvaný „eye vektor“. Je to vektor směřující z pozice kamery do jednoho z koncových bodů paprsku.

$$\boldsymbol{E} = [M_{03} \quad M_{13} \quad M_{23}]^T - \boldsymbol{B_1} \tag{5}$$

Dále je nutné zjistit vektorový součin eye vektoru a vektoru paprsku, čímž získáme vektor kolmý k paprsku v rovině obrazovky.

$$\boldsymbol{P} = \boldsymbol{B} \times \boldsymbol{E} \tag{6}$$

Nakonec vypočítáme normalizovaný vektorový součin kolmého vektoru **P** a dopředného vektoru z modelační matice **F**, což nám dá pro billboard vektor směřující vzhůru.

$$\boldsymbol{F} = [M_{00} \quad M_{10} \quad M_{20}]^T \tag{7}$$

$$\boldsymbol{U} = \boldsymbol{F} \times \boldsymbol{P} / \|\boldsymbol{F} \times \boldsymbol{P}\| \tag{8}$$

Pravý vektor pro billboard opět spočítáme jako vektorový součin.

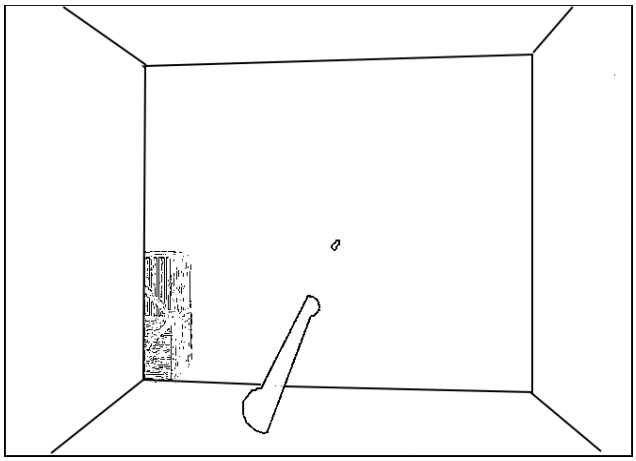
$$\boldsymbol{R} = \boldsymbol{F} \times \boldsymbol{U} \tag{9}$$

Nyní už můžeme sestavit matice pro billboardy představující počáteční a koncový bod paprsku. Shodně obsahují vektory určující orientaci billboardu, liší se v umístění koncových bodů.

$$\boldsymbol{M_1} = \begin{bmatrix} R_x & U_x & F_x & B_{1x} \\ R_y & U_y & F_y & B_{1y} \\ R_z & U_z & F_z & B_{1z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$\boldsymbol{M_2} = \begin{bmatrix} R_x & U_x & F_x & B_{2x} \\ R_y & U_y & F_y & B_{2y} \\ R_z & U_z & F_z & B_{2z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

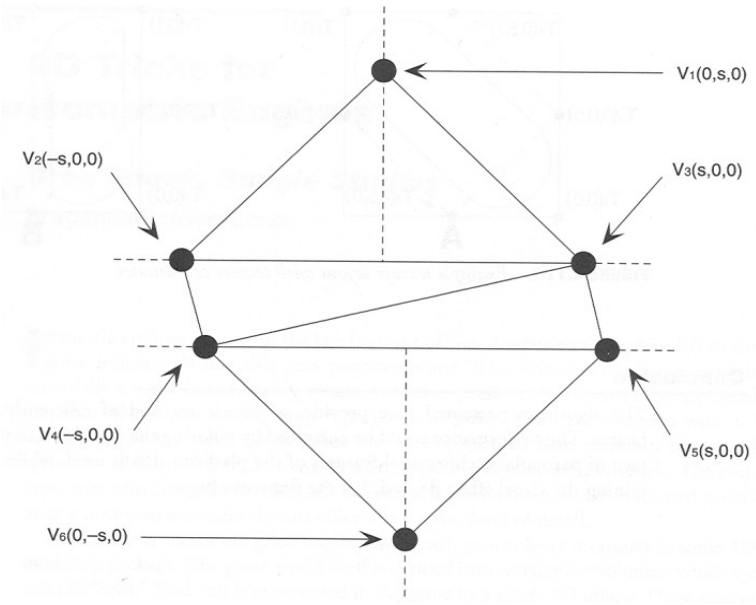
Důležité je také podotknout, z jaké modelovací matice se výše uvedený výpočet provádí. Pokud totiž vektory **E** a **F** získáme z modelovací matice, kterou používáme pro zobrazování scény, nedostaneme správně orientované paprsky, ale jejich ukončení bude na kameru stále kolmé. Tato chyba je tím nápadnější, čím víc je pohledový vektor rovnoběžný s vektorem paprsku. Chyba je znázorněna na obrázku 1. Pro správný výpočet je třeba vytvořit modelovací matici posunutím a otočením kolem os v kladném smyslu. Tedy například posunovat o ten vektor, se kterým pracujeme, ne o jeho zápornou hodnotu, kterou používáme pro rotaci scény.



Obr 1: Chybné zobrazení paprsku. Jsou patrné jeho konce natočené na kameru.

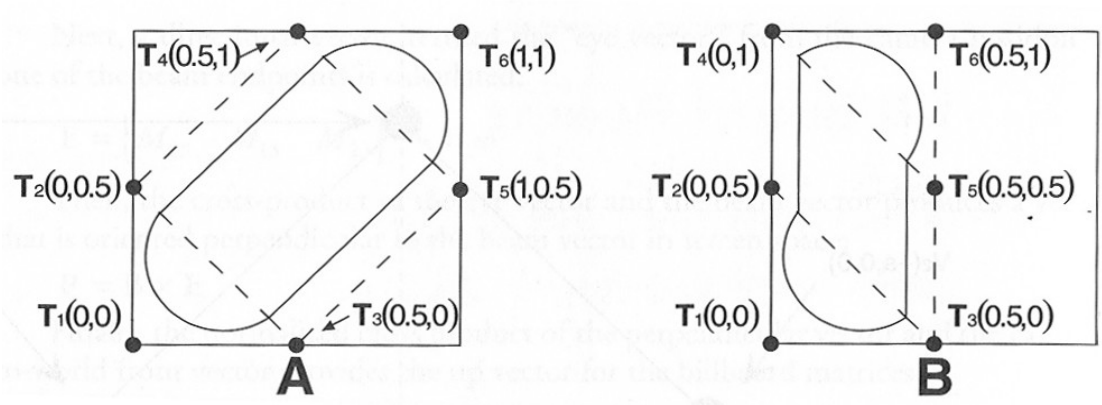
7.2 Vrcholy a mapování textury

Obrázek 2 znázorňuje rozložení trojúhelníků při vykreslování paprsku. Vrcholy $\mathbf{V}_1$, $\mathbf{V}_2$ a $\mathbf{V}_3$ jsou transformovány maticí $\mathbf{M}_1$. Vrcholy $\mathbf{V}_4$, $\mathbf{V}_5$ a $\mathbf{V}_6$ jsou transformovány maticí $\mathbf{M}_2$ a tudíž jsou neustále natočeny na kameru. Díky tomu se zdá, že je na kameru natočen celý paprsek. Vrcholy $\mathbf{V}_2$ až $\mathbf{V}_5$ tedy tvoří tělo paprsku, *s* je poloměr paprsku. Pokud to architektura dovoluje, je vhodné do výpočetního řetězce neposílat jednotlivé trojúhelníky zvlášť, ale najednou v pořadí 1 až 6 podle indexů. V OpenGL je to možné provést nastavením parametru zpracování vrcholů *GL_TRIANGLE_STRIP*. Je důležité, aby potom měly všechny vrcholy stejné parametry, zejména při mapování textury.



Obr 2: Souřadnice vrcholů trojúhelníků pro paprsek

Obrázek 3 znázorňuje dvě možnosti, jak texturu mapovat. Texturovací souřadnice $\mathbf{T}_1$ až $\mathbf{T}_6$ náleží odpovídajícím vrcholům $\mathbf{V}_1$ až $\mathbf{V}_6$. Při výběru, které rozvržení textury použít mějme na paměti následující rozdíly. První rozložení lépe odpovídá požadovanému vzhledu paprsku a tudíž méně podléhá grafickým chybám. Zabírá celou texturu a není už možné do ní uložit jiný obrázek. Druhá metoda je na místo textury úspornější a její druhou volnou část lze využít například pro jiný tvar textury paprsku. Občas však může způsobovat nechtěné chyby ve vykreslování. Protože paprsky v demu létají rychle a poměrně daleko od pozorovatele, nejsou žádné nepřesnosti vidět a byl proto zvolen tento druhý přístup. Volná polovina textury byla využita pro další tvar laserového paprsku, který je možné v demu vidět.



Obr 3: Rozložení textury v obrázku a její souřadnice

7.3 Animace laserů a navázání na výbuchy

Po vytvoření samostatných laserových paprsků, bylo třeba je ještě zakomponovat do dema a navázat na bojující stroje. V první řadě bylo třeba určit čas, kdy bude laserový paprsek vystřelen, a hned nato bod, tedy hlaveň děla, ze které budou laserové střely létat. Dále je nutné zadat směr, kterým poletí, jak daleko poletí, popřípadě co se s nimi po dosažení cíle stane. Kromě toho musí mít každý paprsek určité vlastnosti, jako barvu, texturu, velikost, rychlost apod.

Pro lasery tedy byla vytvořena samostatná třída, jejíž objekt si spravuje a řídí objekt modelů. Časování vystřelení paprsků bylo vyřešeno tak, že jsou všechny výstřely pevně naplánovány na určitou dobu, která se počítá od daného kontrolního bodu v trajektorii pohybu avatara v podzemí. Aby se nemusel čas zbytečně odpočítávat od spuštění programu, začne se odpočítávat až těsně před tím, než začne souboj. Konkrétně až avatar dosáhne určitého kontrolního bodu ve své trajektorii pohybu. O trajektorii pohybu a jejích kontrolních bodech je podán podrobný popis v kapitole Pohyb. Po

uplynutí daného času, v době výstřelu paprsku, je objektu laseru oznámeno, že má vytvořit nový paprsek, a podle pořadí je určeno, který ze strojů střílí. O zbytek se již stará objekt laseru sám.

Nejdříve objekt paprsku přiřadí specifické vlastnosti podle toho, jestli je tento paprsek vystřelen raketou nebo vznášedlem. Každý za strojů má totiž jiný typ laserových střel. Raketa používá kratší a rychlejší červené lasery se zaoblenými konci. Vznášedlo naproti tomu vystřeluje pomalejší a delší žluté lasery, jejichž textura je spíše šípovitá. Každému paprsku je také určena počáteční pozice a bod, ke kterému letí. Počáteční pozice může být zadána buď přímo souřadnicemi, nebo odvozena z aktuální pozice střílejícího stroje. V případě rakety to jsou hlavně kanonů v gondole pod trupem a počáteční bod je tedy odvozen z pozice rakety pouhým posunutím. V případě druhého stroje je výpočet poněkud složitější. Hlaveň děla je vůči pozici vznášedla posunuta, otáčí se s věží okolo osy y a sama o sobě ještě nahoru a dolů.

Taktéž bod, do kterého paprsek letí, je zadán buď konkrétními souřadnicemi, nebo aktuální pozicí některého ze stroje, takže není třeba znát přesné souřadnice v daném čase. Tento a počáteční bod udávají směr letu paprsku. Délka dráhy může být odvozena buď ze vzdálenosti těchto dvou bodů, nebo může být zadána konkrétní hodnotou, což může být v některých případech výhodnější. Nakonec se určí, zda má paprsek po uražení své dráhy zaniknout (minul svůj cíl), nebo inicializovat nový výbuch.

Takto vytvořený paprsek se vloží do seznamu paprsků, který uchovává všechny vystřelené paprsky. Pravidelně podle časovače se aktualizuje jejich poloha a paprsky se vykreslují. V případě zániku paprsku je tento ze seznamu vymazán. Pokud se zároveň jedná o paprsek, který má způsobit výbuch, je takto skutečnost oznámena objektu, který spravuje všechny částicové systémy. Ten podle identifikátoru pozná, který paprsek událost vyvolal, a ví, jaké částicové systémy inicializovat. Jedná se vetšinou o samostatné výbuchy nebo o výbuch a dým, který se spustí o pár okamžiků po doznění exploze. O částicových systémech bylo již pojednáno v kapitole Efekty s částicovými systémy.

8 Osvětlení a stíny

V této kapitole bude představena jedna z důležitých součástí každé grafické práce - osvětlení a stíny. Teprve správné nasvícení scény ji dodává na realističnosti a lze pomocí něj dosáhnout i mnoha zajímavých efektů. Osvětlení se používá téměř ve všech odvětvích grafiky, od složitých demo aplikací a animovaných filmů až po různé třeba i jednoduché vizualizace, neboť světlo je prostředkem vizuálního vnímání světa. Dovoluje nám utvořit si představu o materiálových vlastnostech a šíření světla prostorem je základem pro tvorbu virtuálních scén a jejich zobrazování.

Stíny jsou důležité při prostorovém vnímání člověka. Díky nim lze snáze pochopit rozmístění objektů, jejich tvar a rozměry a také poskytují informace o poloze a vlastnostech zdrojů světel. Stíny jsou důležitým prvkem každé scény, která má vypadat věrohodně a realisticky. Další informace o osvětlení a popis základních modelů osvětlení lze nalézt v [9].

8.1 Nastavení a řízení světel

Osvětlování je celkově výpočetně náročný proces. Již samotný algoritmus osvětlování je časově nákladný, navíc se musí spočítat parametry světla pro každý vrchol a od každého zdroje světla. To je v rozporu s tím, mít co nejzajímavější a co nejlépe nasvícené scény. Je tedy vhodné naplánovat, jak nejlépe využít co nejméně světel. V této práci se podařilo snížit počet použitých světel na čtyři, přičemž zůstalo osvětlení tak, jak bylo původně naplánováno. Je to na úkor o něco složitějšího plánu pro nastavování světel a napomohla tomu i vhodně zvolená trajektorie avatara. Na druhou stranu jsou po většinu času využita všechna čtyři světla, takže se nemusí jednotlivá světla při vykreslování neustále vypínat a zapínat.

Bylo tedy potřeba určit, kdy přesně jsou vidět která světla a jejich účinky a podle toho naplánovat jejich rozmístění a zapínání ve vhodnou dobu. Některé situace mohou vyžadovat velký počet světel ve scéně, protože prostor, kterým avatar prochází je příliš rozlehlý na to, aby se světla skryla za rohy nebo jiné překážky. Tyto situace byly vyřešeny ve spolupráci s pohybem avatara. Ten se v jistém, šikovně vybraném místě otočil nebo rozhlédl, zatímco za jeho zády se přepnula některá světla tak, aby tuto operaci pozorovatel neviděl.

Dále byla snaha, aby se parametry světel měnily co nejméně a tudíž aby světla, která si jsou ve scéně svými vlastnostmi podobná, byla vždy přiřazena stejným světlům vykreslovacího stroje. Takže například směrová světla byla vždy přiřazena k co nejnižším světlům OpenGL.

Světelné zdroje se inicializují vždy po přechodu do nové oblasti, a to právě jednou, takže poměrně náročná inicializace nezdržuje výpočet. Výjimkou z tohoto pravidla jsou světla, která jsou určitým způsobem časována. U těch inicializace probíhá celou dobu aktivace těchto světel. Po jejím

dokončení však už opět probíhá jen nastavení jejich pozice. Ta se kvůli pohybu musí nastavovat v každém cyklu vykreslování, ještě před pohledovou transformací, jinak by se světla pohybovala zároveň s avatarem a nebyla by v podzemí statická. Výjimkou jsou ještě světla, která se v čase nějak mění, např. jejich intenzita, barva atd. U těch je nutné tuto měnící se vlastnost měnit v každém cyklu, ale opět stačí měnit pouze tuto jednu vlastnost, což netrvá tak dlouho jako celá inicializace. Podívejme se nyní na konkrétní řešení jednotlivých osvětlovacích efektů.

8.1.1 Pochodeň a výstražné světlo

Pochodeň a výstražné světlo jsou příklady těch světel, které se inicializují pouze jednou, ale jejich intenzita se mění v každém cyklu. Jsou to také první světla použitá v demu. Tato světla jsou tedy zapnuta v době, kdy ještě nejsou vidět ani jejich zdroje, ani například odraz na stěně. Kdyby tomu tak bylo, pozorovatel by zapnutí osvětlení zpozoroval, což právě není účelem. Požadováno naopak je, aby světla budila dojem, že osvětlují prostor už dávno před tím, než na dané místo pozorovatel přijde. To je zajištěno právě vhodným navržením oblastí, takže při přechodu do nové oblasti jsou zapínaná světla například za dvojitým ohybem chodby (kvůli osvětleným stěnám) apod.

Celkově mají tato světla nastaven velký index útlumu, takže osvětlují jen své blízké okolí, což je třeba pro pochodeň typické. Navíc zmíněnou změnou intenzity je dosaženo efektu blikání plápolajícího ohně. Aby byl tento efekt opravdu přesvědčivý, musí být změna náhodná a ne skoková, takže ve skutečnosti se intenzita pouze zvyšuje a snižuje. Náhodně se určuje pouze to, kdy se má směr změny intenzity obrátit.

Naopak v případě elektrického varovného světla by tato náhodnost působila rušivě. Změna intenzity je tedy zcela periodická, ale má vysokou frekvenci, takže osvětlení evokuje svit žárovky.

8.1.2 Zářivky

Po příchodu do první prostorné haly v podzemí je tato místnost celá potemnělá. Teprve po několika okamžicích začne nesměle blikat první světlo, až se postupně rozzáří všechny čtyři zářivky v místnosti. Dvě z nich jsou klasická bílá světla, jedno má žlutavý nádech, takže budí dojem dosluhující zářivky. Čtvrtá zářivka má špatný kontakt. Většinu doby svítí s menší intenzitou než ostatní a v pravidelných intervalech problikává do jasného světla.

Tyto zářivky jsou příkladem světel s časováním na začátku. Jakmile avatar vstoupí do oblasti této místnosti, spustí se hodiny, které začnou odpočítávat čas. Zářivky se zapínají postupně asi se sekundovým odstupem. Efektu nažhavení zářivek bylo dosaženo tím, že se zdroj světla imitující zářivku vždy rychle několikrát po sobě zapnul a zase vypnul. Po zapnutí všech zářivek je dokončena inicializace světel a již ji není třeba měnit. Výjimkou je blikající zářivka, která podobně jako varovné

světlo pravidelně mění svoji intenzitu s tím rozdílem, že průběh nemá sinusový tvar, ale intenzita vždy nějakou dobu zůstává na nízké hladině.

Stejné je to i se zářivkami v poslední místnosti. Tam žádné blikající zářivky nejsou, takže je situace ještě jednodušší. Dojde pouze k prvotnímu časování. Navíc některé z nich svítí už při příchodu, takže efekt rozsvícení není tak patrný. Zato je po chvíli patrný efekt toho, že všechny zářivky zhasnou. Jedná se pouze o inverzní proces, takže při zhasnutí každé zářivky tato několikrát problikne a zůstane vypnutá. Místnost se ponoří do tmy.

8.1.3 Halogen, lampy, měsíc

Všechna tato světla jsou bodová světla a pozorovatel je spatří v druhé polovině dema. Halogen osvětluje druhou velkou místnost, lampy připevněné na zdi a měsíc osvětlují největší místnost podzemí, která jako jediná nemá strop a umožňuje pohled na noční oblohu.

Jsou to světla bez časování i bez průběžné změny, kromě úvodního nastavení se s nimi už dál nic neděje. Bylo spíše potřeba nastavit vhodný exponent útlumu pro tato světla. Čím je exponent menší, tím jsou světla ostřejší a je jasně vidět hranice mezi osvětlenou a neosvětlenou částí. Čím je exponent větší, tím je přechod pozvolnější.

U světla měsíce je exponent větší, jeho směrová podstata totiž nesmí být patrná, protože světlo letí z velké dálky. Přitom ale byla snaha zachovat určité ohraničení a soustředění měsíčního světla kvůli otvoru ve stropě. Naopak reflektor a lampy jsou ostrá světla, u kterých šlo právě o to, aby byl vidět rozdíl. Lampy jsou dokonce umístěné přímo na zdi tak, že je vidět nejen klasický osvětlený kruh na podlaze, ale i celý světelný kužel na zdi pod lampami. Tato světla mají nastavený menší exponent.

Spíše než samotné nastavení světel je zde problém přímo v modelu podzemí. Stěny jsou tvořeny určitým počtem polygonů a ty jsou pak na osvětleném povrchu patrné. Místo toho, aby byla hranice pěkně vyhlazená, je kostrbatá. Určitým řešením by bylo stěnu rozdělit na více polygonů, čemuž se věnuje kapitola Řízení vykreslování. V tom případě by ale vznikl velký počet vrcholů a výpočet osvětlení by trval příliš dlouho. Čím více polygonů by tvořilo stěnu, tím by byl tvar hranice pěknější a vyhlazenější, ale o to déle by výpočet trval. Čím méně by jich bylo, tím rychlejší výpočet bude, ale o to bude hranice schodovitější. Úplně nejlepším řešením by bylo rozdělit na menší polygony stěnu jen v místech hranice osvětlení. K tomu by ale bylo potřeba sofistikovanější algoritmus dělení ploch a jeho určité řízení vhledem k nastavení světla. Pro tuto práci to však není tak podstatné, aby se tím zabývala.

Místo toho je těmto světlům nastaven o něco vyšší exponent útlumu. Hranice osvětlené části se tím mírně vyhladí, ale světlo zůstane pořád dost ostré na to, aby byla vidět. Sice se tím problém schodovitosti neodstraní, ale aspoň částečně potlačí.

8.2 Stíny

Přesto, že jsou stíny důležitou součástí počítačové grafiky, neexistuje jednoduchý způsob, jak je automaticky do scény generovat. Existují metody jako ray tracing, particle tracing, radiozita a mnoho dalších, které to umožňují, a poskytují podle zvoleného algoritmu velmi dobré výsledky, ale žádná z těchto metod není vhodná pro výpočet v reálném čase. I když je snaha tyto algoritmy upravit a zrychlit, pořád není dosaženo úplného real time výpočtu a generování jednoho snímku scény může u nejnáročnějších algoritmů trvat i několik hodin.

Často se tedy stíny předpočítají do tzv. stínových map, což jsou v podstatě obyčejné textury. Jeden z algoritmů, pro takovýto výpočet je popsán v [1] na konci kapitoly o mapování textur. Podléhá však mnoha chybám a po jeho vyzkoušení od něj bylo raději upuštěno.

Nakonec jsou pro vytvoření stínů v této práci zkombinovány dva přístupy, ručně vytvořené stínové mapy a práce se stencil bufferem.

Stencil buffer je použit především na stíny v ohybech chodeb nebo obecně na stíny způsobené přímo podzemím samotným. V tom případě je od příslušné stínící stěny vytvořen stínový polygon a části scény v tomto stínovém polygonu jsou vykresleny s jinými světly než části scény mimo stínový polygon. Stejným způsobem lze vytvořit naopak i světlo dopadající do místnosti skrze okno. Celkově se jedná o obyčejnou práci se stencil bufferem, která je taktéž popsána v [1]. Mimo to bylo v plánu použít stencil buffer i pro vytvoření stínů vrhaných předměty v podzemí. To se nakonec ukázalo jako extrémně složité a bylo od tohoto záměru upuštěno. Pouze v první místnosti podzemí jsou stíny od obou beden vytvořeny pomocí stencil bufferu. Nevýhodou stencil bufferu je to, že vytváří jen ostré stíny, které nejsou realistické, a navíc se musí scéna vykreslovat víckrát, takže to zpomaluje výkon aplikace.

Lepší, ale umělecky náročnější, je způsob využití ručně nakreslených stínových map, který je použit pro všechny ostatní případy stínů v podzemí. Jedná se například o stíny vrhané bednami a barely, zničenými stroji či sloupky plotu. Stínová mapa se nanese jako textura s průhledností na místo, kam má dopadat stín od daného objektu. Základem je ovšem dobrá a věrohodná stínová mapa, na jejíž vytvoření bohužel neexistuje žádný návod. Jedná se o subjektivní práci a výsledek je do značné míry ovlivněn šikovností designéra, který mapu vytváří. Touto metodou lze vytvořit realisticky vypadající měkké stíny a ani nezpomaluje aplikaci, protože pro vykreslení jedné stínové mapy stačí jediný polygon, na který bude mapa nanesena. Metoda tedy vyžaduje pouhé čtyři vrcholy navíc oproti dvěma vykreslovacím průchodům při použití stencil bufferu.

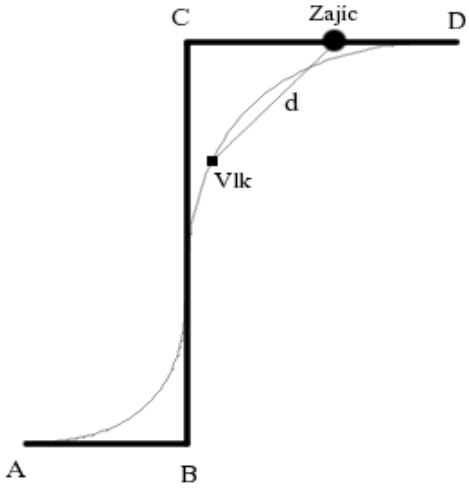
9 Pohyb

V této kapitole bude popsán mechanismus pohybu avatara a létajících strojů v podzemí. Na rozdíl od her se nejedná o interaktivní aplikaci, takže není vyžadován žádný zásah uživatele. Aplikace je řízena zcela autonomně.

9.1 Vlk a zajíc

Pohyb kamery v prostoru lze popsat mnoha způsoby a záleží na jeho složitosti, jaký popis je pro daný pohyb nejlepší. Pokud se má jednat o nějaký jednoduchý lineární pohyb, určitě se pro něj nebude vytvářet žádný složitý model, ale popíše se několika úsečkami. Podobně lze popsat pravidelný pohyb po křivce například sinusoidou nebo nepravidelný pohyb Beziérovými křivkami. Možností je nepřeberné množství. Pokud je ale potřeba popsat nějaký složitý a dlouhý pohyb, trvalo by to podobným způsobem příliš dlouho a bylo by to velice pracné. V takových případech je lepší zvážit jiné možnosti definování pohybu, které lze snadno popsat a modifikovat.

Model pohybu nazvaný vlk a zajíc umožňuje pomocí jednoduchého popisu trajektorie definovat poměrně složitý pohyb v prostoru. Tento model názorně ukazuje obrázek 4. Popis pohybu je dán popisem pohybu bodu zvaného zajíc. Jeho trajektorie může být velice jednoduchá a na obrázku je zvýrazněna tlustou čarou. V podstatě jsou zaznamenány jen souřadnice bodů A, B, C, D a přesná poloha na trajektorii je dána rychlostí zajíce a časem.



Obr 4: Model pohybu vlk a zajíc

Kamera je umístěna do bodu zvaného vlk a je neustále natočena na bod zajíce. Ať se zajíc pohne kamkoliv, vlk jej stále sleduje. Dále je definována určitá vzdálenost *d*, kterou má na počátku vlk od zajíce a kterou se snaží pořád udržovat. Takže jakmile se zajíc pohne a tato vzdálenost se

zvětší, pohne se ihned i vlk směrem k zajíci, aby ho zase na danou vzdálenost dohnal. Tímto pohybem potom vznikne oblá trajektorie vlka - kamery, znázorněná na obrázku tenkou linií. Čím větší bude rychlost zajíce, tím větší bude mezi ním a vlkem vzdálenost a tím rychleji bude vlk zajíce pronásledovat. Pokud zajíc zpomalí, vzdálenost se zmenší a tudíž zpomalí i vlk. Když zastaví zajíc, zastaví i vlk.

Určením počáteční vzdálenosti je v podstatě definována i oblost trajektorie vlka. Čím bude vzdálenost větší, tím oblejší trajektorie bude. Čím bude menší, tím více bude vlk kopírovat trajektorii zajíce. V případě vzdálenosti 0 bude trajektorie zajíce a vlka téměř totožná.

Je tedy vidět, že i takto jednoduchý popis trajektorie s několika málo parametry jako rychlost zajíce a vzdálenost od něj může popisovat poměrně složitý pohyb kamery.

9.2 Formát uložení trajektorie

Celý pohyb kamery v podzemí je popsán v souboru `BunnyPoints.txt`, kde je definována právě trajektorie zajíce. Formát vypadá následovně:

NUMDODGEPOINT pocet_bodu_trajektorie
STARTINGPOINT startovaci_bod

x y z v1 v2 d t typ p1x p1y p1z p2x p2y p2z
x y z v1 v2 d t typ p1x p1y p1z p2x p2y p2z (12)

// další body trajektorie

Soubor opět začíná hlavičkou, kde informace *pocet_bodu_trajektorie* slouží pro alokování dynamického pole, do kterého jsou po spuštění aplikace body načteny. *startovaci_bod* udává první bod, kde má pohyb na trajektorii začínat, a slouží pouze pro ladící účely. *x, y, z* jsou souřadnice bodu trajektorie, *v1* je počáteční rychlost a *v2* koncová rychlost na daném úseku, který začíná stávajícím bodem a končí v následujícím. Stejně tak všechny další údaje platí vždy pouze pro aktuální úsek a mohou se tedy v průběhu pohybu různě měnit. *d* je nulová vzdálenost mezi vlkem a zajícem, *t* je čas, po který zajíc čeká, než se opět po překonání aktuálního úseku rozběhne. Parametr *typ* udává, kterým z několika možných způsobů bude vlk zajíce pronásledovat. Ty jsou popsané v podkapitole Pohyb avatara. Zbývající parametry jsou souřadnice počátečního a koncového bodu pohledu pro aktuální úsek a platí pouze pro určité hodnoty parametru *typ*.

9.3 Pohyb avatara

Implementaci tohoto algoritmu není třeba blíže představovat. Jedná se o aplikaci základních vzorců kinematiky, které lze nalézt v každé středoškolské učebnici fyziky, a o práci s vektory. Zajímavější bude představení několika typů pohybu vlka za zajícem. Nejjednodušší způsob je ten, který byl popsán v podkapitole Vlk a zajíc. Jeho nedostatky se ale projeví hned při prvním průchodu ohybem chodby. Kamera je totiž natočena proti přední zdi, dokud zajíc nezatočí, a to i v případě, když už by se měl dávno dívat do chodby, kam zatáčí. I když je pohyb podle požadavků pěkně oblý, nepůsobí v některých případech realisticky, zvláště má-li jako v tomto případě napodobovat chování člověka. Bylo tedy potřeba zajistit, aby se kamera pohybovala jedním směrem, ale dívala se do směru jiného. Za tím účelem byla vytvořena druhá, pohledová trajektorie. Ta je určena pro každý úsek zvlášť a je v souboru popsána šesti volitelnými parametry na konci řádku. Jsou to souřadnice počátečního a koncového bodu. Princip funguje následovně. Když zajíc dojde do nového bodu, zjistí se, jakým typem pohybu se bude následující úsek pohybovat. Jestliže je to popisovaný mód, kdy jde vlk jinam než se dívá, vytvoří se i druhý, tzv. pohledový zajíc, který nastupuje svoji cestu ke koncovému bodu pohledové trajektorie. Vlk tedy putuje za původním zajícem, ale dívá se za zajícem pohledovým. Tento mód má dvě úskalí. Prvním je, že je pro plynulý pohled kamery třeba zajistit spojitost cesty, kterou vlk sleduje. Tedy musí na sebe navazovat trajektorie zajíce, pohledového zajíce a pohledových zajíců v jednotlivých úsecích navzájem.

Druhým, mnohem nenápadnějším, ale o to závažnějším problémem je to, že zajíc i pohledový zajíc musí dorazit do koncového bodu svých úseků zároveň. Oba dva úseky přitom mají jinou délku a navíc nemusí být rychlost zajíce konstantní. Musí se tady neustále upravovat rychlost pohledového zajíce podle toho, kolik ještě zbývá urazit cesty skutečnému zajíci. Kdyby se totiž spočítala pouhá průměrná rychlost, nedorazili by do koncového bodu společně. Nová pohledová trajektorie potom má jiné než definované parametry a tato odchylka se neustále zvětšuje. V mnoha případech to nevadí a ani není poznat. Vyskytlo se však několik míst, kde se nastřádaná chyba projevovala náhlým nepochopitelnými zastavením pohledového zajíce či jinými anomáliemi, kdy se zajíc nechoval tak, jak měl popsáno ve své trajektorii.

Jinou vlastností základní verze tohoto modelu je, že se vlk snaží dostat vždy za zajíce. Někdy je ale toto chování nežádoucí a je potřeba, aby se vlk pohyboval paralelně se zajícem. V tom případě se pro vlka nepočítá směrový vektor k zajíci, ale je mu přiřazen přímo směrový vektor zajíce, který je aplikován na souřadnice vlka. Tento mód lze jednoduše zkombinovat s předchozím, takže vlk může putovat paralelně se skutečným zajícem a dívat se jiným směrem. Lze ho také pohledovým módem úplně nahradit, ale to nemusí být vždy výhodné, proto byl tento mód také plně implementován.

Samozřejmostí je i mód, kdy vlk stojí na místě a jen se rozhlíží kolem, tedy sleduje pohledového zajíce.

Výše popsaný model podléhá ještě jedné nepříjemné okolnosti, která vychází z časovacího systému. Vždy, když zajíc dojde do koncového bodu úseku, je na něj navázáno a zajíc pokračuje dalším úsekem. To, že zajíc do tohoto koncového bodu dospěl, je indikováno tím, že mu zbývá nulová dráha, kterou má ještě urazit. Může být ale zvolen i jiný princip rozhodování, jestli je již zajíc v koncovém bodě úseku. Celkově se totiž jedná o to, že se zajíc pohybuje v závislosti na čase. Nyní si představme situaci, že je zajíc již blízko koncového bodu a v příštím cyklu by do něj dospěl a tedy by pak pokračoval novým úsekem kupříkladu úplně jiným směrem. V tomto okamžiku ale dojde vlivem různých okolností (dlouhý výpočet, pomalý procesor, zásah uživatele...) k tomu, že se zajíc dlouho neaktualizuje a tedy se nekontroluje ani podmínka na dosažení koncového bodu úseku. Zajíc v tom případě pokračuje stále svým původním směrem, přestože dávno překročil koncový bod. Ve chvíli, kdy opět dojde ke kontrole, se sice správně indikuje přechod do nového úseku, ale to už může být pozdě. Zajíc se vyskytuje daleko od své původní dráhy, může se dokonce dostat do jiné oblasti, čímž se přepnou i světla, aktivují částicové systémy, modely, které vůbec aktivní být nemají, atd. Dojde tedy k naprosto jinému chování, než bylo očekáváno, a k úplně jinému vzhledu demo aplikace. Jednoduše řečeno bude minimálně část demo ukázky totálně znehodnocena. K těmto jevům skutečně docházelo, ale pouze na pomalých procesorech ve chvílích, kdy avatar přecházel do některých nových oblastí, které byly obzvláště náročné na inicializaci. U výkonnějších sestav již tento problém pozorován nebyl, protože výpočet inicializace proběhl vždy dostatečně rychle.

Jednoduchým a efektivním řešením, jak se tomuto vyhnout i u pomalejších sestav je zamezit, aby mohl přírůstek času nabývat velkých hodnot. Ve chvíli, kdy k tomu dojde, je automaticky snížen na určitou pevnou akceptovatelnou hodnotu. I když tedy dojde k mírnému seknutí demo ukázky, opět se rozjede tak, jak má, a nedojde k jejímu pokažení. Problém byl tímto způsobem v práci vyřešen.

Dalo by se vymyslet i ještě elegantnější řešení, aby ani na pomalých sestavách nedocházelo ani k sekání aplikace. Bylo by například možné trajektorii vygenerovat a uložit do souboru. Po spuštění demo ukázky by se pak již nemusela počítat, ale pouze by se braly hodnoty ze souboru a interpolovaly podle času. Jelikož ale toto není primárním cílem práce a předpokládá se spuštění na některém z dnešních strojů, které výpočet bez problémů zvládají, nebyla tato pokročilejší možnost řešena.

9.4 Rozšíření modelu pro pohyb objektů

Nebylo nic jednoduššího než stejným způsobem jako pohyb kamery popsat i pohyb strojů v podzemí. Jejich trajektorie jsou uloženy v souborech `Rocket1.txt`, `Rocket2.txt`, `Hovercraft1.txt` a `Hovercraft2.txt`. Názvy přímo korespondují s objektem, jehož trajektorii popisují.

Dokonce se i v tomto případě uplatnil pohledový mód. Vznášedla totiž mají otočnou věž a pohyblivé dělo. Obě tyto části se mohou pohybovat nezávisle na vznášedlu a na sobě navzájem. Trajektorii vznášedla tedy udává pohyb zajíce, zatímco hlaveň děla ve spolupráci s otočnou věží sledují pohledového zajíce.

Tento případ byl rozšířen ještě o jednu další možnost, a to je sledování jiného pohyblivého objektu, konkrétně nepřátelských raket. Nešlo o nic jiného, než že se modelu místo souřadnic pohledové trajektorie ze souboru předala parametrem trajektorie rakety a ta se použila jako nová pohledová trajektorie.

Protože ale vznášedlo v demu nesleduje pouze jednu raketu, ale více, pohledová trajektorie se neustále mění. Bylo tedy potřeba zajistit plynulý přechod z jedné trajektorie na druhou, aby nedocházelo k prudkým, nepřirozeným skokům. V takovém případě byl jeden úsek pohybu vyhrazen pro „změnu“ trajektorie. Tohoto speciálního módu bylo využito i při změně externí trajektorie (trajektorie raket) na vlastní trajektorii (definovanou v souboru) a naopak. Nebylo pak potřeba se starat o návaznost.

Tento mód funguje tak, že se jako startovací bod pohledové trajektorie přiřadí aktuální bod pohledového zajíce. Cílový bod se však neustále mění a odpovídá zajíci na nové trajektorii. Kdyby byl za cílový bod vzat aktuální bod zajíce nové trajektorie, byl by tento nový zajíc v době, kdy na sebe mají aktuální a nový zajíc navázat, na jiném místě a ke skoku by stejně došlo. Takto je zajištěno, že v kritické době, kdy se oba zajíci setkají, budou na stejném místě. Přechod mezi libovolnými trajektoriemi je tedy pěkně plynulý.

10 Zvuk

I když bylo cílem práce vytvořit grafickou demo ukázku, byla k ní vytvořena i zvuková stopa. Zvuk totiž silně dotváří emocionální zážitek z vizuálního vjemu. Dokonce je pro navození atmosféry mnohem důležitější než obraz. Vztahem mezi obrazem a zvukem se zabývá diplomová práce [21].

Protože je zvuk k této práci jen doplňkem, byl k ní přidán tím nejjednodušším způsobem, i když sebou toto řešení nese značná úskalí. Nebyla použita žádná zvuková knihovna (např. OpenAL, Mini Fmod), ale byl použit jednoduchý příkaz WinAPI PlaySound se všemi jeho nevýhodami. Tato funkce dovoluje přehrát naráz jen jeden zvukový wav soubor a neumožňuje ani aplikaci žádných efektů. Zvukové efekty i podkladová hudba byly zpracovány v programu Audacity [20], kde byly všechny požadované efekty aplikované přímo na originální zvuk a kde byly zvukové stopy smíchány do výsledného stereo souboru. Nevadí tedy, že funkce PlaySound přehraje jen jeden soubor, protože v něm jsou již všechny zvuky i hudba smíchané a je tedy možné slyšet i více zvuků najednou.

Podkladová hudba je originální tvorbou a nelze tedy očekávat profesionální úroveň. Základy zvukových efektů byly staženy ze zdrojů na internetu [22, 23] a případně dále upraveny. Více o zvukových efektech je možné se dočíst v [24].

Nejdříve byl pro uložení zvukové stopy použit jediný wav soubor. Ukázalo se ale, že ne vždy by byl zvuk i obraz synchronní až do konce demo ukázky. Příčinou je jednak dlouho trvající inicializace nových oblastí na pomalejších architekturách, jak je popsáno na konci podkapitoly Pohyb avatara a také doba načítání wav souboru funkcí PlaySound. Přestože na dnes běžných počítačích by tento problém neměl nastat, byl zvukový soubor rozdělen na několik menších. Jejich přehrání je pak řízeno přímo výskytem avatara v podzemí a i kdyby někde došlo k posunu zvuku a obrazu, následující zvukový soubor bude s obrazem opět synchronizován. Ke spuštění každého audio souboru totiž dojde, když avatar dosáhne určitého bodu ve své trajektorii, a není tak závislé na čase.

Velkou nevýhodou funkce PlaySound je, že umožňuje přehrání právě jen nekomprimovaného formátu wav. Při dané délce demo ukázky a při volbě alespoň průměrné kvality by měl výsledný soubor velikost kolem 60MB, což je opravdu hodně. Jednak kvůli samotné velikosti, když by se měl tento soubor například stahovat z internetu, tak i kvůli době, která je potřeba na zpracování funkcí PlaySound. Dochází tak při načítání (zejména z externích pamětí, CD, flash) kvůli přenosové rychlosti a dalším vlivům ke zpoždění a je téměř nemožné zaručit synchronizaci s obrazem. To je bez možnosti načtení souboru do paměti v tomto případě neřešitelný problém a tato chyba se může kdykoliv vyskytnout.

Co se velikosti zvukového souboru týče, není třeba mít zvuk v nejlepší kvalitě a jako podklad pro demo stačí i horší parametry zvuku. Byl tedy snížen bitrate i frekvence. Velikost souboru se tím snížila na přijatelnou velikost, ovšem za cenu nižší kvality zvuku.

11 Závěr

V této práci byl vytvořen jednoduchý komplex pěti místností propojených chodbami, kterými prochází avatar po předem zadané trajektorii. Ta je navržena tak, aby si pozorovatel mohl prohlédnout celé podzemí, jeho zajímavosti a efekty v něm vytvořené.

Jako statické objekty byly do podzemí různě rozmístěny barely a několik typů beden. Dále se v něm nachází několik složitějších prvků jako mříž, přes kterou lze vidět do druhé místnosti, plot nebo tekoucí voda.

Atmosféra podzemního komplexu byla podpořena vhodným nastavením osvětlení a několika světelnými efekty. K osvětlení se váží i stíny, které byly v práci vytvořeny dvěma způsoby, pomocí stencil bufferu a užitím stínových map.

Hlavní důraz byl kladen na částicové systémy a pohyblivé modely bojujících strojů. Částicové efekty jsou reprezentovány několika typy výbuchů, dýmů a ohňů a tvoří asi nejzajímavější efekty práce. Naproti tomu modely a jejich chování zajišťují dění v podzemí. Po tom, co je pozorovatel v podzemí objeví, se stroje „proberou“ k životu a začnou mezi sebou bojovat. Jejich pohyb je zadán tak, aby připomínal válečné manévrování, při kterém po sobě navíc střílejí laserovými paprsky. Bojová scéna je tedy plná akce, výbuchů a dalších efektů.

Pro emotivní doplnění demo ukázky byla přidána i podkladová hudba s různými zvukovými efekty navázanými na dění v podzemí.

Při vytváření této práce tak bylo možné získat mnoho cenných zkušeností. Byly to jak základní dovednosti, týkající se programování v OpenGL a využívání jeho možností, tak i pokročilé možnosti nastavení a řízení vykreslování. Konkrétně tak bylo použito osvětlení, jeho nastavení a parametrizace, s tím spojené využití materiálů, texturování a celá obsáhlá kapitola spojená s nimi, práce s display listy a zobrazovacími seznamy. Významnou část tvoří práce s transformacemi a metody projekce. V neposlední řadě i práce se stencil bufferem, blending a využití Z-bufferu.

Dále práce umožnila získání zcela nových zkušeností s různými pokročilými zobrazovacími technikami, kam spadají částicové systémy, načítání, obsluha a vykreslování modelů nebo část týkající se pohybu. Ve všech těchto částech se vždy vyskytl nějaký problém, který bylo třeba vyřešit, takže kromě samotného seznámení s těmito technikami jim bylo třeba i podrobněji porozumět.

Velice cenné zkušenosti byly získány při vytváření celkového konceptu aplikace. Ten bylo potřeba v průběhu práce měnit, vylepšovat a zdokonalovat, aby jednoduše a vhodně obsáhl všechny aspekty práce. Spadá sem třeba řešení viditelnosti a řízení hustoty polygonové sítě, což byly důležité momenty pro to, aby bylo vykreslování demo ukázky plynulé a netrhalo se.

Důležitým prvkem byla také interakce mezi jednotlivými událostmi v demu, protože ty jsou na sobě různým způsobem závislé, a celkově řízení všech událostí, jež se v demu odehrávají, navazují a reagují na sebe.

Tuto práci je samozřejmě možné rozšířit, a to všemi myslitelnými způsoby. Může se jednat o prostorové rozšíření podzemí o další místnosti, chodby a jiné architektonické záležitosti. Mohou být přidány další objekty, modely nebo zcela nové efekty. Lze přidat další stroje a naprogramovat jim novou trajektorii, aniž by byl ovlivněn stávající plán demo ukázky.

Nabízí se i možnost zdokonalit již hotové části práce, ať už výbuchy, pohyb nebo samotné podzemí. Při otvírání okna programu musí uživatel počkat cca 3 sekundy, než se objeví obraz, protože se načítají textury celého podzemí. Tento problém by šel vyřešit tak, že by se jednotlivé textury načítaly až ve chvíli, kdy jsou skutečně potřeba.

V případě výbuchů je pro každou částici použita jedna textura, což může při malém počtu částic nebo v případě velkých částic vytvářet škaredé vizuální artefakty. Řešením by bylo použít více druhů textur pro stejný typ částic. To ale neřeší situaci, kdy bude částic málo. Lepším způsobem je přiřadit jedné částici více textur, které se budou v čase měnit. Textury musí být samozřejmě vhodně vytvořeny tak, aby v čase působily kontinuálně. Špatně zvolená posloupnost textur může problikávat, mizet a zase se objevovat na jiném místě nebo působit jinak ošklivě. Časová návaznost textur je pro tento případ velice důležitá. Jedna z publikací, která se touto problematikou zabývá, je [11]. Potřebnou vhodnou texturu lze získat pomocí šumových funkcí, ze skutečných obrázků nebo i z videa reálného ohně. Když by potom byl částicový systém tvořen i jedinou částicí, vypadal by realisticky, protože by se textura neustále měnila. Tím pádem by to nevypadalo, jako že se pohybuje jedna částice pevného tvaru daným směrem, ale že se pohybuje celý měnící se částicový systém. I obarvování částic lze simulovat lépe realisticky a je popsáno v [12].

Jelikož jsou částice tvořeny impostory, určitě se vyskytnou vizuální artefakty, když budou výbuchy umístěny v blízkosti jiných, pevných objektů. Dojde totiž k tomu, že bližší pevný objekt zakryje kus výbuchu a v místech, kde se budou protínat částice s objektem, vzniknou ostré nápadné hrany. Ty se v realitě nikdy neobjeví, působí škaredě a degradují celý částicový systém. Způsob, jak tento jev potlačit, je popsán v [13].

Tak jako výbuchy lze rozšířit a vylepšit všechny části této práce. Mnoho návrhů je popsáno nebo alespoň zmíněno přímo v jednotlivých kapitolách.

Kromě přidávání nových částí a efektů a vylepšování těch stávajících je možné demo ukázku přetvořit několika více či méně jednoduchými úpravami na akční 3D střílečku typu Doom nebo Quake. Je vidět, že možností na pokračování této práce je opravdu mnoho. Záleží jen na fantazii každého, co si kdo dokáže vymyslet.

Literatura

[1] D. Shreiner at all: *OpenGL průvodce programátora*. Brno: Computer Press, 2006, ISBN: 80-251-1275-6

[2] WWW stránky: OpenGL, URL: http://www.opengl.org/ (květen 2008)

[3] P. Pokorný: *Blender - naučte se 3D grafiku*, Ben, 1. české vydání, 2006, ISBN 80-7300-203-5

[4] WWW stránky: Blender, URL: http://www.blender.org/, (květen 2008)

[5] WWW stránky: CodeColony, URL: http://www.codecolony.de/ (květen 2008)

[6] WWW stránky: NeHe OpenGL Tutoriály, URL: http://nehe.ceske-hry.cz/tut_10.php (květen 2008)

[7] W. T. Reeves. *Particle systems - techniques for modelling a class of fuzzy objects*. In SIGGRAPH ’83 Proceedings, pages 359–376, 1983.

[8] W. Reeves, D. *Particle systems – A technique for modeling a class of fuzzy objects*. ACM Transaction on Graphics, 2(2):12-22, 1983

[9] J. Žára a spol.: *Moderní počítačová grafika* . Brno: Computer press, druhé přepracované a rozšířené vydání, kompletní průvodce metodami 2D a 3D grafiky, 2004,

ISBN 80-251-0454-0, 609 s.

[10] K. Pallister: *Game Programming Gems 5*, Charles River Media, Inc., 2005, ISBN: 1584503521

[11] X. Wei at all: *Simulating Fire With Texture Splats.* Proceedings of the conference on Visualization '02, Pages: 227 – 235, ISBN:0-7803-7498-3

[12] J. Stam and Fiume, E. L. 1995. *Depicting fire and other gaseous phenomena using diffusion processes*. In Proceedings of ACM SIGGRAPH 95, 129–136.

[13] T. Umenhoffer at all: *Spherical Billboards and their Application to Rendering Explosions.* Proceedings of Graphics Interface 2006, Pages: 57 – 63, ISBN ~ ISSN:0713-5424,

1-56881-308-2

[14] K. Pallister at all: *Game Programming Gems 3*, Charles River Media, Inc., 2002,

ISBN: 1-58450-233-9

[15] WWW stránky: Grafika on-line: URL: http://www.grafika.cz/art/3d/clanek726655005.html (květen 2008)

[16] WWW stránky: Root.cz, URL: http://www.root.cz/clanky/vrml-jazyk-pro-popis-virtualni-reality (květen 2008)

[17] WWW stránky: 3Dsource.de, URL: http://www.3dsource.de/deutsch/vrml.htm (květen 2008)

[18] WWW stránky: Media Machines, URL: http://www.mediamachines.com/developer.php (květen 2008)

[19] WWW stránky: GIMP, URL: http://www.gimp.org (květen 2008)

[20] WWW stránky:Audacity, URL: http://audacity.sourceforge.net (květen 2008)

[21] A. Koutný: Vztah zvuku a kinetického obrazu. Brno, 2004, diplomová práce, FaVU VUT v Brně.

[22] WWW stránky: Radio Sounds, URL: http://www.a1freesoundeffects.com/radio.html (květen 2008)

[23] WWW stránky: Royalty Free Music & SFX , URL: http://www.partnersinrhyme.com/soundfx/warsounds.shtml (květen 2008)

[24] WWW stránky: Sound effect, URL: http://en.wikipedia.org/wiki/Sound_effect, (květen 2008)

[25] WWW stránky: Root.cz, URL: http://www.root.cz/serialy/tvorba-prenositelnych-grafickych-aplikaci-vyuzivajicich-knihovnu-glut/, (květen 2008)

Příloha A

O programu

Program je spustitelný pod operačním systémem Windows a vyžaduje knihovnu Glut (je přiložena). Výsledná aplikace projektu je tvořena standardním oknem o základním rozlišení 1067 x 600 a lze přepnout do celoobrazovkového módu. Program nevyžaduje žádnou interakci uživatele a po skončení demo ukázky se sám ukončí. Demo je doplněno zvukovou stereo stopou.

Je velice doporučeno spouštět aplikaci z pevného disku, nikoliv přímo z CD, a to kvůli okolnostem týkajících se zvuku, které byly popsány v příslušné kapitole.

Ovládání

Esc, q - ukončení aplikace

f - režim celé obrazovky (fullscreen)

w - režim okna (window)

Příloha B

Screenshots