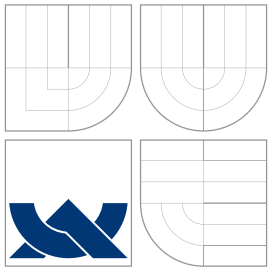
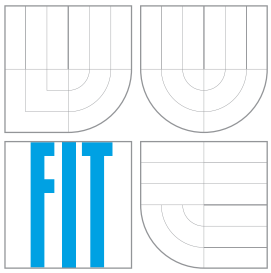


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČÁSTICOVÉ SYSTÉMY

PARTICLE SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN SYNÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ROMAN JURÁNEK

BRNO 2009

Abstrakt

Obsahem této práce je návrh a implementace rozšíření částicového systému Particle Systems API. Rozšíření poskytuje prostředek, kterým lze rozdělit 3D model definovaný pomocí okrajové reprezentace na části pomocí rovin a prostřednictvím částicového systému je vykreslit. Protože jsou částice reprezentovány jako hmotné body, existující systém je rozšířen také o možnost jejich objemové reprezentace. Dále je vytvořena jednoduchá hra demonstrující toto rozšíření.

Abstract

The subject matter of this work is a concept and implementation of an extension of Particle Systems API. The extension provides a means by which it is possible to split a 3D model defined by its boundary representation into pieces with a geometric plane and render them using the particle system being extended. Because all particles are represented as point mass, the existing particle system is also extended to provide a way of representing its particles by their volume. Furthermore, a simple game is created to demonstrate the extension.

Klíčová slova

Částice, částicové systémy, počítačová grafika, fyzikální simulace, počítačové hry, vizualizace

Keywords

Particles, Particle Systems, Computer Graphics, Simulation of Physics, Computer Games, Visualization

Citace

Jan Synáček: Částicové systémy, bakalářská práce, Brno, FIT VUT v Brně, 2009

Částicové systémy

Prohlášení

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

.....
Jan Synáček
14. května 2009

Poděkování

Na tomto místě bych chtěl poděkovat Ing. Romanu Juránkovi za cenné rady, odbornou pomoc a věcné připomínky při vypracovávání této práce.

Obsah

Kapitola 1

Úvod

Částicové systémy již mají svoji historii. Byly a jsou využívány pro různé účely ve fyzikálních simulacích, vizualizaci ve vědeckém prostředí a především v počítačové grafice. Jejich použití je vhodné hlavně tam, kde by bylo jinak obtížné používání konvenčních vykreslovacích technik. Mezi běžné využití tedy mužeme zařadit například vytváření efektů výbuchů, kouře, tekoucí vody, padajících listů, různých druhů počasí, atd. Jako méně typický způsob využití můžeme uvažovat třeba simulace hejn zvířat a nebo generování bublinek v pivě.

Tato práce se bude zabývat využitím částicových systémů v počítačových hrách za účelem vytvoření kvalitních grafických efektů. Bude vytvořeno rozšírení již existujícího systému The Particle Systems API, který poskytuje vhodné prostředí. Je dostatečně jednoduché a zároveň velice výkonné.

Jako rozšírení bylo zvoleno velmi jednoduché rozbití 3D modelu na menší kousky a jejich následná simulace a vykreslování jako částicového systému. Vzniklé kusy umožní svým vzhledem a realistickým vzhledem přispět k vyšší úrovni realističnosti a také k lepšímu grafickému požitku v počítačových hrách.

Takto vytvořené rozšírení nebude plně respektovat fyzikální zákony a výpočty nebudou zcela přesné.

Druhá kapitola se bude zabývat částicovými systémy a jejich využitím. Ve třetí kapitole bude možné nalézt principy zobraznovacích technik ve 3D scéně, základní problematika transformací a projekce. Bude také nastíněna funkčnost grafické knihovny OpenGL. Čtvrtá kapitola popisuje vlastní návrh a řešení práce.

Kapitola 2

Částicové systémy

V této kapitole bude popsán význam, činnost a využití částic a částicových systémů jako celku. Dále budou popsány a ozřejmeny jednotlivé fáze, kterými činnost v částicovém systému prochází.

2.1 Částicový systém

Částice
Částice jsou objekty reprezentované různými fyzikálními veličinami. Za základní vlastnosti částic je považována pozice, rychlost, zrychlení a hmotnost. Další důležitou vlastností je reakce na vnější síly. Jsou ovšem reprezentovány hmotnými body, to znamená, že nenesou žádnou informaci o objemu.

Částicový systém
Částicový systém je způsob modelování fuzzy objektů, jako jsou například voda, oheň, výbuchy, a jiné [9]. Tyto objekty nemají definovaný povrch (jsou to pouze hmotné body). Od „normálních“ objektů se liší v několika bodech. Fuzzy objekt, tedy ten, který je tvořen částicemi, není reprezentován množinou polygonů určujících jeho povrch, ale skupinou částic vymezujících jeho objem. Částicový systém není statická entita, nové částice jsou dynamicky vytvářeny, staré zanikají, a všechny jako celek formují a pobyhují celým systémem. Částicový systém není deterministický, jeho tvar není kompletně definován. Částicové systémy jsou příkladem stochastického procedurálního modelování.

Systém je obvykle složen s více podsystémů, tj. skupin částic majících odlišné vlastnosti, například působicí síly, textury použité pro vykreslování nebo čas, po který daný podsystém přežívá v celém systému a také, po který je renderován. Životní cyklus částicových systémů lze rozdělit do několika částí, které budou popsány v následujících sekcích.

Simulační fáze
Během simulační fáze je rozhodováno o celém životě všech částic v systému. Jednotlivé Částice jsou vytvářeny na základě rychlosti jejich vytváření (spawning rate) a intervaly mezi jednotlivými aktualizacemi. Dále se také aktualizují informace o pozicích, rychlostech a zrychleních, barvách, apod.

Existují dva způsoby, jakými částicový systém může vznikat. Prvním z nich je způsob, kdy celý systém vznikne najednou jako celek. Po celou dobu se v něm nachází konstantní počet částic. Takový systém jako celek i zanikne. Jeho použití je především v simulacích, kdy

předem známe počet entit v systému (naprř. n-body problem). Druhým způsobem je vznik částicového systému dynamicky v čase. Tento typ je vhodný především do počítačových her či vizulizačních nástrojů.

Simulace částic má několik fází.

1. *Vznik a zánik částic*
   Částice v celém systému se mohou vyskytovat buď permanentně, nebo po omezený časový úsek. Z hlediska výkonu simulace je předem daný počet částic, které nezanikají, velice výhodný a výpočetně nenáročný. Tento případ je ale využíván velmi zřídka, protože nemá zdaleka takové možnosti, jako systém dynamický.

2. *Aktualizace rychlostí*
   Nejprve jsou vypočítány veškeré síly ovlivňující zrychlení částic. Pomocí těchto sil je poté určeno zrychlení a to udává změnu rychlosti. Rychlost určuje pozici částic v následujícím časovém okamžiku (typicky snímku).

   Následující diferenciální rovnice shrnuje základní vývoj částice i v čase t, kde m udává hmotnost částice, v její rychlost a f součet všech sil, které na ní působí.

   $$\frac{f_i(t)}{m_i} = \frac{dv_i(t)}{dt}; \quad v_i(t) = \frac{dx_i(t)}{dt}$$

   Základními silami, které se v částicovém systémů vyskytují jsou gravitační a třecí síla

   $$f_g = mg; \quad f_t = -k_d v,$$

   kde g je gravitační zrychlení a k je koeficient tření.

   Složitější silou, která působí mezi několika částicemi, může být například pružnost vyjádřena Hookovým zákonem.

3. *Aktualizace částic*
   V tomto kroce je zapouzdřena všechna výše popsaná činnost. Aktualizují se pozice částic, jejich zrychlení a vypočítají se nové síly, které na částice působí. Pokud je existence částice podmíněna kladnou hodnotou jejího života, zjišťuje se také, zda není roven nule. Podle života se také může vypočítat například nová hodnota barvy. V neposlední řadě je proveden test kolizí, pokud ovšem není zcela zanedbán.

Vykreslovací fáze

Účel vykreslovací fáze zřejmě není třeba vysvětlovat. Samotné simulace a výpočty s částicemi by nebyly to pravé, pokud bychom je nemohli pěkně vizualizovat. Zde je ale nutno podotknout, že při velkých množstvích částic může být vykreslování slabým místem celého programu. Proto je ideální, když je všechno renderování prováděno za pomoci GPU. Procesoru je pak ulehčeno a může se věnovat činnostem jako jsou fyzikální výpočty. Vykreslovácí fáze nemusí během simulací vůbec probíhat.

4. *Řazení (volitelná)*
   Řazení částic je nutné, pokud jsou částice vykreslovány pomocí metody zvané blending. Korektní vykreslování průhledných částic v kombinaci s neprůhlednými objekty je velice komplexní záležitost, a proto zde nebude uvedena. Informace o této problematice lze nalézt v [3].

5. *Převod texturovacích souřadnic na vertexy*
   Pokud jsou použity textury a částice není vizualizována pouze jako jeden voxel, je nutno namapovat souřadnice textury, popř. textur, na povrch vykreslovaného tělesa. Více o texturování lze nalézt v [6] a [12].

6. *Renderovaní*
   Nakonec dochází po všech výpočtech a převodech texturovacích souřadnic k vykreslení celé scény – renderování.

2.2 Využití částicových systémů

Zde jsou uvedeny některé vybrané částicové systémy a jejich využití.

N-body problem
N-body problem, v češtině problém N těles, je po staletí považován za jednu z největších fyzikálních výzev. Řešení této úlohy znamená určit ze znalosti počátečních podmínek a zákona vzájemného silového působení polohy a rychlosti těles v libovolném okamžiku. Jako charakter vzájemného silového působení je obvykle považován pohyb podle Newtonova gravitačního zákona [2].

Matematicky je problém N těles popsán

$$m_j \ddot{q}_j = \gamma \sum_{k \neq j} \frac{m_j m_k (q_k - q_j)}{|q_k - q_j|^3}, j = 1, \ldots, n \tag{2.1}$$

kde $m_1, m_2, \ldots, m_n$ reprezentuje hmotnosti těles a $q_1, q_2, \ldots, q_n$ značí funkce času udávající pozici v trojrozměrném prostoru.

Protože je tento problém neřešitelný analytickými metodami (pro n>3), využívá se simulací, ve kterých se uplatňují částicové systémy a numerická integrace, pomocí níž lze dosáhnout vysoké přesnosti. V problému N těles jsou velice důležitá silová působení ostatních částic, proto je nelze vypustit a ulehčit si tak výpočty tak, jak je to možné u jiných typů částicových systémů.

Tuhá tělesa
Dynamika částic je založena na newtonovské fyzice. Kdybychom chtěli sledovat chování *systému částic*, použili bychom metody částic zmíněné dříve. Existuje ale speciální případ, kdy si všechny částice mezi sebou zachovávají relativně stejné vzdálenosti i když se částicový systém jako celek pohybuje. Takový typ částicového systém můžeme nazvat *tuhým tělesem* (angl. rigid body). Simulace tuhého tělesa pomocí částicových systémů není příliš typická.

Vytváření implicitních ploch
Částicové systémy jsou také jednou z možností, jak simulovat resp. vizualizovat vytváření implicitních ploch [11].

Generování trávy a modelování rostlin
Generování rostlin jde provést různými způsoby, například procedurálně, nebo právě pomocí částicových systémů [9]. Na modelování trávy se použije explozivní typ částicového systému podobný částicovému systému použítému při simulaci ohňostroje. Částice ale nejsou vykreslovány pouze v diskrétních okamžicích jako u ohňostrojů, jsou vykreslovány po celou dobu

jejich existence, tj. vykreslují se po celé trajektorii jejich pohybu, částice vykreslené na starých pozicích zůstávají. Za požití odstínu zelené vznikají jednotlivé proužky trávy. K vytvoření celých travnatých ploch se použije mnoho takových systémů náhodně umístěných na ploše a vzájemně se překrývajících.

Modelovat realistické rostliny je kvůli jejich komplexnosti a vysoké míře detailu velice obtížné. Kompletní studie je uvedena v [10].

Exploze a ohňostroje
Typickým využitím částicových systému jsou různorodé pyrotechnické efekty využívané především v počítačových hrách. V těchto systémech se tedy klade důraz na vzhled po vykreslení, neznamená to ale, že by byla zanedbávána simulační fáze. Dobré exploze přidávají také jednoduchou detekci kolizí mezi částicemi a objekty ve scéně, které nemají dočasný charakter, např. zdmi nebo terénem.

Mraky
Modelování mraků je komplexní záležitostí. Individuální částice nelze v tomto případě renderovat pouze jako „barevné body“, je nutno je renderovat jako jednotlivé objekty odrážející světlo.

Mraky jsou komplexní z několika důvodů. Zaprvé, jejich tvar se nedá přesně popsat a je závislý na mnoho faktorech jako jsou směr větru, vlhkost, teplota, terén, apod. Jejich výpočty vedou na parciální diferenciální rovnice a řesení těchto rovnic je velice výpočetně náročné. Zadruhé, mraky jsou složité, protože na sebe mohou vrhat stín. Tato vlastnost je důležitá a musí se dodržet, aby mrak vypadal jako mrak. Posledním důvodem je fakt, že je na model mraku nutno použít obrovkého množství částic a z toho plyne potřeba efektivního algoritmu.

Protože simulace mraků pomocí částicových systémů je velice náročná a není ideální, používají se fraktální metody.

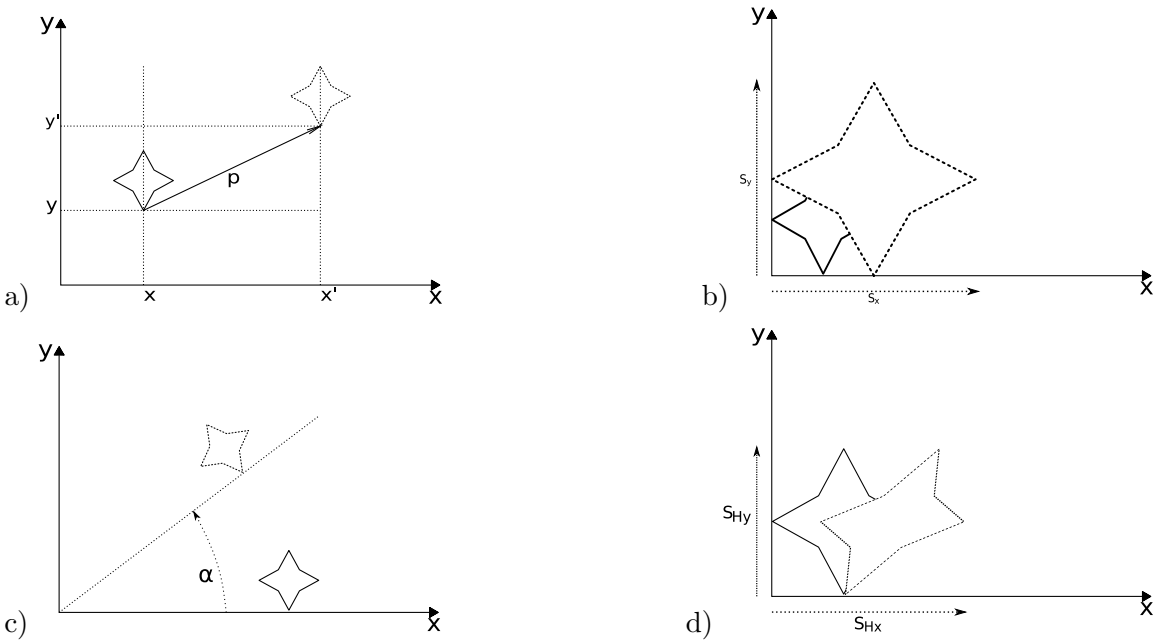
Simulace kapalin a plynů
Dalším z typických využití částicových systémů je simulace kapalin a plynů. Je velice populárním prostředkem pro generování realisticky vypadajících animací explozí, kouře, vody a věcí příbuzných. Existuje několik způsobů, jak simulace prchavých látek provádět, jednou z nich je použití zjednodušených Navier-Stokesových rovnic, popisujících proudění nestlačitelných newtonovských kapalin. V počítačové grafice je tohoto druhu simulace využíváno v mnoha oborech. Extrémně časově náročné vizualizace se provádí například ve filmech. Ve hrách se používají zjednodušené verze schopné pracovat v reálném čase [1].

Vytváření efektů na texturách
Využití těchto efektů je čistě v počítačových hrách. Pokud například hlavní hrdina ve hře začne střílet po svém nepříteli, který stojí před zdí, a mine, očekávaný optický výsledek jsou nerovnosti na zdi. Tyto nerovnosti je možné vytvořit pomocí částicových systémů. Když hlavní hrdina nepřítele zasáhne, na zdi se objeví krvavé skrvny, opět pomocí částicového systému.

Kapitola 3

Principy zobrazování objektů ve 3D scéně

Tato kapitola se bude zabývat obecnými principy, které jsou uplatněny při zobrazování ve 3D. Bude nastíněna základní problematika týkající se transformací a projekce, na závěr bude popsáno vykreslovaná pomocí grafické knihovny OpenGL.



Obrázek 3.1: Transformace: a) posunutí, b) změna měřítka, c) rotace, d) zkosení

3.1 Transformace ve 3D

Transformace bodů popsaných v homogenních souřadnicích jako $P = (x, y, z, \omega)$ můžeme chápat jako manipulace těmito body v prostoru. Pokud je potřeba tyto operace skládat, záleží na tom, v jakém pořadí jsou řazeny. Skládání transformací se věnuje následující sekce. Veškeré transformace mají také svojí inverzní formu, existují tedy jejich transformační matice, které způsobí posun v opačném směru, popř. rotaci a jiné.

Homogenní souřadnice
Homogenní souřadnice umožňují reprezentovat afinní transformace pomocí matic [12]. Umožňují se všemi transformacemi pracovat jednotným způsobem. Homogenní souřadnice vzniknou rozšířením původního prostoru o jednu dimenzi. Souřadnice bodu lze vyjádřit v homogenních souřadnicích jako $[x, y, z, \omega]$ a platí

$$X = \frac{x}{\omega}, \quad Y = \frac{y}{\omega}, \quad Z = \frac{z}{\omega} \tag{3.1}$$

Souřadnice ω se nazývá *váha* bodu a často se volí 1.

Translace (posunutí)
Posunutí bodu o vektor $\vec{p} = [p_x, p_y, p_z]$.

$$x' = x + p_x, \quad y' = y + p_y, \quad z' = z + p_z \tag{3.2}$$

Transformační matice pro posunutí bude vypadat takto:

$$\left[x', y', z', 1\right] = [x, y, z, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} \tag{3.3}$$

Pokud bychom chtěli docílit posunutí zpět na původní pozici, museli bychom použít matici inverzní:

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -p_x & -p_y & -p_z & 1 \end{bmatrix} \tag{3.4}$$

Scaling (změna měřítka)
Jedná se o vynásobení jedné nebo více souřadných os konstantou.

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$S^{-1} = \begin{bmatrix} \frac{1}{S_x} & 0 & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 & 0 \\ 0 & 0 & \frac{1}{S_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Rotace
S rotacemi je to o něco obtížnější než to bylo v případě posunutí či změny měřítka. Musí se rozlišit, kolem které souřadné osy chceme daný bod otáčet a podle toho sestrojit transformační matici. Transformační matice R_x, R_y, R_z postupně udávají směr rotace kolem os X, Y a Z o úhel α. Podle znaménka úhlu se rozlišuje, kterým směrem se bude rotovat. Když je kladný, otočení se provede proti směru hodinových ručiček, v opačném případě v jejich směru. Jako střed otáčení se bere střed souřadného systému.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\ \alpha & sin\ \alpha & 0 \\ 0 & -sin\ \alpha & cos\ \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$R_y = \begin{bmatrix} cos\ \alpha & sin\ \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -sin\ \alpha & cos\ \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$R_z = \begin{bmatrix} cos\ \alpha & sin\ \alpha & 0 & 0 \\ -sin\ \alpha & cos\ \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

Shearing (zkosení)
U zkosení se musí také rozlišovat, ve směru kterých souřadných os má zkosení proběhnout, podobně jako tomu bylo u rotací. Existují tedy transformační matice S_{XY}, S_{YZ}, S_{XZ} pro zkosení ve směrech XY, YZ, XZ

$$S_{XY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_X & S_Y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$S_{YZ} = \begin{bmatrix} 1 & S_Y & S_Z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

$$S_{XZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_X & 1 & S_Z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

3.2 Skládání transformací

V praxi by nám ovšem jednotlivé transformace nestačily, je proto potřeba skládat je, to znamená vytvářet jejich posloupnosti. Taková posloupnost se pak dá vyjádřit jednou maticí, nazývanou *obecná*, která vznikne vynásobením všech *dílčích matic*. Takové násobení však musí být prováděno z pravé strany a v přesném pořadí jednotlivých transformací.

Složení několika transformací bude vhodné demonstrovat na příkladu. Předpokládejme, že chceme vykreslit například trojúhelník, ale dále požadujeme, aby byl zmenšený na poloviční velikost, otočený „vzhůru nohama“ a posunutý o 10 jednotek do prava. Obecný maticový zápis tedy bude

$$M = M_I \cdot T' \cdot R' \cdot S' \tag{3.13}$$

kde M je výsledná projekční matice, M_I je jednotková matice a T', R' a S' vyjadřují po

řadě posunutí, otočení a změnu měřítka. Po dosazení našich konkrétních hodnot dostaneme

$$M_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

$$T' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 10 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$R' = \begin{bmatrix} cos\ 180 & sin\ 180 & 0 & 0 \\ -sin\ 180 & cos\ 180 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$S' = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

Pořadí je zde velice důležité. Pokud bychom například jako první aplikovali operaci změnu měřítka, následující operace by pracovaly s polovičními hodnotami. To znamená, že by se trojuhelník posunul o 5 jednotek a otočil o 90 stupňů.

Kdybychom se dále rozhodli, že náš trojuhelník potřebujeme vykreslit na jeho původní pozici, bylo by nutno operace aplikovat pomocí inverzních matic, ale v opačném pořadí.

$$M = S'^{-1} \cdot R'^{-1} \cdot T'^{-1} \tag{3.18}$$

3.3 Projekce

Projekce je transformace realizující redukci dimenze 3D prostoru na 2D prostor při zachování parametrů použitého zobrazení. Projekce provádí redukci dimenze prostoru v definovaném směru (směru pohledu), nejčastěji ve směru osy -Z. *Pracuje se s vrcholy projektovaných objektů a zachovává se druh jejich reprezentace (vektorová, rastrová).*
[4]

V podstatě se tedy jedná o zobrazení (promítání) bodů v prostoru do části roviny, typicky jde o obdélník. Podle toho jakým způsobem se promítání bodů provádí rozlišujeme dva typy projekce - paralelní (rovnoběžná) a perspektivní (středová), které je možno dálé dělit.

Paralelní (rovnoběžná) projekce je projekce zobrazující bod pomocí rovnoběžných paprsků. Zachovávání velikosti objektů ovšem nepodává dostatečně reálný vjem, nepoužívá se proto v aplikacích zachycujících nějakou 3D scénu, ale například v aplikacích pro tvorbu techických nákresů. Proto se jí tato práce nebude dále zabývat.

Paralelní projekci je možné ještě dále dělit podle úhlu, pod kterým dopadají paprsky na průmětnu.

Perspektivní (středová) *projekce je nelineární neafinní projekce, která zobrazuje vrcholy promítaných objektů prostřednictvím paprsků protínajících se v jednom bodu, ve sřtedu projekce. Střed projekce je zároveň většinou místem pozice pozorovatele. Velikost*

průmětů objektů je nepřímo úměrně závislá na jejich vzdálenosti od průmětny. Čím je objekt blíže průmětně, tím je jeho obraz větší a naopak. Rovnoběžnost promítaných hran není v této projekci zachována. [4]

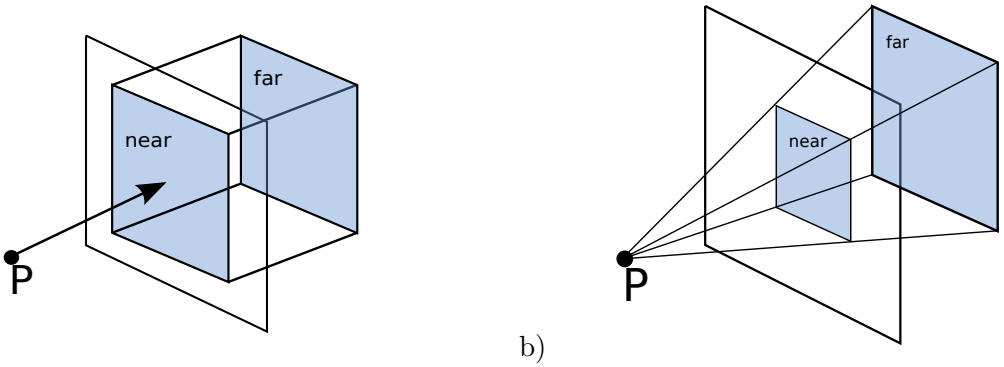
Perspektiva je zde chápána jako přibližná reprezentace objektů viděných lidským okem na ploše (např. monitoru). Její využití se tedy více projeví v počítačových hrách nebo virtuální realitě.

Z velkého množství druhů perspektivních vykreslování jsou nejběžněji používaná jedno, dvou a tříbodová projekce. Názvy napovídají kolik bylo použito bodů (angl. vanishing point), do kterých se paprsky sbíhají.

3.4 Pohledový objem

Pohledový objem je část 3D prostoru, obsahuje zobrazované (viditelné) objekty. Pohledový objem je ohraničen okraji okna protaženými do prostoru do tvaru hranolu (při paralelní projekci) nebo komolého jehlanu (při perspektivní projekci). Ve směru pohledu je objem uzavřen přední a zadní ořezávací plochou. [4]

Zjednodušeně řečeno, pohledový objem (angl. frustrum) tedy definuje, co jsme zrovna schopni ve scéně vidět. Tím je také jednoznačně určeno, co vidět není, a proto je důležitý i z hlediska výkonu. Objekty mimo vymezený prostor nebo objekty viditelné pouze z části jsou ořezány a tím je urychlen vykreslovací proces.



a) b)

Obrázek 3.2: Projekce: a) paralelní projekce, b) perspektivní projekce

3.5 OpenGL

OpenGL (Open Graphics Library) je multiplatformní knihovna, podporující hardwarovou akceleraci a 3D API na nízké úrovni. Je nezávislá na hardwaru a neposkytuje prostředky pro popis modelů trojrozměrných objektů. Takové entity je potřeba vykreslit pomocí grafických primitiv - bodů, úseček a polygonů. Více informací ohledně OpenGL lze nalézt v [6].

Vykreslování v OpenGL probíhá jako série navzájem propojených operací, která se nazývá *vykreslovací řetězec (angl. rendering pipeline)*. Vykreslovací řetězec je znázorněn na obrázku 3.3. Data nesoucí informaci o geometrii – body, úsečky a polygony [1] – je potřeba

[1] Součástí OpenGL je rozšíření GLU, pomocí kterého je možné vykreslovat i složitější entity, jako jsou například koule nebo elipsy či jejich části

zpracovat jinak, než data pixelů (bitmapy, obrázky, nebo samotné pixely). Oba druhy vstupních dat poté prochází řetězcem, jsou zpracovány, sloučeny a zapsány do framebufferu. OpenGL může také obsah framebufferu poskytnout aplikaci (viz přerušované linky v obrázku 3.3).

Display Lists
Display list je sekvence OpenGL příkazů uložených pro pozdější nebo okamžité použití. Všechna data, geometrie a obrazová data mohou být v těchto sekvencích uloženа.

Evaluators
Evaluátory poskytují metodu, která odvozuje body plochy vzniklé z kontrolních bodů pomocí metody polynomiálního mapování.

Primitive assembly
Odstraňování těch částí geometrie, které zrovna nejsou ve scéně viditelné. V této části je také prováděno odstraňování odvrácených polygonů (angl. culling).

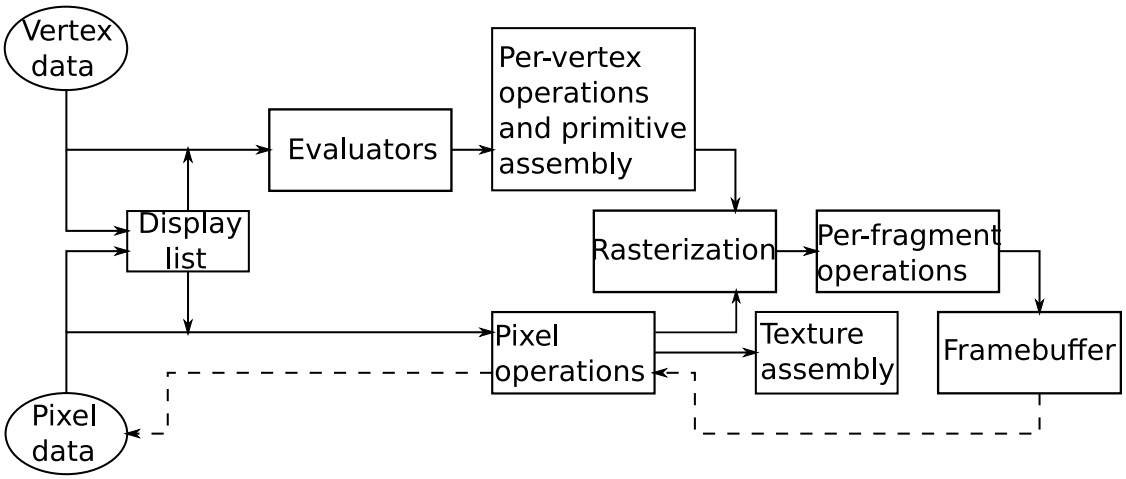
Pixel Operations
Načítání pixelů z paměti a rozbalení z různých formátů do správného počtu komponent. Aplikace změny měřítka a zpracování pixelovou mapou. Výsledek je sloučen a poslán dále do rasterizační části, nebo zapsán do texturovací paměti.

Texture Assembly
V této části se provádí různé operace nad texturami, jako je mipmapping, multitexturing, apod. Textury jsou v ideálním případě poslány rovnou do grafické karty, v tom horším pak uloženy do systémové paměti, odkud jsou posílány do grafického akcelerátoru při vykreslování.

Rasterization
Rasterizace je přeměna geometrických i obrazových dat do fragmentů. Každá část fragmentu odpovídá pixelu ve frame bufferu.

Fragment Operations
V případě, že je povoleno texturování, je pro každý fragment vygenerován jeden pixel nazývaný texel. Ten je pak na fragment aplikován. Spolu se samotným texelem je generována také jeho barva.

Frame buffer
Výstup všech operací se nakonec dostane do frame bufferu, který se skládá z několika samostatných bufferů - color buffer, Z-buffer, stencil buffer a accumulation buffer. Ve frame bufferu jsou uloženy fragmenty, které představují průřez všemi buffery.



Obrázek 3.3: Vykreslovací řetězec

Kapitola 4

Návrh a implementace

V této kapitole bude popsán způsob, jakým bude navrhnut a implementován systém zmíněný výše. Budou objasněny principy dělení objektu na menší části, jejich integrace do hry. Nakonec bude popsána implementace hry a jejích jednotlivých subsystémů.

4.1 Nástroje pro tvorbu programu

Jako programovací jazyk použitý pro implementaci tohoto projektu bude použito C++. Tento jazyk se přímo nabízí, protože je velice rozšířen v oblasti vývoje počítačových her a jeho vlastností je podpora objektově orientovaného programování.

Základní funkčnost částicových systému bude vytvořena pomocí Partice Systems API. To patří mezi jeden z mála samostatných systémů, který v sobě neobsahuje další funkčnost navíc (např. celý herní engine), která by pro tvorbu rozšíření mohla být matoucí a zbytečná. Bylo také oceněno jako „pravděpodovně nejlepší nástroj pro poznávání a vytváření částicových systémů“. Možné alternativy jsou uvedeny v dodatku B.

Vizualizační část bude tvořena za pomoci OpenGL knihoven. Její funkčnost je ve větším detailu popsána v sekci 3.5, podrobně pak v [6].

Další součástí implementace byly kompilátor GCC a podpůrné programy jako detektor chyb při práci s pamětí valgrind a GNU profiler.

4.2 Rozšíření systému Particle Systems API

Možnosti systému Particle Systems API
Ačkoliv se nejedná o kompletní fyzikální engine, funkčnost tohoto API je dostačující pro středně složité fyzikální operace a je užitečné pro základní animační software nebo například spořiče obrazovky. Je také vhodné pro integraci ve hrách, především díky jeho jednoduchosti. Funkčnost Particle Systems API je možné rozdělit do několik kategorií.

Seznamy akcí, nebo-li action lists, jsou posloupnosti akcí, které je možné uložit pro pozdější použití. Obdoba pro vykreslování jsou display lists v OpenGL. Seznamy akcí je možné přidávat, odstraňovat a volat.

Do této kategorie se také řádí příkaz pro nastavení časového kroku. Systems API používá diskrétní aproximaci času pro všechny svoje akce. To znamená, že akce jsou na částice aplikovány instantně v určitém čase a po krátkém akumulovaném časovém itervalu dt. Čas se poté posune právě o tento interval. Tato metoda je standardem téměř pro všechny

simulace v čase. Závislost časového kroku na fps (frames per second, počet snímků za sekundu) je vyjádřena jako

$$dt = \frac{1}{fps}.$$

Časový krok *dt* je implicitně nastaven na 1.0, tzn. akce nad systémy jsou prováděny při každém snímku. [1]

Akce kontextů umožňují definovat chování částicových systémů a také vytvářet částice různým způsobem – generovat v časových intervalech nebo instantně. V systému je podporováno relativně velké množství typů chování částic a částicových systémů. K základním požadavkům patří nastavení a aplikace gravitace, přímý a rotační pohyb, nastavení rychlostí a zrychlení, odstranění částic při překročení určité hranice a další. Z pokročilejších funkcí lze jmenovat například nastavení odporu prostředí, který je aplikován při pohybu, definice explozivních sil aplikovaných na částice směrem od středu exploze, odrážení částic od definovaných primitiv (koule, rovina, kužel, kvádr, aj.) nebo aplikace sil, které systém částic formují do tvaru víru. Uvedené akce nejsou kompletní. Výčet všech akcí a jejich stručný popis lze nalézt v dokumentaci [7].

Ostatní akce nad systémy a jejich skupinami. Kopírování částic, nastavování callback funkcí, které se volají při vzniku a zániku částic, operace se skupinami, přístup k částicím a další.

Štěpicí systém

Štěpicí systém je první částí rozšíření systému Particle Systems API. Jeho úkolem je zajistit rozdělení modelu na menší kousky a jejich správa. Jádrem celého stěpicího systému je třída s názvem `CModelSplitter`. Ta se stará o udržování informaci o modelu, který se má rozštěpit, a o všech částech vzniklých jako výsledek operace `Split`, nebo-li štěpení. To se provádí pomocí *roviny zadané v obecném tvaru.*

V následujících mechanismech je často potřeba provádět testy na shodu dvou bodů, tj. jestli mají oba body v prostoru totožné souřadnice. Protože byl k jejich reprezentaci použit datový typ `float`[2], není možné vzhledem k nepřesnostem, kterých se dopouští procesor při výpočtech, testovat jejich shodnost pomocí operátoru `==`, ale používá se intervalu o jisté nepřesnosti. V případě této práce se využívá hodnoty 0.001, k níž bylo dospěno pokusy. Jedná se o nepřesnost poměrně velkou, bohužel se ní však nelze vyhnout. Její použití si nejspíše vynutilo to, že jeden z modelů je obrovský a asi ne zcela perfektní. Pro zajímavost: jedná se o model hráčovi lodi a musel být 11000× zmenšen.

Nejdůležitějšími částmi rozbíjecího mechanismu jsou rovina zadaná v obecném tvaru a datová struktura popisující hraniční reprezentaci 3D modelu, tedy vektor obsahující všechny trojúhelníky. U každého bodu trojúhelníku (vertexu) je informace o jeho pozici v prostoru, texturovacích souřadnicích náležících danému bodu a souřadnice normálového vektoru.

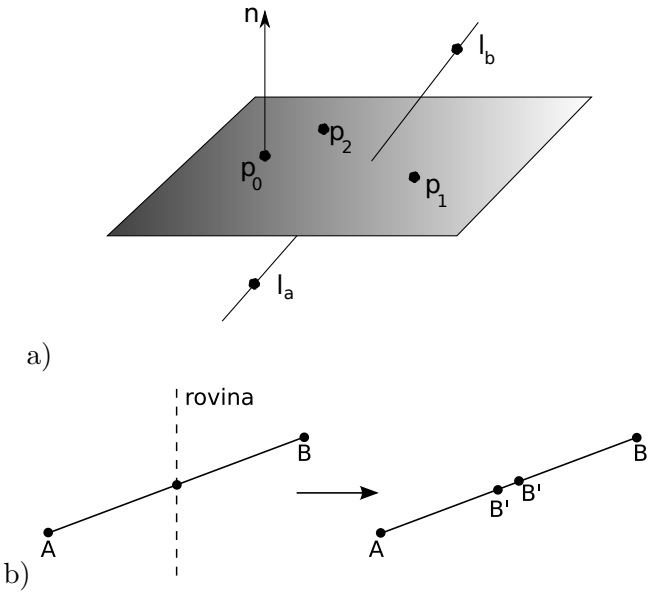
Aby bylo možné trojúhelník rozdělit pomocí roviny, je nutno vypočítat průsečíky dané roviny a úseček tvořených mezi každými dvěma body tohoto trojúhelníku. Jednoduchý algoritmus se dá tedy vyjádřit takto:

1. zjisti vzájemnou pozici dvou bodů na úsečce

2. nejsou-li oba na stejné straně roviny, pokračuj, jinak skonči

[1] přebráno z dokumentace

[2] http://en.wikipedia.org/wiki/Floating-point_number



Obrázek 4.1: Průnik přímky a roviny

3. zjisti poměr u dvou po řezu vzniklých úseček
4. pokud je u rovno 0, rovina protla úsečku v koncovém bodě
5. pokud je u v intervalu od 0 do 1, vypočti nové souřadnice

Rovina je zadána v obecném tvaru

$$Ax + By + Cz + D = 0, \tag{4.1}$$

kde $\vec{n} = (A, B, C)$ udává souřadnice normálového vektoru roviny, pomocí kterého se dá zjistit vzájemná poloha z bodu 1. Bod leží na kladné části poloprostoru (ta část, kam směřuje normálový vektor), pokud platí, že

$$\vec{n} \cdot \vec{p} \geq -D, \tag{4.2}$$

kde $\vec{n}$ je normálový vektor roviny a $\vec{p}$ je vektor udávající pozici bodu.

Pokud jsou tedy oba body na jiné straně roviny, má smysl dále počítat její průsečík s úsečkou jimi tvořenou, jak je tomu v kroku 3. Poměrná vzdálenost u je spočtena jako

$$u = \frac{A \cdot l_{a_x} + B \cdot l_{a_y} + C \cdot l_{a_z}}{A \cdot (l_{a_x} - l_{b_x}) + B \cdot (l_{a_y} - l_{b_y}) + C \cdot (l_{a_z} - l_{b_z})} \tag{4.3}$$

a platí, že $u \in (0, 1)$. Graficky je to znázorněno na obrázku 4.1.

Tímto bylo dosaženo vypočtení poměrné vzdálenosti. Výsledkem řezu jsou ale dva nové body [3], jejichž nové souřadnice, texturovací souřadnice a normály musí být vypočteny.

$$\vec{p}_{new} = l_a + (l_b - l_a) \cdot u \tag{4.4}$$

[3] protnutí pouze jednoho bodu znamená také dva body, jsou ale identické

Texturovací souřadnice jsou vypočítány naprosto stejně, pouze se místo souřadnic bodů l_a a l_b použíjí texturovací souřadnice v těchto bodech. Jelikož se normálové vektory ve vertexech nikde nevyužívají, jsou nastaveny na hodnotu -1 a dále se s nimi nepracuje. Kdybychom však chtěli později do programu dodat například výpočty osvětlení, byly by normály potřeba. Osvětlení bylo z implementace vypuštěno kvůli komplexnosti scény.

V této chvíli jsou k dispozici dva polygony vzniklé po průřezu (může dojít ke vzniku dvou trojúhelníků, ale i jednoho trojúhelníku a jednoho čtyřúhelníku, viz obr. 4.2). Oba se zařadí na správnou stranu poloprostoru. Tato metoda je pak systematicky použita na všechny trojúhelníky a tím je docíleno, že byl model „rozpůlen“. Jako vedlejší výstup vznikne vektor úseček (dva body průřezu vždy tvoří jednu úsečku), které jsou dále použity na vytvoření polygonu zakrývajícího jinak dutý vnitřek modelu.

Vytvoření polygonu pro zakrytí vnitřku částí modelu

Poté, co se provede jedno nebo několik rozříznutí zvoleného modelu rovinou, vzniknou části, do kterých „je vidět“. To je dáno tím, že je model reprezenován pouze jeho ohraničením, tedy dalo by se říct pláštěm. Když ale například ve hře sestřelíme letadlo, jeho prázdnost uvnitř a fakt, že je vidět na venkovní plášť zevnitř, působí velice rušivě. Byl proto vytvořen jednoduchý algoritmus starající se o vytvoření polygonu navíc. Tento polygon je přiložen na průřez a tak vnitřek modelu uzavře.

Když probíhá rozřezávání polygonů, vzniká současně jako vedlejší produkt seznam úseček. Ty jsou později použity pro správné určení obvodu polygonu, viz dále. Následující algoritmus vytváří z těchto úseček polygony na uzavření modelu po průřezu.

```
dokud seznam s úsečkami není prázdný {
    první = první bod první úsečky v seznamu
    aktuální = první + 1<<16
    vytvoř nový polygon
    dokud se nerovnají první a aktuální {
        pokud se jde poprvé, aktuální = první
        cyklus přes všechny úsečky {
            pokud se rovna bod 1 úsečky {
                aktuální = bod 2
                smaž úsečku
                přidej bod 1 do polygonu
            }
            pokud se rovna bod 2 úsečky {
                aktuální = bod 1
                smaž úsečku
                přidej bod 2 do polygonu
            }
        }
    }
    přidej výsledný polygon k části modelu
}
```

Po průřezu nejsou body seřazeny správně po obvodu, a proto je nelze správně vykreslit. Pokaždé když je rovinou rozdělen trojúhelník, vznikají dva body (pokud jej rovina rozdělí v jednom z bodů, jsou za výsledek považovány dva body se shodnými souřadnicemi). Nelze ovšem zajistit jejich správné pořadí, protože není zřejmé, v jakém pořadí jsou procházeny

trojúhelníky při rozdělování. Je tedy nutné uchovávat informaci o návaznosti. K tomuto účelu jsou vzniklé dva body uloženy do abstraktního datového typu úsečka. Seznam úseček není sice také seřazen, vždy ale existují dvě úsečky sdílející bod se shodnými souřadnicemi. Tento shodný bod je klíčový pro určení návaznosti jednotlivých úseček a tedy i správné seřazení.

Pokud by došlo k rozdělení modelu rovinou, jehož výsledkem je více než jeden polygon na průřezu, musí se správně rozlišit všechny „obvody“ a pro každý vytvořit jeden polygon. Očekávaný výsledek lze vidět na obrázku 4.4.

Předpokládá se, že výsledné polygony nebudou konvexní, protože ani model nemusí být konvexní. Při vykreslování nekonvexních polygonů běžnou technikou v OpenGL, tzn. sekvence příkazů

```
glBegin(GL_POLYGON);
    ...
    glVertex3f(...);
    ...
glEnd();
```

nedojde k vytouženému výsledku. Takto vykresovaný polygon není vykreslen správně, proto je při jeho vykreslování nutno použít GLUtesselator – vestavěnou strukturu z pomocné knihovny GLU, která zajistí rozčlenění polygonu na trojúhelníky a následné vykreslení.

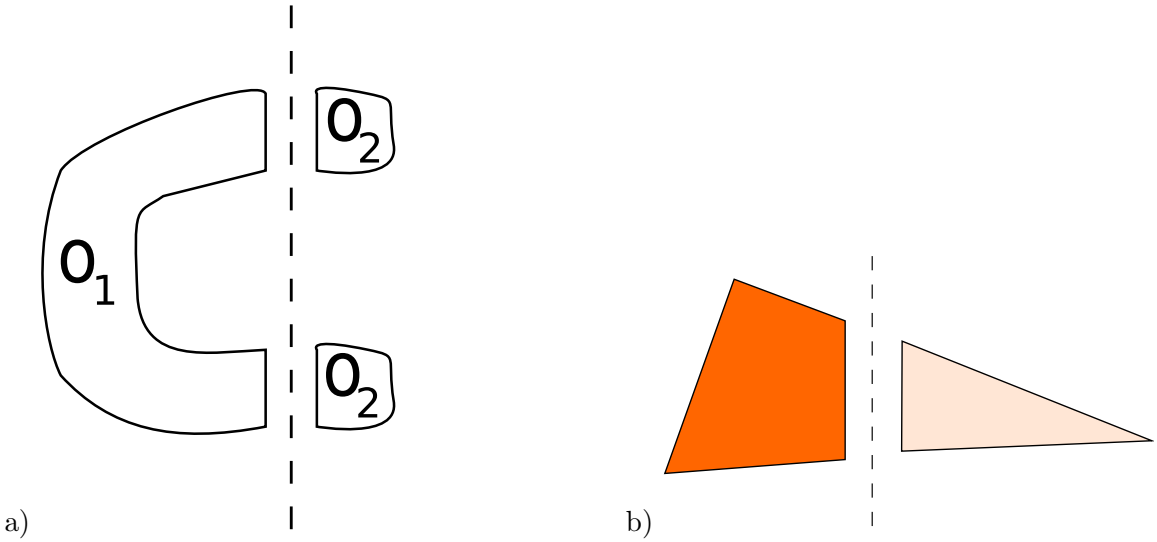
Problém nekonvexnosti
Model je tvořen trojúhelníky. Protože je trojúhelník konvexní útvar, po jeho rozdělení rovinou vzniknou dva útvary, které jsou také kovexní (obr. 4.2 b). Pokud dělíme dále, což je samozřejmě možné, vznikají konvexní čtyřúhelníky. Konvexnost zaručuje, že při řezu rovinou nebude docházet k nepřesnostem u vzniklých částí.

Ovšem zaručit konvexnost celého 3D modelu je téměř nemožné, pokud tedy pomineme velice jednoduché modely typy krychle, koule, apod. Při průřezech nekonvexních modelů budou vznikat nepřesnosti takového charakteru, jak je znázorněno na obrázku 4.2 a. Po řezu rovinou (naznačena jako přerušovaná čára) vzniknou z původního objektu dva nové. Jak je jistě cítit, objekt O_2 byl měl být ještě rozdělěn na další dva objekty a výsledek by tedy měly být objekty celkem tři. Není tomu ale tak. Tento fakt je však v práci zanedbán z důvodu přílišné komplexnosti problematiky dělení 3D objektů na části. Ve velkém detailu a s ohledem na fyzikální přesnost je tato problematika popsána v [8].

PartsSystem
Druhá část rozšíření je specializována na částicový systém, který je rozšířen o několik funkcí potřebných především k vykreslení rozděleného modelu. Je rozdělena do několika tříd.

ParticleSystem je nadstavba na funkčností Particle Systems API. Umožňuje pracovat se skupinami částic jako s celkem, rozdělování skupin do podskupin pro větší dynamičnost a zajišťuje vykreslování a pohyb všech skupin. Dále je možné skupinám definovat textury nebo tvary. Implicitně jsou základní částice v této třídě vykreslovány jako texturované čtverce, pro větší rozmanitost lze však pomocí tvaru lze definovat jakýkoliv čtyřúhelník.

PartsSystem je rozšíření ParticleSystem. Obsahuje dvě části. První je zděděná od třídy ParticleSystem a představuje především fyzikální funkčnost. Druhou část tvoří systém, který zajišťuje vykreslování částí modelu.



Obrázek 4.2: a) Rozříznutí nekonvexního tělesa rovinou, b) rozříznutí trojúhelníku

Protože jsou částice v prostoru reprezentovány pouze jako body a části modelu jako prostorové útvary, je potřeba zvolit vhodnou metodu pro jejich vykreslení. Nabízí se mapovat pozici částic na pomyslný střed rozbitých částí. Složky souřadnic pomyslného středu jsou vypočítany jako aritmetické průměry složek všech bodů tvořících jednotlivé rozdělené části modelu.

Součástí funkčnosti této třídy jsou také struktury pro ukládání informací o modelu (aby bylo možné na rozbité kousky mapovat správnou texturu a vykreslovat je v odpovídajícím měřítku) a ukládání geometrie jednotlivých kusů. Jako poslední je zde systém, který v určitých intervalech vypouští z letících kusů obláčky kouře.

4.3 Demostrační hra

Pokud čtenář patří mezi hráče, kteří si rádi zahráli takovou legendu, jako byl Raptor, jistě jim demonstrační hra nebude cizí. Samozřejmě se s takovým titulem nemůže rovnat, sloužil ale jako insiprace. Pro inspiraci také posloužila novější verze „typické“ vesmírné střílečky – Astromenace.

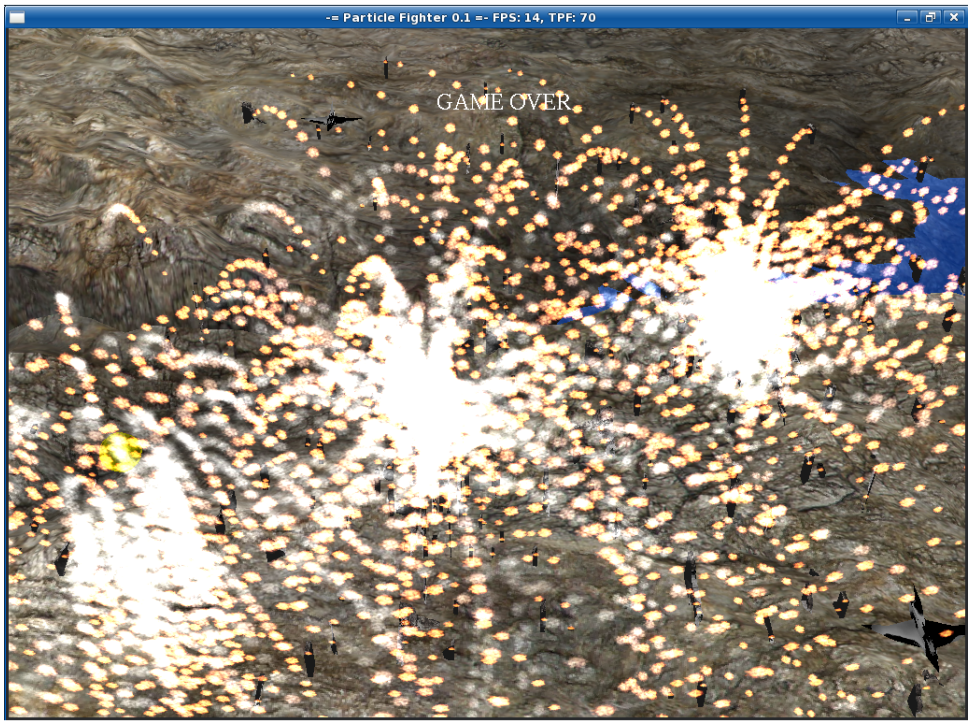
Hra nese výstižný název *Particle Fighter* a v následujících sekcích budou popsány detaily implementace.

4.4 Manažery

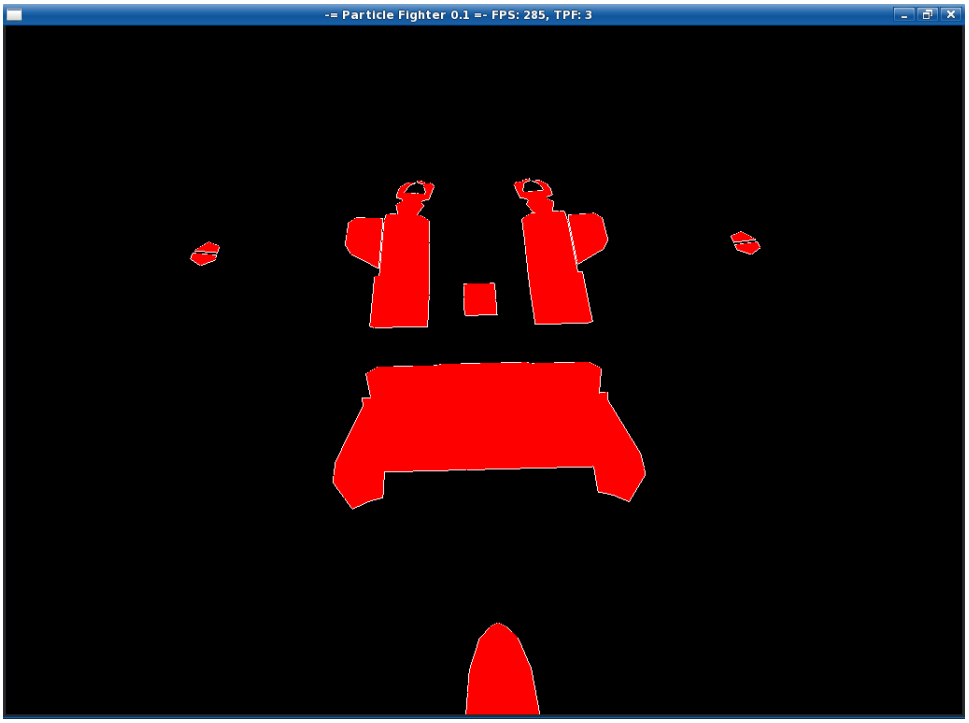
Manažery jsou třídy umožňující jednoduchou správu a ovládání jim přiřazených entit.

Manažer modelů
Správce modelů je využit pouze na přidávání, odebírání a přístup k jednotlivým modelům ve hře. Žádné složitější funkce se v něm neprovádí.

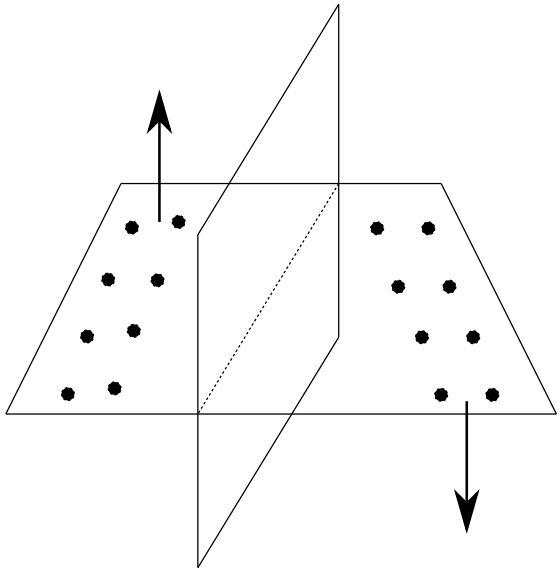
Manažer objektů
Objekt je vše, co není částicový systém. Za objekt jsou považovány nepřátelé, střely nebo



Obrázek 4.3: Výbuch hráče a nepřátel



Obrázek 4.4: Správný průřez modelem



Obrázek 4.5: Průběh jedné iterace při generování terénu

hráč. Objekty mezi sebou mohou kolidovat a je potřeba vést informaci o tom, které objekty mohou kolidovat s kterými a jaká akce se má provést, pokud se tak stane (alternativním a také lepším přístupem ke kolizím různých druhů objektů je systém založený na zasílání zpráv; nebyl implementován z důvodu větší složitosti). Správce objektů tedy podporuje kontrolu kolizí. Řídí také pohyb objekty a jejich vykreslování. Zajišťuje správné přidání a odebrání objektů ze hry.

Manažer částicových systémů
Zajišťuje správné přidávání, odstranění, veškerou aktualizaci a vykreslování částicových systémů.

4.5 Terén

Pro vymezení desky (soubor bodů) je použito 65 bodů na výšku i šířku, tvořících čtvercovou plochu (viz obrázek 4.8). Vzdálenost mezi body v kolmých směrech je 0.96 jednotek[4]. Tyto body zároveň určují i výšku terénu. Desku tedy tvoří celkem 64×64 políček. Všechny hodnoty v této části byly zvoleny jako optimální a bylo k nim dospěno pokusy. Uvedený postup pro generování terénu patří k těm nejjednodušším, alternativní postupy lze nalézt v [5].

Generování terénu
Pomyslným středem desky je vedena rovina kolmá na orientaci desky (deska je položená v rovině os x a z) a tím jí rozděluje na dvě části. Následně jsou všechny body s výjimkou okrajových v jedné části posunuty o náhodnou hodnotu (všechny o stejnou) nahoru nebo dolu, viz obrázek 4.5. Okrajové body nejsou posunuty, aby bylo možné vykreslovat terén jako spojitou plochu a nebylo pro to třeba dalších výpočtů. Tím je vytvořen „zlom“ na desce a dokončena jedna iterace.

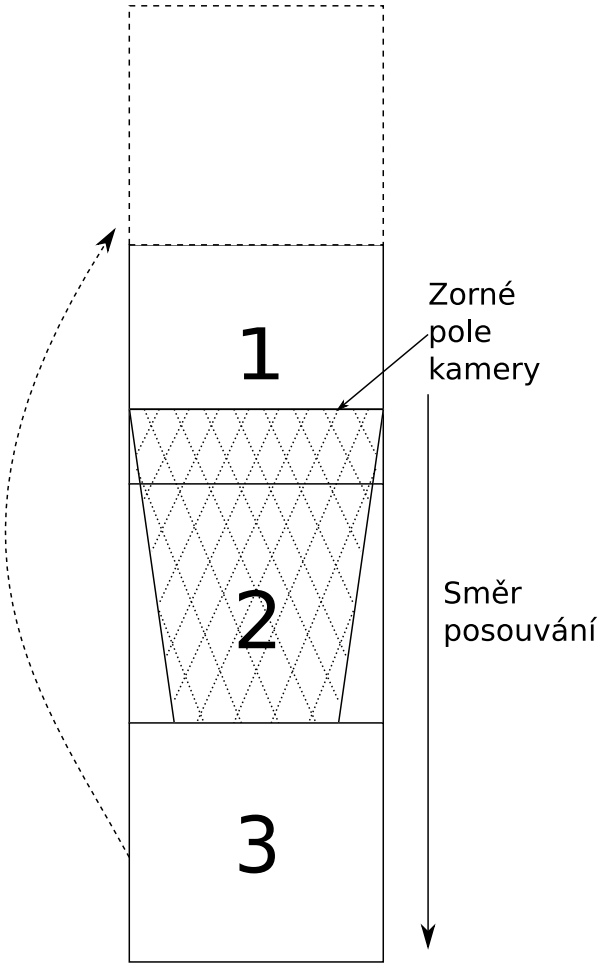
Rovný terén, na kterém se nachází pouze jeden zlom by nepůsobil příliš přirozeně. Je

[4]jednotek, které používá OpenGL

proto potřeba provést iterací více. Čím více iterací, tím realističtěji terén vypadá. Je ale důležité vyvážit poměr mezi počtem iterací a hodnotou, o kterou se mají části desky snižovat nebo zvyšovat, aby nedocházelo k příliš ostrým zlomům. V implementaci demostrační hry je použito 800 iterací.

Způsobem popsaným výše může někdy docházet (je to závislé na vygenerovaných hodnotách) k vzniku vyšších kopců, popř. nižších údolí, než je požadováno. Proto je možné nastavit limity pro maximální, resp. minimální výšku. Je-li limit v některém ze směrů překročen, je vygenerován nádhodný vektor, který posune limitní hodnotu opačným směrem, tzn. pokud je vygenerována hodnota překračující maximální výšku terénu, náhodný vektor posune výšku terénu v daném bodě směrem dolů. Podobně je to to pro minimální výšku. Tím vznikají nerovnosti. Po překročení limitu není výška zarovnána přímo na hodnotu limitu, protože by v těchto místech docházelo ke vzniku rovných částí terénu, což by opět nepůsobilo přirozeně.

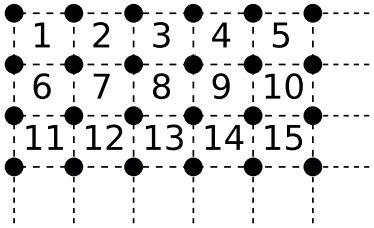
Vygenerovány jsou tři desky, jejichž pobyhem je později vytvořena iluze letících objektů nad terénem.



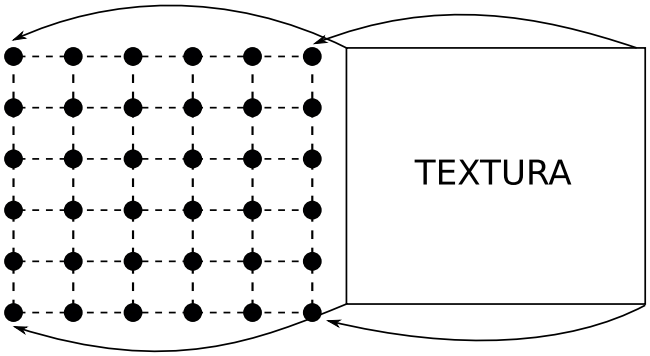
Obrázek 4.6: Přesouvání desek s terénem

Posouvání terénu

Celý terén je složen ze tří desek, které jsou posouvány konstantní rychlostí směrem ke



Obrázek 4.7: Procházení políček na deskách



Obrázek 4.8: Způsob texturování terénu

kameře. Jsou za sebou položeny tak, že pokud se jedna dostane „pod kameru“, zbylé dvě stále vyplňují spodní část scény. Deska, která se nyní nachází pod kamerou je posunuta před první dvě (viz obrázek 4.6). Tím dochází k vytvoření souvislého a nekonečně dlouhého pásu terénu.

Textura a vykreslování
Aby terén mohl být dostatečně velký, je tvořen výškovými body, mezi kterými jsou určité vzdálenosti. Není proto vykreslován po bodech, jako by tomu bylo například při vykreslování výškové mapy, ale po čtvercích. Způsob procházení desky po čtvercích je znázorněn na obrázku 4.7.

Terén je také otexturován. Způsob mapování textury je naznačen na obrázku 4.8. Jako poslední je provedeno nastavení světlosti podle toho, jak je daný výškový bod vysoko. Tím je simulováno osvětlení terénu.

4.6 Levely

Level, nebo-li herní úroveň, jak je chápán v Particle Fighteru, tvoří pouze nepřátelé. Jsou vedeny informace o typu nepřítele, čase jeho objevení a počáteční pozici.

Skriptování
Levely je možné upravovat v externích souborech. Lze vytvořit 9 úrovní a neomezený počet nepřatel v nich.

Jednotlivé skripty s levely se musí nacházet ve složce `levels` a musí být pojmenovány jednotným stylem – `levelX.lvl`, kde `X` udává pořadové číslo úrovně. Číslování začíná číslicí jedna.

Název	prodleva	význam prodlevy
first_level	125	před prvním levelem
start_level	600	mezi levely
longwave	800	delší mezera mezi nepřáteli
wave	400	mezera mezi nepřáteli
long	160	dlouhá
normal	100	střední
short	50	krátká
none	0	žádná prodleva

Tabulka 4.1: Časové hodnoty symbolických konstant

Typ	Popis
interceptor	pouze se pohybuje
fighter	pohybuje se a střílí na hráče
boss	doletí do jedné třetiny obrazu, začne létat ze strany na stranu vypouštět proti hráči salvy ničivých střel

Tabulka 4.2: Typy nepřátel

Syntax ve skriptu vypadá následovně:

```
level
typ_nepritele    px, py, pz    vx, vy, vz    prodleva
typ_nepritele    px, py, pz    vx, vy, vz    prodleva
...
nextlevel
end
```

Každá úroveň musí začínat klíčovým slovem `level` a končit slovem `end`. Mezi nimi se nachází výpis nepřátel. Položka nepřítele je uvedena jedním z klíčových slov z tabulky 4.2. Souřadnice jsou zaznamenány pomocí jednotlivých složek `px`, `py`, `pz`. Trojice `vx`, `vy`, `vz` udává složky vektoru rychlosti. Posledním slovem je jedna ze symbolických konstant z tabulky 4.1. Její význam je takový, že nepřítel bude vytvořen po uplynutí tolika časových jednotek, kolik udává konstanta. Klíčové slovo `nextlevel` se ve skriptu musí vyskytovat, pokud je požadováno načtení další úrovně.

4.7 Kolize

Protože se v běžných částicových systémech zpravidla nachází velké množství částic, kolize mezi nimi nejsou většinou brány v potaz. V pokročilých systémech se však řeší kolize mezi částicemi a různými objekt, které nemají dočasný charakter. Jako příklad může být považováno například odrážení jisker od zdi.

V této práci byly kolize částic vůči ostatním zanedbány z důvodu velké složitosti a náročnosti na výpočetní zdroje. Byly implementovány pouze mezi entitami, které nějakým způsobem reagují na okolí, tzn. hráč, nepřátelé, projektily a objekty vylepšující hráčovi vlastnosti (běžně ve hrách označovány jako powerupy). Pro jednoduchost byla zvolena kolizní metoda pomocí *bounding boxů*, nebo-li aproximace 3D modelu pomocí kvádru.

Kapitola 5

Závěr

Mým úkolem bylo rozšířit existující částicový systém. Jako systém vhodný pro rozšíření jsem zvolil Particle Systems API a z jeho funkčnosti jsem také vycházel.

Byl navržen a implementován systém, který načtený 3D model rozdělí pomocí rovin na části. Jejich pohyb je ovlivňován částicovým systémem, pomocí kterého jsou také vykreslovány. Postup práce byl rozdělen na dvě části.

Jako první byl vypracován algoritmus rozdělující model na části pomocí roviny. Tento přístup k rozbití modelu je značně jednoduchý, naprosto ale neodpovídá realitě. Byl však zvolen z důvodu jednoduchosti. Reálističtější řešení by znamenalo využití komplexních metod pro rozbití modelu.

Druhá část implementuje „objemové rozšíření částic“. Jelikož jsou částice simulovány jako hmotné body a části 3D modelu reprezentují objem, byl implementován způsob, který mapuje částice na rozbité kousky. Pozice částic byly namapovány na pomyslné středy kousků. Tím byl zajištěn prostředek pro jejich simulaci. Geometrie rozbitých kousků byla poté použita pro vykreslování částic, tedy pro jejich objemovou reprezentaci.

Vzhled a funkčnost výsledného rozšíření se jeví jako dobré a demonstrační hra jím je zpestřena a má realističtější vzhled. K žádným závažným problémům v implementaci nedochází.

Implementovaný systém je připraven k použití v jednoduchých hrách nebo vizualizacích, kde není potřeba vysoké úrovně realističnosti. Budoucí vývoj by mohl klást důraz na zvýšení fyzikální přesnosti kolizí a dělení modelů, správných výpočtů normál a implementace osvětlení. Kolize by mohly být implemetnovány pomocí složitějších datových struktur, například oktalových stromů. Tím by mohly být zajištěny fyzikálně přesné interakce mezi objemy jednotlivých částic. Pro dělení modelů na části by mohl být implementován komplexní algoritmus, jako je uveden v [8]. Takto realizovaný částicový systém by nabyl fyzikální přesnosti a byl tak použitelný pro složitější simulace.

Literatura

[1] Fluid Simulation (3D Graphics). [online], 2009, [cit. 2009-05-11].
URL ⟨http://en.wikipedia.org/wiki/Fluid_Simulation_(3D_Graphics)⟩

[2] Fishwick, P. A.: *Simulation model design and execution: building digital worlds*. Prentice-Hall Inc., 1995, ISBN 0-13-098609-7.

[3] Hargreaves, S.: Depth sorting alpha blended objects. [online], 2009, [cit. 2009-05-11].
URL ⟨http://blogs.msdn.com/shawnhar/archive/2009/02/18/depth-sorting-alpha-blended-objects.aspx⟩

[4] Kršek, P.: *Základy počítačové grafiky - studijní opora*. FIT VUT Brno, 2006.

[5] Martz, P.: Generating Random Fractal Terrain. [online], 1997, [cit. 2009-05-09].
URL ⟨http://gameprogrammer.com/fractal.html⟩

[6] Mason Woo: The OpenGL Programming Guide. [online], [cit. 2009-04-15].
URL ⟨http://www.glprogramming.com/red/⟩

[7] McAllister, D.: Particle System API documentation. [online], 2008, [cit. 2009-05-12].
URL ⟨http://particlesystems.org/Distrib/ParticleHelp_221.html⟩

[8] Miller, A. N.: A Cracking Algorithm for Exploding Objects. [online], 2004, [cit. 2009-02-08].
URL ⟨www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2004/pdf/u1am3.pdf⟩

[9] Reeves, W. T.: Particle Systems—a Technique for Modeling a Class of Fuzzy Objects. *ACM Trans. Graph.*, 1983: s. 91–108, ISSN 0730-0301, doi:http://doi.acm.org/10.1145/357318.357320.

[10] Rodkaew, Y.: Particle systems for plant modeling. [online] Int. Symposium on Plant growth Modeling, simulation, visualization and their Applications, Beijing, China, 2003, [cit. 2009-04-10].
URL ⟨http://rodkaewy.polygondevices.com/ParticleSystemsForPlantModeling_PMA03.pdf⟩

[11] Zhang, J.: An Object-Oriented Particle System for Simulation and Visualization. [online], [cit 2009-05-11].
URL ⟨www.cs.unm.edu/~moore/tr/01-06/particle.pdf⟩

[12] Žára, J.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.

Dodatek A

Obsah CD

Tučně jsou vyznačeny složky obsažené na CD. Kurzívou jsou vyznačeny některé důležité soubory.

- `particle_figther` - demonstrační hra
  - `gfx` - textury
  - `models` - modely
  - `src` - zdrojové kódy
    - `Particle` - rozhraní Particle Systems API
    - `ParticleLib` - zdrojové kódy Particle Systems API
    - `levels` - herní úrovně
  - *README* - instrukce pro kompilaci, instalaci a ovládání
- `text` - text bakalářské práce
  - `fig` - obrázky

Dodatek B

Dostupné částicové systémy

Zde jsou uvedeny některé dostupné částicové systémy v různých podobách.

B.1 Samostatné API

API částicových systémů se ve většině případů samostatně nenachází, protože jsou zabudovány do komplexnějších celků – enginů.

Particle Systems API - open source

Flint Particle System - open source

Quagmire Particle Engine

B.2 Součásti herních enginů

Prakticky každému hernímu enginu nesmí chybět implementace kvalitního částicového systému.

Havoc - komplexní herní engine s kvalitní fyzikou, jehož součástí je i časticový system Havoc FX API

Ageia - engine s částicovým systémem použit ve hře Unreal Tournament 3

Ogre 3D - open source

Irrlicht - open source

B.3 Ostatní

3D modelovací nástroje - Lightwave, Houdini, Maya, XSI, 3D Studio Max, Blender

Simulace tekutin - AfterBurn, RealFlow

SDK - Fork Particle