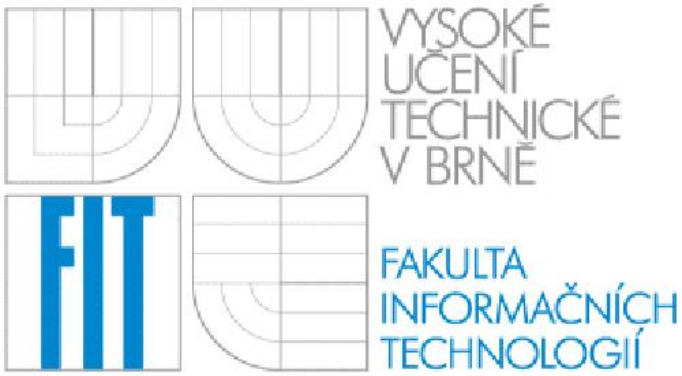
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Simulace a vizualizace explozí

Ročníkový projekt

2006 **Jan Heiník**

Abstrakt

Projekt se zabývá návrhem a tvorbou systému generujícího exploze. Popisuje metody při implementaci a pojednává o možném rozšíření programu. Jedná se o systém pro modelování explozí v reálném čase. Návrh je uzpůsoben pro použití ve větším projektu. Cílem je simulovat explozi připomínající tento děj v reálném světě a porovnat náročnost různých druhů řešení. Úkolem je také objasnit pojem částicový systém a popsat jeho úkol při modelování systému.

Program je vytvořen v jazyce C++ s využitím grafického rozhraní OpenGL a grafické nadstavby Coin3D – Open Inventor.

Klíčová slova

Exploze, OpenGL, Coin3D, Open Inventor, částicový systém, částice

Poděkování

Chci poděkovat svému vedoucímu Ing. Janu Pečivovi za přátelský přístup a pomoc při řešení obtížných problémů.

Abstract

This project is engaged in design and creating system which is generating explosions. Project describes implementation methods and deals with possible program development. It is real-time explosion modelling system. Project is adjusted to be used as a part of bigger project. The main goals are to simulate explosion close to real phenomenon, compare different types of solutions and theirs hardware demands. Another task is to explain what particle system is and describe its function for system modelling.

Project is created in C++ with graphic interface OpenGL and Coin3D - Open Inventor library.

Keywords

Explosion, OpenGL, Coin3D, Open Inventor, particle system, particle

Obsah

Kapitola 1

Úvod

Problematice explozí je v současné době věnováno mnoho pozornosti, jelikož je to jeden z hojně užívaných efektů v mnoha oborech. Nejčastější metodou zobrazování exploze je částicový systém.

Co je to částicový systém a jak je možno pomocí něj namodelovat explozi? Každý částicový systém je složen z velkého množství jednotlivých částic, jež mají své atributy, které mohou přímo nebo nepřímo ovlivňovat jejich chování. Ve většině případů jsou částicemi grafická primitiva jako např. body nebo čáry. Všechny částice dohromady tvoří komplexní dynamický systém. Takto lze vytvořit objekdy libovolných tvarů a rozměrů.

Převážná část práce se týkala zjišťování informací o typech částicových systémů a o možnostech měření rychlosti vykreslování scény. Uživatelský vstup je omezen na minumum nutné pro demonstraci použitých typů explozí. Důvodem je zamýšlené použití ve větším projektu.

V následujících kapitolách se pokusím objasnit, jak tento program pracuje.

V kapitole č. 2 uvádím soubory z nichž je projekt složen je zde uveden implementační jazyk.

V kapitole č. 3 lze najít informace o rozhraní OpenGL a Open Inventor, včetně jejich vzájemného uspořádání

V kapitole č. 4 je popsán částicového systému, co představuje. Rovněž je předveden jeho návrh. V podkapitolách nalezneme informace o různých typech částicových systémů. Dozvíme se zde o pokročilých metodách zobrazování částic. V poslední kapitole je možno najít rozpravu o odvětvích použití programu.

V kapitole č. 5 se popisuje podrobně formát částice, její vlastnosti a parametry. První podkapitola se zabývá popisem implementace pomocí Open Inventoru, v druhé podkapitole jsou popsány point sprites. Dále se dozvíme o dalších parametrech, které je možno u částic použít a způsob ovládání aplikace.

V kapitole č. 6 jsou popsány některé problémy, které se vyskytly při implementaci jednotlivých druhů explozí.

V kapitole č. 7 jsou uvedeny výsledky testování výkonnosti jednotlivých druhů explozí včetně parametrů testovacích sestav a výsledků měření.

V kapitole č. 8 je uvedeno shrnutí naměřených dat.

V kapitole č. 9 najdeme náměty na možné další rozšíření projektu, nápady jak lze dále vylepšit vizuální dojem z explozí a případné další možnosti modifikace systému.

Kapitola 2

Struktura programu

Projekt je napsán ve Visual Studiu .Net v programovacím jazyce C++ [7] s využitím rozhraní OpenGL. Nad OpenGL je ještě použita knihovna GLUT a dále knihovnu Open Inventor, což je knihovna pro tvorbu realtimové 3D grafiky. Poskytuje množinu tříd C++, které skrývají vlastní OpenGL API.

Moduly programu:

- Base.cpp – hlavní modul, ve kterém je zajištěno ovládání programu z klávesnice, vytvoření okna, vykreslování scény, výpočet času, ...
- Coinboard.cpp – modul pro realizaci billboardu
- Sprites.cpp – modul pro realizaci OpenGL 2.0 point sprites. Tento modul byl upraven z tutoriálu Christophe Hermanse [3]
- SoPerfGraph.cpp - obsahují třídy pro zobrazení informace o počtu snímků za sekundu, počtu trojůhelníku ve scéně, počtu viditelných trojúhelníků a rychlosti pohybu kamery. Ty jsou vytvořeny vedoucím mé práce, Ing. Janem Pečivou.

Textury jsou uloženy ve stejném adresáři jako program. Používány jsou textury “explode.png”, “anialp.png” a “explosion1rotalp.png” pro billoardovou explozi a “exp-particle.bmp” pro explozi realizovanou point sprites.

Kapitola 3

OpenGL v programu

3.1 Co je OpenGL?

OpenGL je softwarové rozhraní ke grafické kartě. Příkazy OpenGL se používají ke specifikaci objektů a operací potřebných k vytvoření trojrozměrné aplikace.

OpenGL je hardwarově nezávislé rozhraní. Dnes je dostupné ve většině operačních systémů (Windows, Unix, Linux, Sun, Irix). Tato hardwarová a platformová nezávislost je umožněna díky nepoužívání příkazů pro práci s okny nebo pro zpracování uživatelského vstupu.

Pomocí OpenGL lze vykreslit model pouze použitím základních příkazů vykreslujícím body, čáry a polygony.

Při zobrazení pomocí této metody jsou na vstupu objekty 3D scény, které jsou převedeny na 2D obrázek pomocí vykreslovacího řetězce (tzv. rendering pipeline viz Obrázek 3.1). Tento obrázek je uložen v tzv. paměti snímku (framebufferu viz Obrázek 3.1), odkud se již zobrazuje na monitor. Zobrazení snímku viz Obrázek 3.1.

OpenGL je tzv. stavový stroj. Pokud nastavíme v programu nějakou vlastnost, pak platí do té doby, dokud není nastavena vlastnost jiná.

Výhodou OpenGL je, že vývojář na její použití nepotřebuje žádnou licenci.

3.2 Co je Open Inventor?

Open Inventor [4] je velmi populární knihovna pro tvorbu realtimové 3D grafiky, tedy i her.

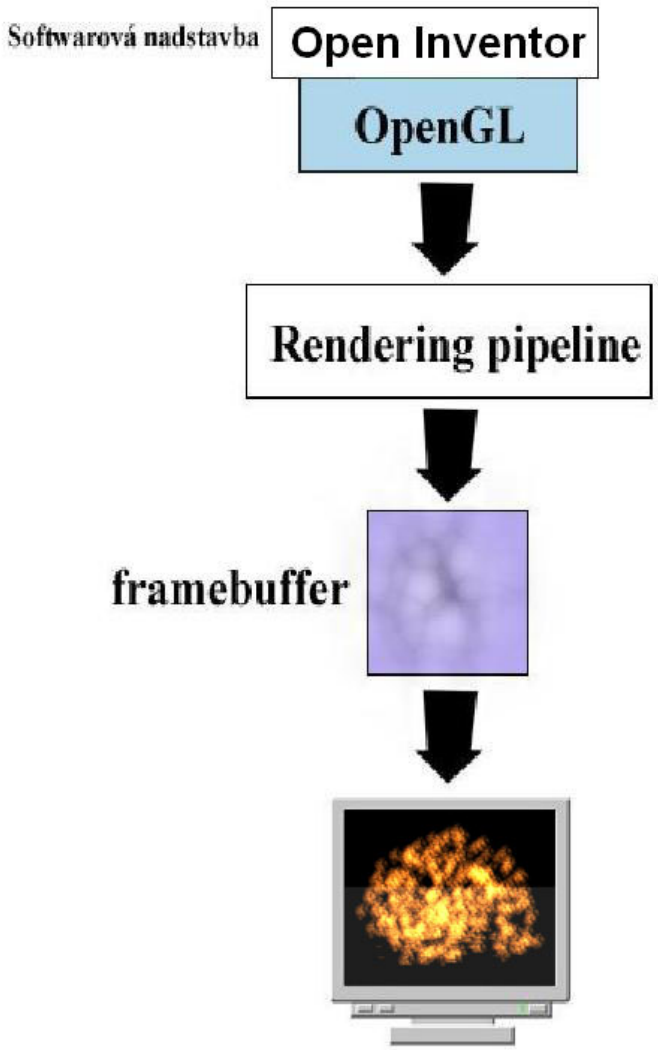
Programátorovi poskytuje rozsáhlou množinu C++ tříd, které skrývají před programátorem vlastní OpenGL API a posunují ho na mnohem vyšší úroveň. Tak může programátor mnohem rychleji vyvinout to, co potřebuje.

Navíc, aplikace napsané v Open Inventoru jsou obyčejně rychlejší než ty přímo psané v OpenGL.

Je rovněž platformově nezávislé, stejně jako OpenGL.

V tomto příkladu vytvoříme minimální aplikaci zobrazující červený kužel osvětlený jedním světlem. Graf scény, zobrazující červený kužel osvětlený jedním světlem, vidíme na Obrázku 3.2.

Kořen grafu tvoří objekt typu SoSeparator. Když se podíváme pod separátor, zjistíme, že má čtyři syny: kamera, světlo, materiál a kužel. Kamera (SoCamera) je nod, který určuje umístění pozorovatele a některé další atributy pohledu do scény. Světlo (SoLight) osvěcuje scénu bílým světlem. Materiál (SoMaterial) udává optické vlastnosti kužele, jednoduše řečeno - udává jeho barvu. Posledním nodem je pak vlastní kužel, což je nod specifikující geometrii tělesa. Výsledkem takového grafu je pak scéna znázorněná na obrázku 3.3.



Obrázek 3.1: Vykreslení pomocí OpenGL a knihovny Open Inventor

3.3 Využití technik OpenGL

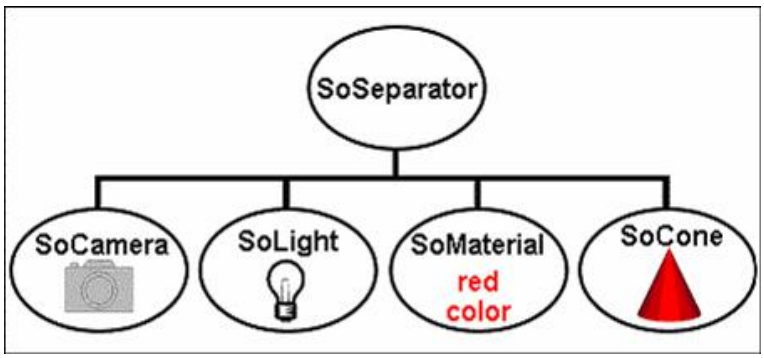
Procesu vytváření obrazu scény ve framebufferu říkáme rendering [6]. Data scény jsou pak často výsledkem simulace nějakého modelu viz obrázek 3.1.

Pro vylepšení vzhledu částic systém využívá **texturování**. Je to technika nanášení barevného obrázku na grafické těleso. Texturování je využito pro jednotlivé částice.

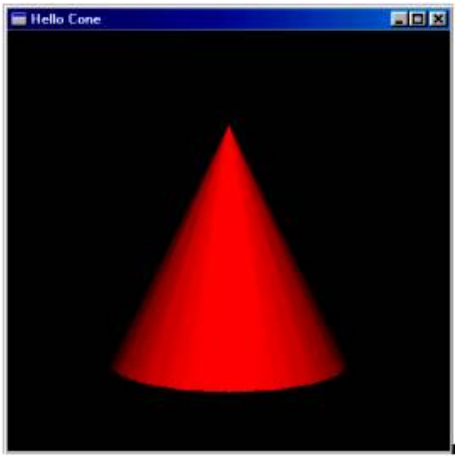
Aby na scéně bylo vůbec něco vidět, je scéna osvícena. Světlo přidá tělesu lesk a osvítí celou scénu. Je použito materiálové nastavení, to definuje vlastnosti povrchu tělesa.

Hlavní stavební jednotkou tělesa je trojúhelník, případně více trojúhelníku seskupených do čtverce.

Jak jsem se již dříve zmínil, k těmto možnostem OpenGL, přistupuji skrze knihovnu Open Inventor.



Obrázek 3.2: Graf jednoduché scény



Obrázek 3.3: Jednoduchá scéna

Kapitola 4

Částicový systém

4.1 Návrh

Cílem projektu je porovnat výkonnost jednotlivých druhů explozí, proto není realizována možnost více explozí, či více druhů explozí zároveň. Bylo by možno vytvořit strukturu, pro odlišení jednotlivých explozí a jejich inicializaci. Vypovídací hodnota informací získaných měřením, by však byla minimální.

Nejvhodnější pro modelování exploze je tzv. systém volných částic. Na částice působí pouze globální síly. V tomto případě jsou to síly třecí a síla gravitační, která způsobuje pád částic.

V programu je možno úpravou parametrů dosáhnout dalších efektů, jako působení větru, či "dohoření" částice na zemi. Počet částic velmi ovlivňuje výslednou rychlost vykreslování exploze. Podle nastavení lze změnit jak daleko částice doletí, či dopadnou a také jak silná exploze bude.

Při návrhu jsem se inspiroval tutoriály na stránkách Nehe.cz [5], přestože v tomto případě byl částicový systém použit pro demonstraci proudu částic. Všechny tutoriály na těchto stránkách jsou pro OpenGL a v tomto projektu je využíván Open Inventor, proto bylo třeba transformovat všechny informace tak, aby vyhovovaly definici v *coinboard.cpp*.

4.2 Druhy systémů

Kromě systému volných částic, zde popisovaného, existují ještě systémy, kde kromě globálních sil působí na částice také síly lokální. Složitost takovýchto systémů je však větší než systému mnou použitého. Mezi lokální síly patří např. vzájemná gravitační síla, elektrostatické síly a také vazby elastické. Takovéto systémy se používají pro modelování např. vodní hladiny, pružných membrán atd.

4.3 Aplikace

Částicové systémy lze použít pro vyobrazení modelů reálného světa, jejichž tvar a chování lze těžko popsat jinými vizualizačními metodami. Jejich využití tudíž není pouze pro zobrazení explozí [2], ale rovněž pro simulaci jiných reálných jevů, např. vody, kouře, ohně, sněhu, mlhy a dalších. Částicové systémy jsou používány i některými modelovacími programy (např. 3D studio)

Systém určený pro exploze je zpravidla využíván v těchto případech:

- herní aplikace
- filmové efekty

Využití však můžeme nalézt v řadě jiných aplikací. Uplatní se například jako šetřič obrazovky, v demu různorodých programů a také jako pluginy pro audio a video přehrávače.

Kapitola 5

Vlastnosti částice

5.1 Moje částice

Jak již bylo řečeno, částice jsou volné, neexistují mezi nimi žádné vazby. Je nastaveno, že částice, které jsou pod plochou (představuje povrch), nejsou vykreslovány, ale “zhasnou”.

Poziční parametry částice jsou definovány ve třídě *Particle*, nastavení místa výbuchu a rychlosti částic obstarává funkce *reset_particles*. Tato třída také realizuje částečnou obsluhu druhu exploze.

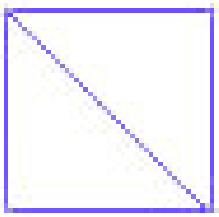
Základem je třída *billboard* z modulu `coinboard.h`. Billboard je plocha tvořená trojúhelníky, které jsou neustále orientovány směrem ke kameře. Je definován 3D vektorem

```
billboard->numVertices.set1Value(i,n)
```

kde parametry `i` a `n` určují index billboardu a počet trojúhelníků, z kolika bude složen.

Dále je nutno nastavit pro každý takovýto billboard jeho orientaci. určuje nám to třída *coord*, kde je na základě indexu přiřazena jednotlivým bodům billboardu pozice v prostoru kvůli orientaci pro nanesení textury. Pro správné zobrazení se dle parametru v `numVertices` tvoří čtveřice bodů, tudíž jeden billboard je tvořen z bodů s indexem *coord* 0-3, 4-7 atd.

V našem případě, je použito na vytvoření billboardu dvou trojúhelníků, jak vidíme na obrázku 5.1.



Obrázek 5.1: Triangle strip

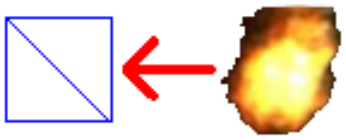
Pozice billboardu je určena parametrem *position*:

```
billboard->position.set1Value(i, SbVec3f(x, y, z))
```

kde opět `i` označuje index částice, pro kterou jsou nastaveny souřadnice `x`, `y` a `z`, kde bude billboard vykreslen.

Na billboard je dále nanesena průhledná textura, což je zajištěno zapnutím alpha kanálu [6]. V aplikaci lze přepnout, zda se má textura animovat, nebo má být statická. Animace má za následek zlepšení vzhledu exploze, které však jde na úkor rychlosti vykreslování celé scény, protože se musí počítat texturovací souřadnice.

Princim texturování je naznačen na obrázku 5.2.



Obrázek 5.2: Nanášení textury

5.2 OpenGL Point Sprites

Když chceme nakreslit "klasický" obdélník, předáme OpenGL souřadnice čtyř bodů, texturovací koordináty a normálové vektory. Na rozdíl od toho point sprite vyžaduje pouze x, y, z souřadnice a nic jiného. Grafická karta vykreslí kolem těchto souřadnic obdélník, který bude vždy orientován k obrazovce. Jejich velká nevýhoda spočívá v implementaci, existují pouze jako rozšíření (GL_NV_point_sprite), takže se může stát, že je grafická karta nebude podporovat.

Point Sprites jsou tedy body, okolo kterých je vytvořen malý čtverec, na který se mapuje textura. V dnešní době jsou často používány pro částicové systémy ve hrách pro zobrazování nejrůznějších objektů a efektů (exploze, dým, atmosférické jevy atd.)

Vykreslování pomocí point sprites by mělo být rychlejší, než billboard, protože pro jeden billboard je nutno počítat se čtyřmi body, kdežto v tomto případě je jeden bod ekvivalentem billboardu.

Při realizaci exploze pomocí point sprites jsem vycházel z tutoriálu Christopha Hermanse [3]. Modifikoval jsem však vykreslování, aby exploze nevypadala jako krychle. Jde o přidání testu, zda částice po vygenerování náhodné rychlosti nemají tuto hodnotu příliš velkou:

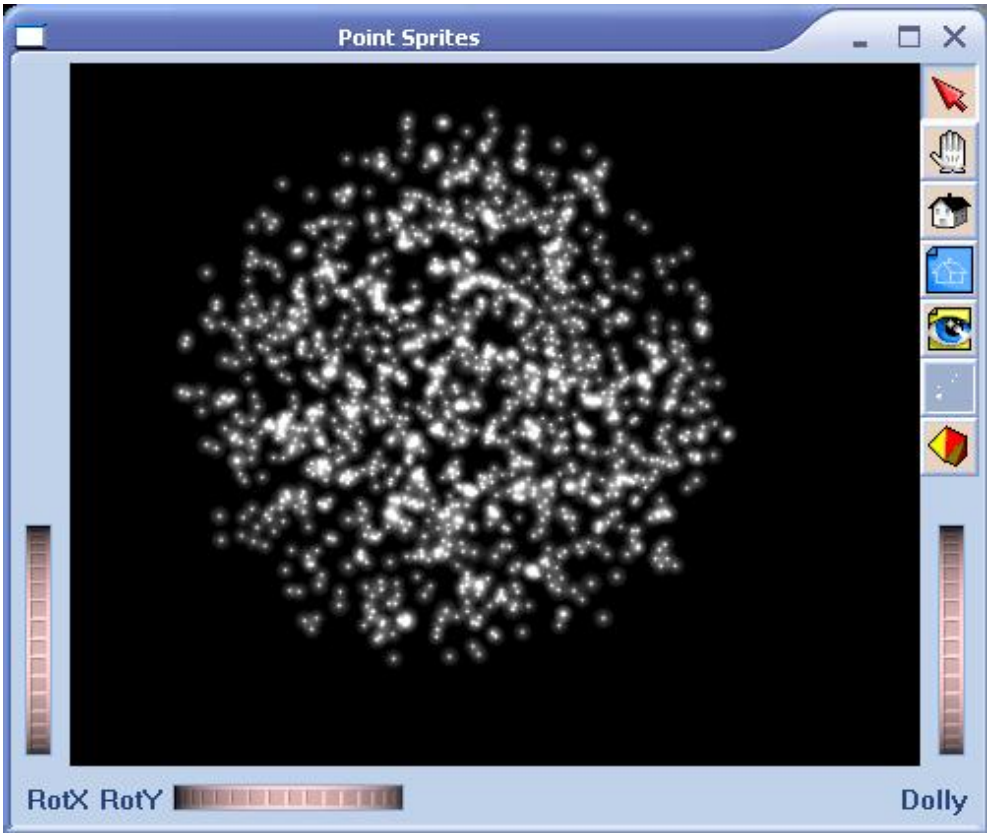
```
do {
  sp_x=((rand()%1000)-500);
  sp_y=((rand()%1000)-500);
  sp_z=((rand()%1000)-500);

  test = sqrt(sp_x*sp_x + sp_y*sp_y + sp_z*sp_z);
}while (test >= 500);
```

Stejný test je použit také v první části, kde jsem původně tento problém řešil již v první části. Exploze sice lépe vypadá, ale kvůli dalšímu výpočtu dojde ke zpomalení vykreslování.

Ukázku exploze vytvořené pomocí point sprites nalezneme na obrázku 5.3.

5.3 Částice v pohybu

Pro animaci exploze je nutné, aby každá částice měla přiřazen nějaký směr letu. Dalším důležitým parametrem je rychlost letu částice. První vlastnost ovlivňuje směr, kam částice poletí, druhá určuje,



Obrázek 5.3: Exploze vytvořená point sprites

kam až doletí. Oba parametry jsou vygenerovány na základě náhodného generování hodnot na počátku každého cyklu exploze ve funkci *reset_particles*. Spolu s rychlostí se každé částici nastaví její pozice na souřadnice (0, 0, 0), jež jsou výchozím bodem pro explozi a dále případné další parametry. Těmi lze nastavit velikost gravitace, či odpor prostředí. Rychlost a směr jsou tedy určeny vektorem, jehož délka určuje samotnou rychlost.

Délka vektoru: $d = \sqrt{x^2 + y^2 + z^2}$

Tvar exploze tvoří kouli, nebo polokouli v případě zapnutí neúplného výbuchu. Vektor nejdále dolétající částice proto musí ležet na povrchu takovéto koule. Vektory ostatních částic leží uvnitř tohoto prostoru a pomalejší částice vyplňují vnitřní oblast koule.

Parametrická rovnice takovéto koule je: $x^2 + y^2 + z^2 = r^2$.

Rychlost částic a tím i velikost exploze lze korigovat pomocí parametru *slow* ve funkci *reset_particles*.

Ukázka způsobu jakým jsou jednotlivým částicím přiřazeny parametry:

```
for (i=1; i<=MaxParticles; i++){
  do {
    sp_x = ((rand()%100) - 50);
    sp_y = ((rand()%100) - 50);
    sp_z = ((rand()%100) - 50);
    // sphere look of explosion
    test = sqrt(sp_x*sp_x + sp_y*sp_y + sp_z*sp_z);
```

```
  }while (test >= 50);

  // particle position
  a_particle[i].pos_x = 0.f;
  a_particle[i].pos_y = 0.f;
  a_particle[i].pos_z = 0.f;

  // particle velocity
  int slow = 400;
  a_particle[i].sp_x = sp_x/slow;
  a_particle[i].sp_y = sp_y/slow;
  a_particle[i].sp_z = sp_z/slow;

  // particle acceleration
  a_particle[i].xi = 0.f;
  a_particle[i].yi = 3.0f; // gravity
  a_particle[i].zi = 0.f;

  // if one -> just 1 central growing billboard
  if (one == true) billboard->numVertices.set1Value(i,0);
  else {
  billboard->numVertices.set1Value(i,4);
    billboard->position.set1Value(i, SbVec3f(0.f, 0.f, 0.f));
  }
}
```

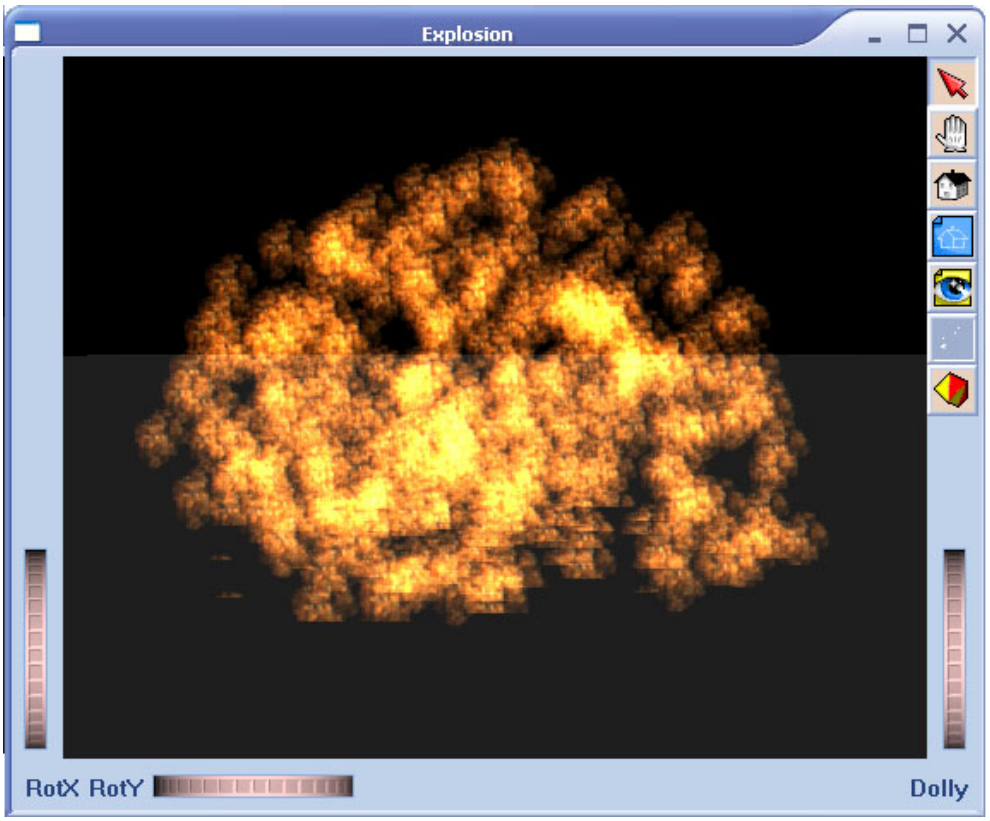
Exploze vygenerovaná uvedeným způsobem za pomoci billboardu z modulu *coinboard* je na obrázku 5.4

5.4 Ovládání

Jelikož byl tento projekt vytvořen jako součást většího projektu, není v něm zabudováno příliš možností interakce s uživatelem. Většina nastavení lze měnit pouze uvnitř programu. Částečně lze zobrazení scény ovládat tlačítky na boční liště okna aplikace. Jde především o pohyb s vykreslenou scénou – rotace, přiblížení, návrat do výchozího zobrazeni. Pro interakci z klávesnice je implementováno přepínání druhu exploze je řešeno strukturou *Klavesa*, převzatou z projektu Tank Hunters, který vytvořil Daniel Výchlopeň [8]. Tato struktura byla mírně upravena aby nebylo nutno klávesu po celou dobu držet stisknutou.

Klávesy a alternativní ovládání:

- „1“ – částicová exploze se statickou texturou
- „2“ – částicová exploze s animovanou texturou
- „3“ – exploze realizovaná na 1 billboardu s animovanou texturou
- „5“ – částicová exploze pomocí Point Sprites
- „s“ – přiblížení ke zvolenému bodu ve scéně
- „q“ – ukončení aplikace



Obrázek 5.4: Exploze vygenerovaná billboardy

Kapitola 6

Problémy při implementaci

6.1 Open Inventor

Při implementaci jsem se několikrát setkal se závažnými problémy při vykreslování ať už částic, celé exploze, nebo textury.

Jednalo se o špatné zobrazování textury na vytvořený billboard. Místo textury byl zobrazen pouze deformovaný barevný úsek na okraji plochy, když byl vykreslován pouze jeden centrální, zvětšující se, billboard, i přesto že texturovací souřadnice byly nastaveny korektně. Na vině je chyba v *coinboard.cpp* a bylo nutno přesunout obsluhu zvětšování plochy z *main()* programu do funkce *time_step*.

Vyskytl se také problém ve finální fázi měření výkonnosti. Open Inventor obsahuje třídu *SoSeparator*, do které se vkládá kořen scény *root*, pod nímž se vytváří strom zobrazovaných objektů. Pro lepší separaci je dále možno použít třídy *SoSwitch*, která dále umožňuje oddělení jednotlivých prvků ve scéně. Pro tento účel jsem si vytvořil *mySwitch* a přidal ji do kořene scény.

```
mySwitch = new SoSwitch;
root->addChild(mySwitch);
```

Jednotlivé stromy napojené na scénu přes *mySwitch* je možno zobrazit všechny, nebo vypnout a zobrazit jen některé. Provádí se to nastavením *whichChild*. Vždy musí být nastaven buď *SO_SWITCH_ALL* nebo jeden ze stromů.

```
mySwitch->whichChild = SO_SWITCH_ALL; //zobrazí všechny stromy

mySwitch->whichChild = 1;  // zobrazí pouze druhý strom přidaný do mySwitch
```

Při aktivním pouze jednom, však i vypnuté stromy scény konzumují výkon. V důsledku čehož je pak pomalejší vykreslování zvoleného stromu a tím i zkreslené ukazatele rychlosti zvolené metody zobrazování.

6.2 Point Sprites

Při realizaci tohoto druhu částicového systému se nevyskytlo tolik problému jako v případě Open Inventoru. Důvodem bylo, že jsem vycházel z funkčního řešení pana Hermanse [3]. Některé problémy již byly řešeny v minulosti ať už mnou v první části projektu, nebo bylo možno vyhledat tipy v četných diskusích na internetu. OpenGL je mnohem více využíváno, než Open Inventor, takže

pro něj existuje více tutoriálů a také dokumentace jednotlivých funkcí je mnohem obsáhlejší a detailnější.

Point Sprites jsou v samostatném modulu *sprites.cpp*. Spuštění z hlavního modulu je řešeno callback funkcí. Kvůli výše zmíněné konzumaci výkonu i vypnutých stromů je vypnuta animace texturování a jsou vypnuty částice před přepnutím z billboardu na point sprites.

```
SoCallback *cb_sprites = new SoCallback;
cb_sprites->setCallback(&draw_sprites);
mySwitch->addChild(cb_sprites);

// vypnutí billboardů
element(tex_x, tex_y, FALSE);
MaxParticles = 0;
reset_particles(true);
```

Dále se vyskytl problém, že bylo vidět pouze prázdné okno bez jakýchkoliv vykreslených objektů. Bylo nastaveno míchání barev textury a nastavena černá barva materiálu, tudíž byly zobrazeny tmavé částice na tmavém pozadí.

Kapitola 7

Porovnání metod vykreslování

7.1 Testovací sestavy

Porovnání výkonu bylo provedeno na dvou počítačích z důvodu porovnání rychlosti vykreslování hw akcelerovaných a hw neakcelerovaných Point Sprites. Jako měřící nástroj byla použit modul SoPerfGraph.cpp, který je v poněkud větší míře použit také v projektu Petra Bulíčka [1]. Podstatou měření je zjišťění počtu *FPS*, které je schopna aplikace vykreslit.

Konfigurace počítačů je uvedena v tabulce:

PC	**6600GT**	**Ti4200**
CPU	Opteron 144 (1,8@2,6 GHz)	Athlon XP 1700+ (1,47 GHz)
GPU	GeForce 6600GT	GeForce 4200 Titanium
RAM	1 GB	512 MB
OS	Win XP SP2	Win XP SP2

7.2 FPS

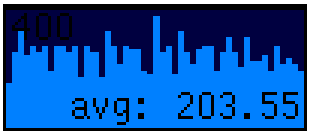
FPS, nebo-li Frames Per Secon je hodnota udávající počet zobrazených snímků za jednu sekundu. Vykreslování grafických aplikací totiž znamená neustálé generování scény na obrazovku monitoru. V závislosti na počtu fps pak vypadá i animace celé scény. Systém je tvořen tak, že rychlost explozí je na různě rychlých počítačích různá. Proto vliv FPS na scénu s explozemi je jiný, než vliv na běžné grafické aplikace, ve kterých se sice objekty pohybují stále stejně rychle, ovšem za cenu pomalého překreslování, kazícího dojem animace při FPS menším než 30. Implementovaný systém je závislý na hodnotě FPS. Při nízkém FPS je exploze pomalejší a není tak efektní.

7.3 Měření

Samotná podstata měření výkonnosti použitých metod spočívala v přidaní modulu *SoPerfGraph* do projektu. Tento modul je schopen zobrazit počet FPS, počet trojúhelníků ve scéně atd. Cílem projektu bylo vyhodnotit počet zobrazených snímků v závislosti na počtu částic ve scéně.

Hodnoty jsem odečítal z grafu zobrazeného v okně zobrazujícího explozi, viz obrázek 7.1

Získané hodnoty se nacházejí v tabulkách 7.1 a 7.2. Před samotným měřením bylo třeba vypnout vertikální synchronizaci vykreslování, protože v tomto případě nikdy nebylo vykresleno více než 90 snímku, což korespondovalo s obnovovací frekvencí nastavenou na grafické kartě.



Obrázek 7.1: Graf zobrazující FPS

	Použitá textura			Point sprites	
částic	**na 1 ploše**	**statická**	**animovaná**	**6600GT**	**Ti4200**
100	75	90	67	1260	300
200	75	88	58	1220	300
300	75	72	50	1200	295
400	75	54	40	1157	290
500	74	45	37	1150	290
600	74	42	33	1140	290
700	70	35	28	1140	285
800	68	32	26	1130	285
900	66	29	24	1120	280
1000	65	25	22	1100	280
1100	64	23	20	1100	280
1200	63	21	19	1060	280
1300	62	20	18	1040	275
1400	61	19	17	1000	270
1500	60	18	16	980	270
2000	57	14	12	950	260
3000	50	9	8	850	250

Tabulka 7.1: FPS pro všechny druhy vykreslování

Z uvedených tabulek je zřejmé, že point sprites se jeví jako místy i 44x rychlejší způsob zobrazení částicového systému využívaného pro explozi. Důvodů může být hned několik, uvedl bych zde tyto:

- nižší počet grafických primitiv na částici
- hw akcelerace point sprites na novějších kartách
- špatný návrh billboardu v modulu *coinboard.cpp*

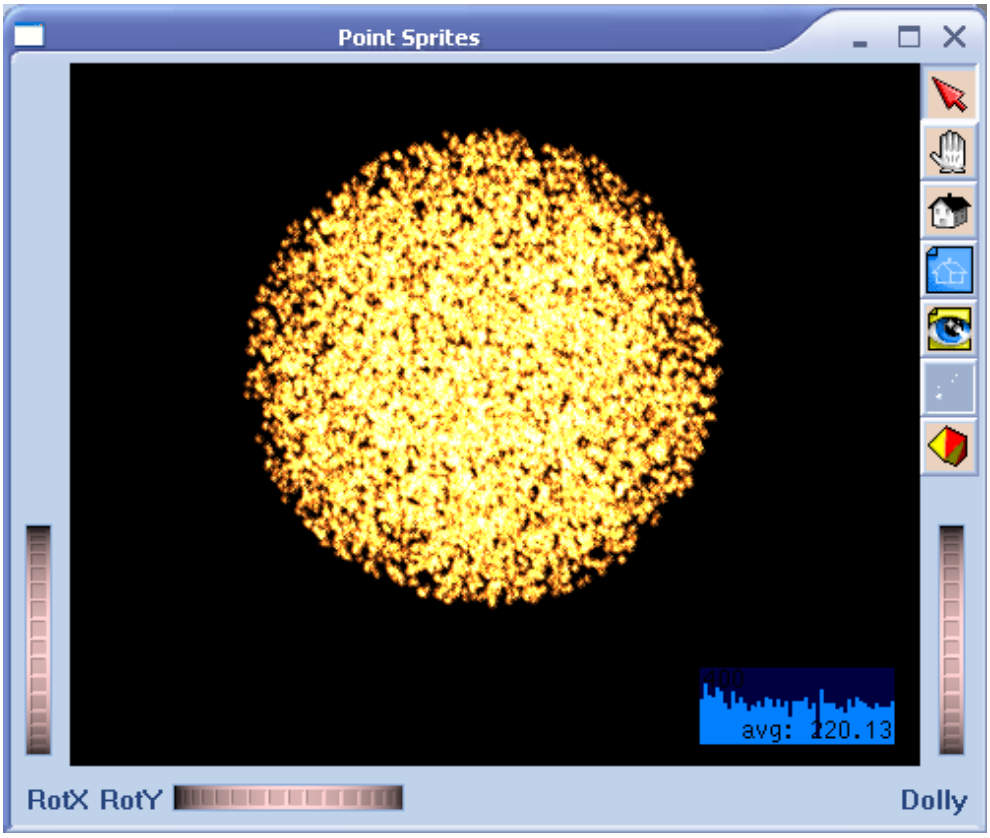
Ač je Open Inventor nadstavbou nad OpenGL a zjednodušuje práci, protože není třeba definovat všechny parametry scény na úrovni funkcí, avšak jak je zřejmé z měření, není příliš optimální pro výsledný výkon aplikace. Pro každou částici je nutno mít 4 body, takže paměťové i výpočetní nároky jsou vyšší.

Point Sprites se jeví jako velmi výkonná a slibná varianta pro vykreslování efektů. Je dále umocněna zahrnutím hardwarové akcelerace tohoto druhu vykreslování výrobci grafických karet, takže při výpočtu není zatěžován procesor. Při vykreslování šesti tisíc částic je stále hodnota FPS velmi vysoká, jak můžeme vidět na obrázku 7.2.

Pro lepší demonstraci rozdílů v metodách jsou přiloženy grafy pro explozi vykreslenou pomocí Open Inventoru 7.3. FPS exploze vykreslené point sprites vidíme na grafu 7.4, který ukazuje průběh

Point Sprites		
částic	**6600GT**	**Ti4200**
1000	1100	280
2000	950	260
3000	850	250
4000	730	235
5000	680	225
6000	620	215
8000	557	190
10000	540	180
15000	430	140
20000	380	125
30000	280	100
40000	225	80
50000	160	70
60000	140	60
70000	130	50
80000	113	45
90000	100	40
100000	90	35
120000	80	30
150000	55	25
175000	50	20
200000	38	20
225000	35	18
250000	30	15

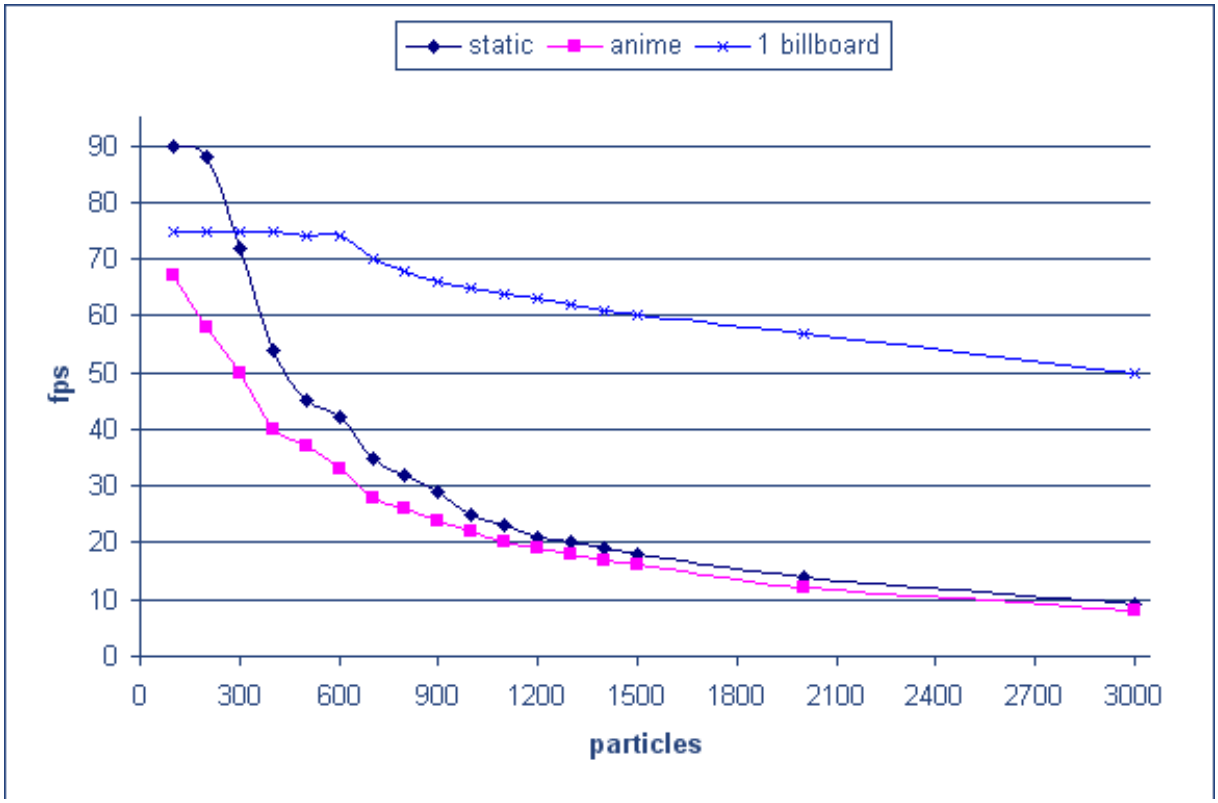
Tabulka 7.2: FPS pro vyšší počty částic - point sprites



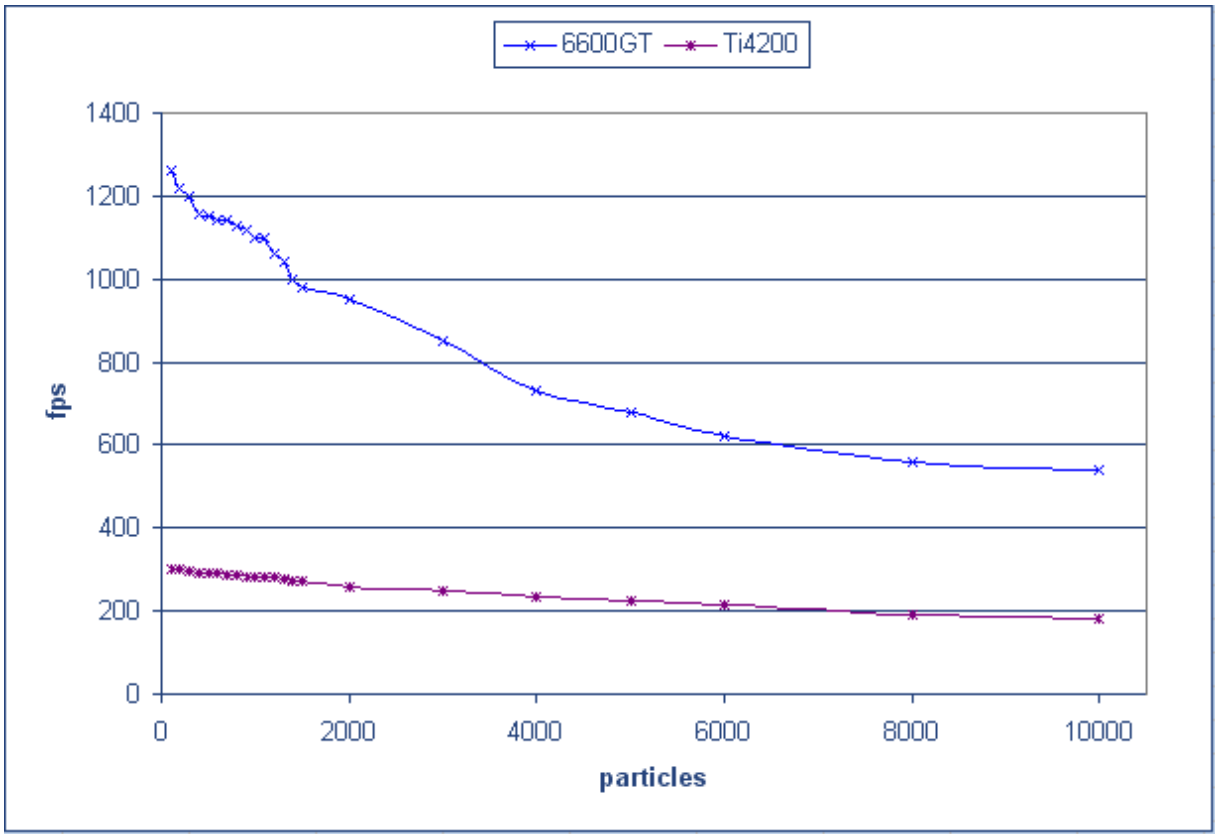
Obrázek 7.2: Exploze 6000 point sprites

pro počty částic do deseti tisíc a graf 7.5 na kterém jsou výsledky pro čtyři tisíce až dvě stě padesát tisíc částic.

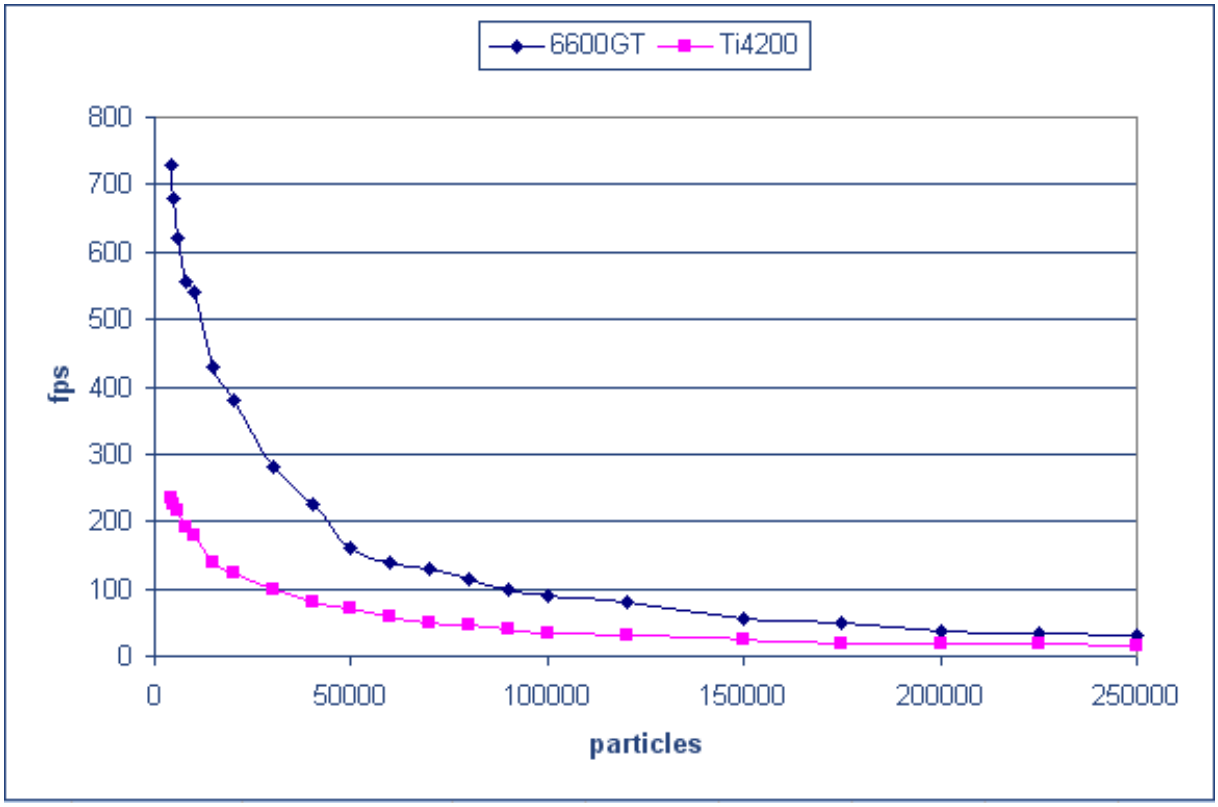
Z grafů je jasně patrné, že závislost počtu vykreslených částic na FPS je velmi podobná hyperbolické funkci. Přitom nezávisí zda se jedná o vykreslování exploze přes billboard z *coinboard.cpp* nebo point sprites. Průběh grafu je vyždy podobný.



Obrázek 7.3: Graf FPS pro billboard



Obrázek 7.4: Graf FPS pro Point Sprites do 10000



Obrázek 7.5: Graf FPS pro Point Sprites do 250000

Kapitola 8

Zhodnocení výsledků

Pokoušel jsem se vytvořit exploze blízké reálnému světu a změřit jejich hardwarovou náročnost. Výsledný efekt však není příliš dokonalý, z důvodu absence dalších projevů a vlivů provázejících skutečnou explozi. Zejména se jedná o kouř, kolize různě hmotných částic a podobně.

Teoretické vlastnosti exploze vytvořený systém obsahuje. Díky průhlednosti mohou mít částice vzhled blízký reálnému. Přesto je zde spousta prostoru pro další vylepšení. Především první část, kde je použito coinboardu není příliš svižná, dalo by se to vyřešit optimalizací výpočtu, či jiného navržení systému.

Reálnější dojem modelovaného jevu získáme použitím většího počtu částic v systému, avšak z měření vyplývá, že použitelné je zejména řešení point sprites, kde můžeme i na slabším hardwaru použít i více než třicet tisíc částic a stále dosáhneme velmi solidního počtu FPS, jak vyplývá z tabulky 7.2, tabulky 7.1 a grafů 7.3, 7.4 a 7.5.

Kapitola 9

Plány do budoucna

Projekt má výhodu, že je možné jej v mnohém rozšířit a doplnit. V systému by mohlo jít vylepšit rychlost vykreslování. Jedná se především o část s Open Inventor řešením, kdy by bylo zřejmě výhodnější použit vlastní billboard, nebo opustit Open Inventor a implementovat systém čistě pod OpenGL. Jako velmi zajimavé se jeví použití point sprites. V současné době je totiž nutno si vybrat mezi počtem částic a rychlostí exploze.

Vhodné by mohlo být implementovat využití více explozí současně - v systému by se vyskytovaly již letící částice z jiných výbuchů.

Velmi zajímavým by mohlo být přidání zdroje světla do epicentra výbuchu. Dosáhlo by se tak dalšího vizuálního zlepšení nejen výbuchu, ale i celé scény, kdy by byly případné objekty vyskytující se v oblasti exploze osvětleny tímto zdrojem dále ovlivněny.

Dále by bylo možno implementovat interakci letících částic s jinými prvky scény, kdy by bylo možno nastavit další body, ve kterých by byla odstartována nová exploze, pokud by se takovýto bod vyskytl v dráze letící částice. Výbuch by tak vytvořil lavinový efekt.

Implementace kolizí pevných těles s explozí by neměla vliv na vzhled exploze, ale na chování částic. Těleso by bránilo dalšímu pohybu částic, jejichž trasa by procházela přes oblast zabranou tělesem. Při detekci kolize by se částice odstranila ze scény, jiná možnost by bylo nastavit částici opačnou rychlost, což by vedlo k jejímu odrazu od tělesa.

Kapitola 10

Závěr

Na základě požadavků byly implementovány čtyři druhy explozí, na kterých bylo provedeno měření rychlosti vykreslování.

Použití projektu je popsáno v kapitole č. 5. Další možnosti rozšíření na diplomový projekt je uvedeno v kapitole č. 9. Aplikace point sprites pro vykreslování efektů vyskytující se v reálném světě je zřejmá z měření, jehož výsledky jsou v kapitole č. 8.

Při vytváření dokumentace jsem se inspiroval z předešlých prací mých kolegů.

Zadání projektu bylo z mého pohledu úspěšně splněno.

Literatura

[1] Petr Bulíček. Ročníkový projekt - asteroid generator, vut 2005. http://merlin.fit.vutbr.cz/wiki/index.php?title=Inventor_Projects_2005.

[2] Ivo Chvojka. Ročníkový projekt. VUT 2004.

[3] Christophe Hermans. Opengl 2.0 point sprites tutorial. http://data.plok.be/ExNihilo/pages/Tutorialpointsprite.htm.

[4] WWW stránky. Open inventor tutoriály. http://www.root.cz/clanky/open-inventor/.

[5] WWW stránky. Opengl tutoriály. http://nehe.opengl.cz.

[6] WWW stránky. Přednášky a cvičení z předmětu počítačová grafika. http://www.fit.vutbr.cz/study/courses/index.php?id=366.

[7] M. Virius. *Od C k C++*. Nakladatelství Kopp, 2000. ISBN 80-7232-110-2.

[8] Daniel Výchlopeň. Ročníkový projekt - tank hunters, vut 2005. http://merlin.fit.vutbr.cz/wiki/index.php?title=Inventor_Projects_2005.

Kapitola 11

Přílohy

CD-ROM se zdrojovými kódy a dokumentací.

Kapitola 12

Seznam obrázků