Abstract

Modelling Moving Geometrical Objects with Particle Systems

In this work, I try to describe techniques used to model various natural phenomena that involve modelling with particle systems. For purpose of this text, particle systems are classified into two general classes: particle systems with and without inter-particle interaction. Simple demonstrating application is also implemented as part of this work, with sample animations created in 3DStudio MAX, that exhibit some of the described particle behavior.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne

Obsah

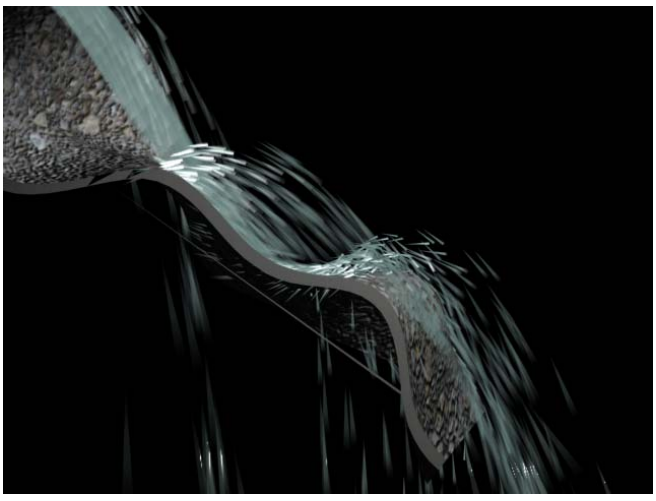
1 Úvod

Již od počátku 80. let se počítačová grafika zabývá modelováním přírodních procesů a jevů, které je obtížné napodobovat klasickými metodami povrchové reprezentace objektů. Modelováním takových jevů, které nemají přesně definovaný povrch, popsatelný pomocí klasické geometrie, ale naopak složitý, nepravidelný, „rozmazaný“. V té době byly učiněny první pokusy modelovat přírodní efekty pomocí částic, ale jednalo se pouze o statické systémy, například hvězdy a mlhoviny nebo model Saturnových prstenců [3], které stále nevykazovaly žádné dynamické chování. Při natáčení filmu Star Trek II: The Wrath of Khan [2], William Reeves pracoval na scéně Genesis, ve které je zachyceno přetvoření pusté planety na živou [1]. V této počítačem vytvořené scéně je zobrazena série mohutných planetárních explozí, které jsou právě modelovány dynamickými částicovými systémy. Reeves si uvědomil, že stačí, když je pohyb částic v systému popsán pouze jednoduchými stochastickými procesy (které do systému vnášejí náhodné chování) a v systému vzniká chaotické, složité chování, ale přitom je zároveň stále možné kontrolovat souhrné vlastnosti systému.

V klasické podobě je částicový systém definován jako skupina (množina) částic, kde částice je bod v trojrozměrném prostoru, který kromě své polohy má případně nějaké další vlastnosti, například barvu, rychlost apod. Na tento systém působí procesy, které v čase mění nějaký z atributů částic nebo odebírají částice ze systému či jej do něj přidávají.

Částice v rámci částicového systému svými polohami v prostoru vyplňují objem modelovaného objektu, vznikají a zanikají, mění se jejich poloha, barva, působí na ně okolní síly prostředí (např. gravitace), můžou interagovat s okolím ve formě kolizí a tak dále. Chování částic je chaotické, na počátku je definován přibližný směr pohybu částic, během simulace na částice působí síly, mění směr a rychlost jejich pohybu.

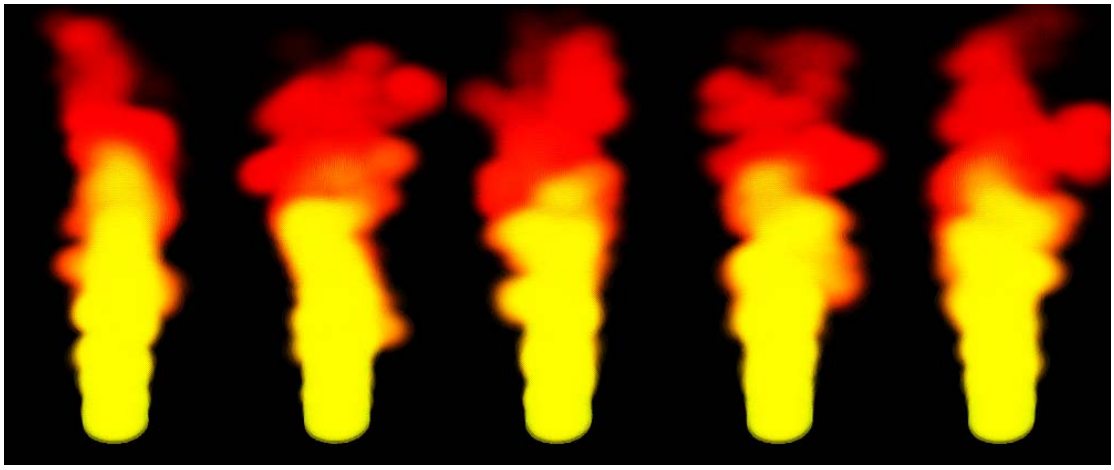
Takto definované částicové systémy jsou vhodné k simulování skupiny přírodních jevů jako je oheň, kouř, mraky, exploze, tekoucí voda a podobně.



Obr. 1.1 Simulace vodopádu

Příkladem takovéto simulace může být třeba model vodopádu (viz obr. 1.1). Částice, které reprezentují tekoucí vodu, jsou generovány v určitém regionu, představujícím vrchol vodopádu. Jejich počáteční rychlosti a směry pohybu je přibližně stejné, liší se pouze o malou náhodnou hodnotu. V průběhu simulace se pak tato odchylka projeví postupným rozptylováním částic, čímž je věrohodně napodobeno rozstřikování vody ve vodopádu. Na

částice působí vnější síla, gravitace, která způsobí, že částice jsou postupem času zrychlovány směrem dolů. Dále probíhá interakce s okolním prostředím, geometrickými objekty, od kterých se částice odráží a ještě více se rozptylují do prostoru, tak jako se rozstřikuje padající voda, když naráží na kameny. Poté, co částice klesnou pod nějakou referenční rovinu (kterou můžeme považovat třeba za hladinu jezera, na kterou vodopád dopadá), jsou ze systému odstraněny, takzvaně „umírají“.



Obr. 1.2 Simulace hoření pomocí částicového systému

Jiným, obdobně klasickým příkladem je simulace ohně a celkového procesu hoření (obr. 1.2). Částice opět vznikají v lokálním regionu (často v jediném bodě) a na počátku dostávají náhodnou rychlost. Na částice působí síla, která je táhne nahoru, takže částice stoupají, ale zároveň se i rozptylují vlivem odlišných počátečních rychlostí. Barva každé částice se mění v čase od jejího vzniku, například tak, že na počátku je žlutá, postupně přechází do červené a nakonec přechází do černé nebo šedé. Tímto způsobem je simulována změna barvy plamene a zároveň je takto zobrazen i vznikající kouř. Nakonec, například po dosažení určitého stáří (doby setrvání v systému) nebo vystoupání do určité výšky, je částice odstraněna ze systému.

Z uvedených příkladů je vidět, že možností, jak je možné ovlivňovat chování částic a jejich vlastnosti, je nepřeberné množství. Navíc částice v systému se nemusí chovat vždy nezávisle na ostatních částicích, vznikají pak další možnosti využití částicových systémů k modelování a simulaci daleko složitějších jevů, než jaké jsou popsány na začátku tohoto úvodu. Tato varianta však s sebou přináší některé významné komplikace, hlavně algoritmického rázu.

Cílem této práce je prozkoumat možnosti použití a způsoby implementace částicových systémů pro účely modelování pohybu geometrických objektů. První část, teoretická, se zabývá vlastnostmi částicových systémů, různými typy částicových systému a problémy, které nastávají při jejich implementaci. Ve druhé, praktické části popisuji ukázkovou aplikaci, která využívá některé postupy naznačené v první části, a dále se pokouším zhodnotit zkušenosti, které jsem získal při implementaci částicového systému.

2 Teoretická část

Z důvodů popsaných na konci úvodního odstavce, se tomto textu pokusím rozdělit popisy využívaných částicových systémů podle závislosti či nezávislosti chování částic na ostatních částicích a potom uvedu některé speciální případy částicových systémů.

2.1 Systémy nezávislých částic (klasické)

Tato původní forma částicových systémů je i v dnešní době tou nejvíce využívanou. Její největší použití je stále v oblasti, ve které tyto částicové systémy vznikly, a to v oblasti vizuálních efektů. Snad každý animační či trikový systém v dnešní době používá nějakou implementaci částicových systémů. Z původního zobrazování částic jako bodů, případně čar, jsou dnes částice zobrazovány ve jako bitmapy či jako složitější geometrické objekty, od koulí a krychlí až po modely postav a podobně (viz dále odstavec Způsoby zobrazení). Dále jsou částicové vizuální efekty využívány v čím dál větším počtu počítačových her, průkopníkem v tomto ohledu grafického zpracování byla hra Quake od firmy id Software [4].

2.1.1 Princip

Základní myšlenkou metody je to, že určité přírodní jevy mohou být simulovány tak, že velkému počtu jednotlivých částic je určen obecný předpis, který kontroluje pohyb a další vlastnosti každé jednotlivé částice při přechodu z jednoho snímku animace do dalšího. Takový proces má některé atributy zpracování jednotlivých částic nějakým způsobem parametricky nastavitelné, navíc se do zpracování částic zavádí prvek náhody, který způsobuje, že každá částice se chová trochu odlišně než ostatní. Jelikož částic v systému je velký počet, jsou jednotlivé částice často zobrazovány jako jednoduchá obrazová primitiva, například jako obrazové body (pixely). Výsledkem jsou animované objekty, jejichž chování je parametricky nastavitelné a které při vhodné volbě předpisu a nastavení jeho parametrů zpracování částic můžou být použity k modelování rozmanitých přírodních jevů, například, jak už bylo řečeno, hoření, mraků, vodopádů a podobně.

Dobrým příkladem toho, jak funguje animace pomocí částicových systémů je původní Reevesův popis vytvoření snímku animace, tak jak je popsán v [1]:

1. Jsou vygenerovány nové částice a vloženy do část. systému.
2. Každé nové částici jsou přiřazeny její počáteční vlastnosti.
3. Všechny částice, jejichž stáří (doba setrvání v systému) přesáhne maximální velikost, jsou ze systému odstraněny.
4. Všechny přítomné částice jsou posunuty podle předpisu (a jejich další vlastnosti jsou také změněny).
5. Přítomné částice jsou zobrazeny.

V typické implementaci částicového systému jsou částicím přiřazovány následující počáteční hodnoty atributů:

1. počáteční pozice
2. počáteční velikost rychlosti a směr pohybu
3. počáteční velikost
4. počáteční průhlednost
5. zobrazovaný tvar

6. zobrazovaná textura (příp. barva)

Počáteční hodnoty jsou nejčastěji generovány s nějakou mírou náhodnosti, většinou pomocí jednoduchého vztahu pro veličinu X, typu:

$$InitX = StredniX + \text{Rand}( ) * VarX$$

Kde *StredniX* je střední hodnota veličiny, *VarX* je variabilita této veličiny, Rand( ) je procedura vracející náhodné číslo s nějakým (např. konstantním) rozložením a *InitX* je výsledná počáteční hodnota veličiny X.Takže například počáteční velikost rychlosti může být vypočítána předpisem individuálně pro každou vznikající částici:

$$InitRychlost = StredniRychlost + \text{Rand}( ) * VarRychlost ,$$

kde *StredniRychlost* je střední hodnota rychlosti generovaných částic v daném systému, *VarRychlost* je variabilita rychlosti částic, a *InitRychlost* je vypočtená počáteční rychlost generované částice. To je příklad jednoduchého předpisu pro skalární veličinu, vektorové atributy mohou být inicializovány ještě mnohem rozmanitěji, typickým příkladem jsou možné předpisy počáteční pozice částice. Od pozice generované v objemu či na povrchu různých geometrických primitiv jako jsou úsečky, čtverce, koule apod. až po využití složitějších povrchově reprezentovaných objektů a procedurálně definovaných objektů (fraktály či L-systémy). (viz např. [5], [11])

2.1.2 Dynamika částic

Většina aplikací částicových systémů je nějakou formou modelování pohybujících se objektů, a proto důležitou kapitolou částicových systém je pohyb částic. Existuje nepřeberné množství možností, jak rozpohybovat částice, od primitivního lineárního pohybu částic až po komplexní simulaci fyzikální interakce s okolím, výpočet odrazů od okolních objektů a podobně.

Nejjednodušší pohyb částice lze realizovat, jak už bylo naznačeno, pomocí jednoduchého předpisu, kdy poloha je funkcí času, čili například pohyb po přímce. Pro velmi jednoduché efekty tato realizace stačí, ovšem většinou je požadováno komplexnější chování, alespoň působení gravitace či jiné síly. V tom případě se v systému zavádí vnější síly, působící na částice, a rychlost a poloha částic se vypočítává z pohybových rovnic, ve kterých je vyjádřen vliv těchto vnějších sil.

Obecně pohybové rovnice vypadají takto:

$$\mathbf{v} = \mathbf{v}_0 + \int \mathbf{a} dt$$

$$\mathbf{p} = \mathbf{p}_0 + \int \mathbf{v} dt$$

kde $\mathbf{p}$ je poloha částice, $\mathbf{v}$ je vektor rychlosti částice, $\mathbf{p_0}$ a $\mathbf{v_0}$ jsou počáteční polohy částic a zrychlení $\mathbf{a}$ je podíl výsledné vnější síly $\mathbf{f}$ působící na částici a hmotnosti částice *m*.

$$\mathbf{a} = \frac{\mathbf{f}}{m}$$

Pro účely simulace pohybu částicových systémů můžeme řešení těchto rovnic aproximovat pro jeden krok simulace, například pomocí Eulerovy metody:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}\Delta t$$

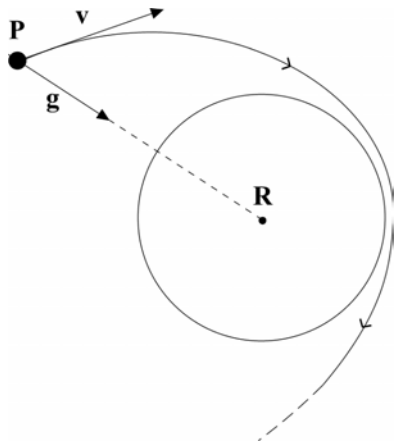
$$\mathbf{p}_{n+1} = \mathbf{p}_n + \mathbf{v}_{n+1}\Delta t$$

kde Δt je časový krok simulace.

Nejčastější silou, která se zavádí v částicovém systému, je gravitace. Je charakterizována směrovým vektorem **d** a gravitační konstantou *g* a intenzita, s jakou působí na částici, je také závislá na hmotnosti částice. Výsledná gravitační síla **g** je tedy definována takto:

$$\mathbf{g} = mg\mathbf{d}$$

Vizuálním výsledkem působení gravitace na částici je parabolický pohyb částice. Směr gravitace nemusí být nutně v prostoru všude stejný, gravitace může směřovat do jednoho či více center, částice pak mají tendenci obíhat tato centra po orbitě (viz obr. 2.1).



Obr. 2.1 Působení sil a pohyb částice v okolí centra gravitace **R**

Další silou, která může působit na částice v systému, je odpor prostředí. Její velikost je závislá na rychlosti, jakou se částice pohybuje, a působí směrem opačným ke směru pohybu částice. Lze ji popsat rovnicí

$$\mathbf{f}_r = -k_r \mathbf{v}$$

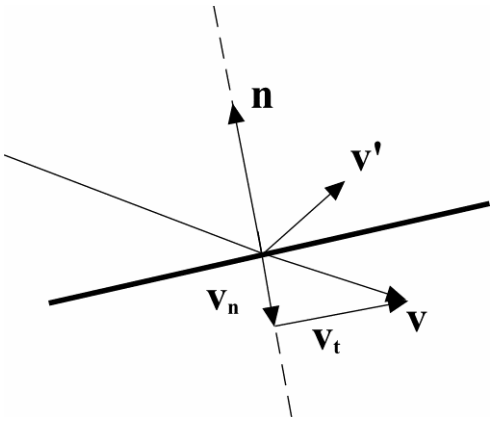
kde k_r je konstanta odporu prostředí. Její efekt působí tak, že zpomaluje pohyb částice a pokud v systému nepůsobí žádná jiná další síla, může dojít až k zastavení pohybu částice.

Častým požadavkem na chování částicového systému je také odrážení částic od geometrických objektů. Detekce kolizí a odpovídající odraz je obecně komplikovaný problém, v případě bodových částic jej však lze podstatně zjednodušit. Vezměme například odrážení částic od polygonální plochy.Podle techniky popsané např. v [6] můžeme kolizi detekovat tak, že dosadíme polohu částice do rovnice roviny polygonu. Pokud částice pronikla povrchem, rychlost částice po odrazu dostaneme ze vztahu

$$\mathbf{v}' = \mathbf{v} - 2(\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$

kde $\mathbf{v}'$je vektor rychlosti částice po odrazu, **v** je původní vektor rychlosti a **n** je normálový vektor odrazové plochy (viz. obrázek 3.)

Toto je nejjednodušší metoda výpočtu odrazu částice, částice se takto odráží až z polohy pod povrchem, do které pronikla během jednoho kroku simulace, což může při větším simulačním kroku vypadat nepřirozeně. Možným řešením je lineární interpolací dopočítat z předchozí polohy částice přesnější polohu odrazu a tu použít k výpočtu nové rychlosti.



Obr. 2.2 Schéma výpočtu rychlosti při elastické srážce

Dalším zpřesněním je zavedení elastické srážky, kdy rychlost po odrazu rozdělíme na normálovou a tečnou složku, $\mathbf{v}_n$ a $\mathbf{v}_t$ (viz obr. 2.2), které spočteme

$$\mathbf{v}_n = (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$

$$\mathbf{v}_t = \mathbf{v} - \mathbf{v}_n$$

a nová rychlost po srážce je rovna

$$\mathbf{v}' = (1 - \mu)\mathbf{v}_t - \varepsilon\mathbf{v}_n$$

kde koeficient tření μ zmenšuje tečnou složku a velikost normálové složky je ovlivněna koeficientem pružnosti ε, kterým je modelována ztráta kinetické energie při odrazu.

2.1.3 Odebírání částic ze systému

V případě, že do systému jsou v průběhu simulace přidávány nové částice, často vzniká požadavek na odstranění částic, které podle nějakého kritéria jsou pro účely modelování jevu již nezajímavé. Důvodem je zejména zvyšující se náročnost výpočtu s rostoucím počtem částic a dále také rostoucí paměťové nároky. Paměť, kterou zabíraly odebrané částice je možné opět použít k tvorbě nových částic. V závislosti na modelované situaci je třeba definovat podmínky, za nichž budou částice odebírány ze systému.

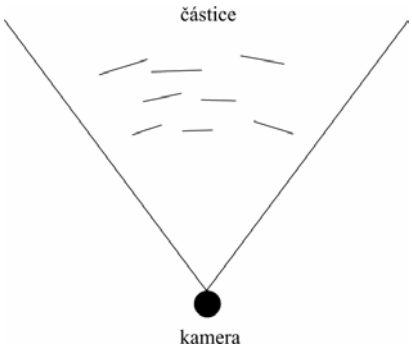
Jedním z nejčastěji používaných způsobů je zavedení *věku* částice. Realizováno je to tak, že každá částice má atribut věku, a v okamžiku přidání částice je tento atribut vynulován. Při každém kroku simulace se věk částice zvyšuje o délku tohoto kroku, a ve chvíli kdy přesáhne určitou předdefinovanou hodnotu (často nazývanou délka života angl. *lifetime*), je částice ze systému odstraněna.

V některých specializovaných simulacích, například u simulace vodopádu, je možné částice odebírat ze systému podle jejich polohy v prostoru, na příkladě vodopádu je to v případě, kdy proniknou rovinou, která představuje hladinu vody.

2.1.4 Způsoby zobrazení částic

Ačkoli je částice pouze bod umístěný v prostoru, existuje množství způsobů, jak tento bod interpretovat a částici zobrazit. Nejjednodušší řešení se nabízí samo, a to je zobrazení částice jako bodu na obrazovce. To je ovšem pro většinu aplikací nedostačující, pro modelování většiny jevů by bylo třeba ohromné množství částic k tomu, aby výsledek

vypadal realisticky. Proto se na místě částic vykreslují složitější objekty, od čar přes jednoduché polygony, texturované obdélníky (tzv. „sprites“) až po plně třírozměrné objekty. Často jsou používané právě texturované obdélníky, které i při pohybu částic zůstávají neustále kolmo natočeny na pozorovatele, aniž by zobrazovaná bitmapa byla nějak deformována transformacemi částic. Tato technika se anglicky nazývá „billboarding“ (na obr. 2.3). Zajímavým problém vzniká při užití některého z grafických rozhraní typu OpenGL, které má svůj vlastní systém modelových transformací a při použití těchto transformací je třeba nějakým způsobem zachovat natočení obdélníků kolmo na kameru.



Obr. 2.3 Natočení částic při použití techniky billboardingu

V poslední době se zobrazení částicových systémů neomezuje pouze na body, čáry či bitmapy, ale často jsou částice zobrazovány jako složité geometrické objekty, například jako pohybující se zvířata, ptáci nebo postavy lidí při simulacích hejna či davu (viz kapitola 4).

2.2 Systémy se vzájemnou interakcí částic

V některých případech modelování a simulace jsou využívány takové systémy částic, ve kterých se částice nějakým způsobem ovlivňují. Toto vzájemné působení může být různého druhu, asi nejčastější je vzájemné silové působení, např. přitahování a odpuzování nebo detekce kolizí a následné odrazy mezi částicemi. Některé další vzájemné interakce se používají například v simulacích chování hejna či davu, v tomto textu je těmto případům věnována samostatná kapitola (viz. kap. 4). Některé problémy a techniky jako odebírání částic, zobrazování atd. jsou samozřejmě obdobné jako u systémů nezávislých částic.

2.2.1 Vzájemné silové působení

Asi nejjednodušším příkladem silového působení mezi částicemi jsou přitažlivé a odpudivé síly. Přitažlivá síla táhne částice k sobě navzájem, naopak odpudivá síla je tlačí od sebe. Odpudivými silami můžeme zabránit kolizím částic mezi sebou, přitažlivé síly lze využít při simulaci gravitačního působení, například planetární soustavy.
Pro dvojici částic, které jsou umístěny na pozicích **a** a **b**, síla působí ve směru vektoru **r** = **b** – **a**, pro velikost přitažlivé síly můžeme použít například gravitační zákon

$$f = g\frac{m_a m_b}{\|\mathbf{r}\|^2}.$$

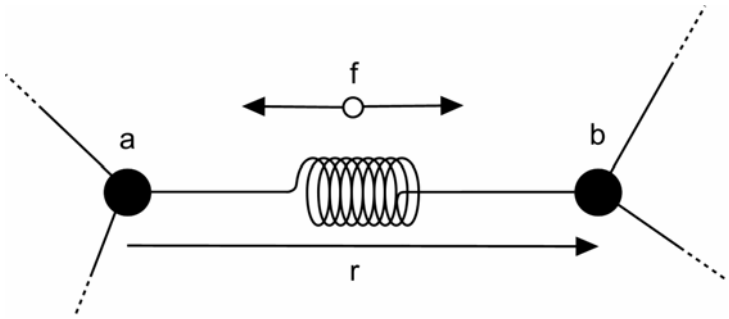
kde g je gravitační konstanta, m_a a m_b jsou hmotnosti částic, takže přitažlivá síla je definována

$$\mathbf{f} = g\frac{m_a m_b}{\|\mathbf{r}\|^3}\mathbf{r}.$$

Pokud u síly změníme směr na opačný, stává se z ní síla odpudivá. Vzorec se změní pouze přidáním záporného znaménka.

$$\mathbf{f} = -g\frac{m_a m_b}{\|\mathbf{r}\|^3}\mathbf{r}\,.$$

2.2.2 Spring-mass systémy



Obr. 2.4 Pružina mezi dvěma částicemi

Částice je možné použít k modelování objektů, jejichž tvar se během simulace mění působením vnějších vlivů prostředí. K tomu se využívají takzvané spring-mass systémy. Myšlenka je taková, že jednotlivé částice představují hmotné body a porůznu jsou pospojovány pružinami (obr. 2.4), tak jak je popsáno např. v [7]. Každá pružina má nějakou klidovou délku *l*, což je délka pružiny ve stavu, kdy na systém nepůsobí žádná vnější síla. Pokud se pružina natáhne (a nebo naopak smršťí), působí na částice v pozicích **a** a **b**, které spojuje, síla ve směru **r** = **b** – **a**. Velikost této síly popisuje Hookeův zákon

$$f = -k_s\left(\|\mathbf{r}\| - l\right)$$

pak

$$\mathbf{f} = -k_s(\|\mathbf{r}\| - l)\frac{\mathbf{r}}{\|\mathbf{r}\|}\,.$$

kde k_s konstanta pružiny, která udává tuhost pružiny a l je klidová délka pružiny. Podle tohoto zákona je síla, kterou pružina přitahuje obě částice k sobě úměrná délce natažení pružiny, naopak pokud se pružina smršťí na délku menší než je její klidová délka, působící síla je odpudivá (má opačné znaménko). Pokud bychom vytvořili částicový systém, který by se choval jen podle tohoto zákona, po skončení působení vnější síly by jeho pružina kmitaly donekonečna s nezměněnou amplitudou. To je vyřešeno tak, že do výpočtu síly se zavádí tlumení. Velikost tlumení je úměrná rychlosti relativního pohybu částic, zavádíme člen $\dot{\mathbf{r}} = \dot{\mathbf{b}} - \dot{\mathbf{a}}$, kde $\dot{\mathbf{a}}$ a $\dot{\mathbf{b}}$ jsou relativní rychlosti částic. Pak velikost síly s tlumením je dána

$$f = -(k_s(\|\mathbf{r}\| - s) + k_d\frac{\dot{\mathbf{r}}\cdot\mathbf{r}}{\|\mathbf{r}\|}))$$

pak

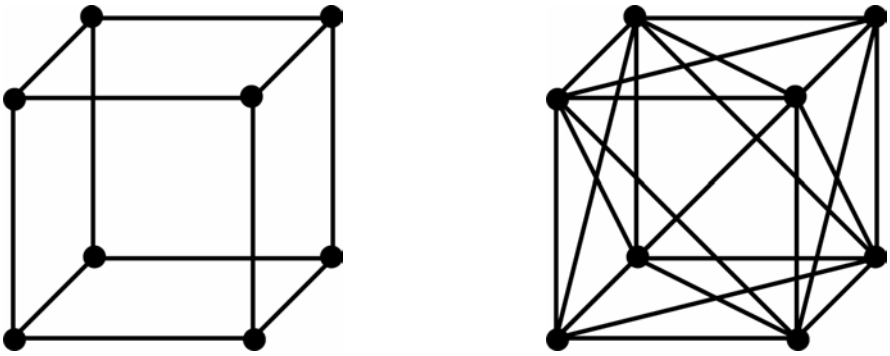
$$\mathbf{f} = -(k_s(\|\mathbf{r}\| - s) + k_d\frac{\dot{\mathbf{r}}\cdot\mathbf{r}}{\|\mathbf{r}\|}))\frac{\mathbf{r}}{\|\mathbf{r}\|}$$

Pokud na těleso, ve kterém jsou pružiny s tlumením, nepůsobí žádná vnější síla, pak se pružiny působením tlumení postupně dostávají do klidové délky.

Ve většině aplikací spring-mass systémů je pouhé působení částic v systému na sebe navzájem nedostačující a je vyžadována i nějaká interakce s okolím. Nejčastěji se jedná buď o

silové působení nebo detekci srážek. Druhů silového působení může být více, od lokálního působení sil na některé uzly systému až po silová pole, například vírová apod. Můžou být detekovány srážky, nejčastěji s jednoduchými geometrickými primitivy typu plocha, kvádr či koule a tvar složitějších objektů je aproximován pomocí těchto primitiv.

Pokud chceme použít spring-mass systém k simulaci deformace geometrického objektu, jehož tvar již máme definován, objevuje se před námi problém, jak vytvořit odpovídající spring-mass systém, který zároveň by odpovídal geometrii našeho objektu a přitom jeho fyzikální chování bude odpovídat tomu, jaké bychom od našeho objektu očekávali. Ve většině dnešních aplikací je geometrický objekt definován jako síť trojúhelníků a nabízí se řešení použít vrcholy trojúhelníků jako uzly spring-mass systémů a hrany trojúhelníků jako spojující pružiny. V tomto případě, jelikož trojúhelníková síť je povrchovým popisem objektu, dostaneme fyzikální model dutého objektu, který má silnou tendenci zkolabovat při nárazu na plochu (příklad na obr. 2.5). Je třeba si uvědomit, že pružiny omezují pohyb uzlu pouze po směru orientace pružiny, ale ne už ve směrech kolmých k směru orientace, proto k tomu, aby uzly byly stabilní a model stále udržel přibližně svůj původní tvar, musí být uzly vázány více pružinami z různých směrů.



Obr. 2.5 Nestabilní pružinový model krychle (vlevo) a stabilní verze (vpravo)

Existuje více možností, jak model „vyztužit“, většinou je ale nejvýhodnější model ručně upravit, tak aby se choval co nejreálněji. Další možností, používanou v některých komerčních 3D modelovacích aplikacích, je vytvoření třírozměrné deformační sítě vrcholů, uspořádaných do krychliček, tak aby přibližně obepínala geometrický model tělesa a vyplňovala objem tělesa(viz. obr.). Vrcholy této sítě pak tvoří uzly spring-mass systému a hrany, kterými jsou vrcholy pospojovány, fungují jako pružiny. Při fyzikální simulaci je pak deformována nejdříve tato síť a poté je vlastní geometrický model deformován pomocí techniky tzv. FFD deformace (o FFD blíže se lze dočíst např. v [8]), podle toho jak se změnila deformační síť (obr. 2.6).

Obr. 2.6 Deformace objektu po dopadu pomocí pružné FFD sítě

2.2.3 Vlastní výpočet simulace

Hlavní znakem simulace dynamického chování objektů na počítačích je to, že čas není v simulaci chápan jako spojitá veličina, ale zvětšuje v diskrétních skocích, takže simulace je vypočítávána v jednotlivých snímcích v určitým časovým odstupem. Důvodem je skutečnost, že většina veličin v dynamické simulaci je popsána diferenciálními rovnicemi nebo jejich soustavami a pokud není model velmi jednoduchý, je analytické (časově spojité) řešení velmi složité, většinou přímo nemožné. Proto se k výpočtu simulace neužívají metody analytické, ale numerické a všechna numerické metody řešení diferenciálních rovnic nějakým způsobem diskretizují čas.

Vezměme jako příklad spring-mass částicový systém, ve kterém navíc působí gravitace a odpor okolního prostředí.

Rychlost částice **u** je definována pomocí rovnice

$$\mathbf{v} = \dot{\mathbf{u}} = d\mathbf{u} / dt$$

změna rychlosti (zrychlení) je definováno rovnicí

$$\mathbf{a} = \ddot{\mathbf{u}} = d\mathbf{v} / dt$$

podle Newtonovské mechaniky se zrychlení rovná

$$\mathbf{a} = d^2\mathbf{u} / dt^2 = \mathbf{f} / m$$

kde *dt* je velikost časového kroku, **f** je výsledný vektor sil působících na částici a *m* je hmotnost částice. Na všechny částice působí vnější gravitační síla **g**, která závisí na hmotnosti částice

$$\mathbf{g} = gm\mathbf{d}$$

kde *g* je gravitační konstanta, a jednotkový vektor **d** je směr kterým působí gravitace (čili směr dolů v daném systému). Dále zde působí odpor prostředí **e** způsobený viskozitou prostředí, který roste úměrně s rychlostí pohybu částice podle rovnice

$$\mathbf{e} = -e\dot{\mathbf{u}}$$

kde *e* je viskozita okolního prostředí. Celková vnější výsledná síla $\mathbf{f_e}$ působící na částici je součtem gravitační síly a odporu prostředí

$$\mathbf{f}_e = \mathbf{g} + \mathbf{e}$$

Dále, jelikož mezi částicemi jsou pružiny, je třeba také započítat silové působení pružin mezi částicemi. Pružina v systému vyvine sílu, která je úměrná jejímu zkrácení či prodloužení oproti klidové délce, tuto sílu můžeme popsat pomocí Hookeova zákona takto (viz. kap. 2.2.2)

$$\mathbf{f}_s = -\left(k_s\left(\|\mathbf{r}\| - s\right) + k_d \frac{\dot{\mathbf{r}} \cdot \mathbf{r}}{\|\mathbf{r}\|} \right) \frac{\mathbf{r}}{\|\mathbf{r}\|}$$

zde vektor $\mathbf{r} = \mathbf{u}_b - \mathbf{u}_a$ je definován polohami částic $\mathbf{u_a}$ a $\mathbf{u_b}$ na koncích pružiny, vektor $\dot{\mathbf{r}} = \dot{\mathbf{u}}_b - \dot{\mathbf{u}}_a$ je rozdíl rychlostí částic, k_s je konstanta tuhosti pružiny a k_d je konstanta tlumení pružiny. Na částici *a* pak pružina působí výslednou silou $\mathbf{f}_{sa} = \mathbf{f}_s$ a na částici *b* působí silou v opačném směru $\mathbf{f}_{sb} = -\mathbf{f}_s$.

Celková výsledná síla působící na částici *i* je tedy součtem všech sil od pružin s_j pro $j = 1..n$, kde *n* je celkový počet pružin spojených s částicí, a vnějších sil prostředí, takže

$$\mathbf{f}_i = \sum_{j=1}^{n} \mathbf{f}_{s_j i} + \mathbf{f}_{ei} = \sum_{j=1}^{n} \mathbf{f}_{s_j i} + \mathbf{g}_i + \mathbf{e}_i$$

Pro numerický výpočet zvolíme nějaký časový krok *h*, který udává přesnost, s jakou budeme aproximovat řešení diferenciální rovnice. Nejjednodušší, ale také nejméně přesnou metodou je Eulerova metoda. Vzorec pro výpočet rychlosti částice v čase $t + h$ vypadá takto:

$$\mathbf{v}_{t+h} \approx \mathbf{v}_t + h\mathbf{a}_t$$

podobně výpočet polohy částice

$$\mathbf{u}_{t+h} \approx \mathbf{u}_t + h\mathbf{v}_t$$

Při vyšší konstantě tuhosti pružin dochází při použití Eulerovy metody k rychlé destabilizaci řešení a systém začne nepřirozeně oscilovat. V takovém případě je možné buď zmenšit krok simulace, což se ovšem může významně zvýšit složitost výpočtu, anebo sáhnout po nějaké dokonalejší integrační metodě. Tou může být například metoda Runge-Kutta 2. řádu anebo některá z implicitních. Blíže o problematice numerického řešení diferenciálních rovnic se lze dočíst např. v [9].

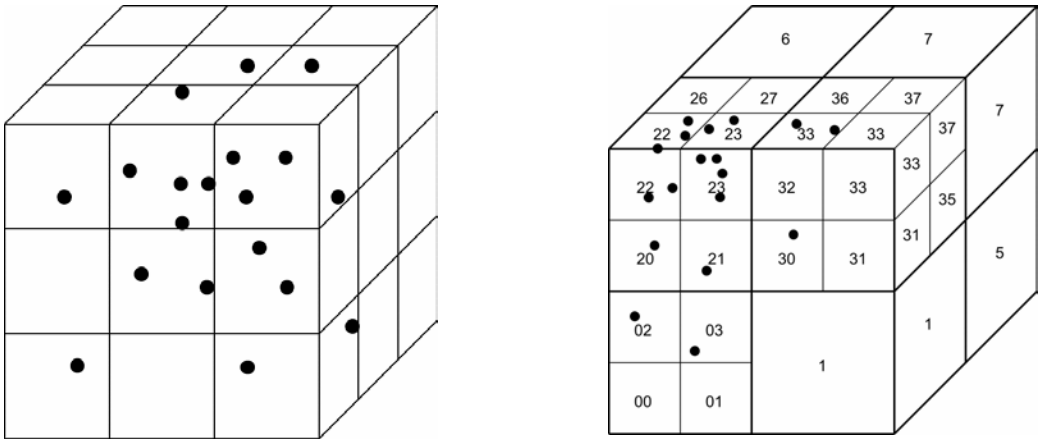
Shrneme-li výše uvedené postupy, pak algoritmus jednoho kroku výpočtu simulace by mohl vypadat zhruba takto:

1. vypočti výsledné působení vnějších sil na každou částici
2. vypočti působení pružinových sil pro každou dvojici částic spojených pružinou
3. vypočti aproximaci rychlosti a polohy v dalším časovém kroku
4. aktualizuj polohu částic (případně další vlastnosti)
5. zvyš čas o časový krok

Poznamenejme nakonec, že například u her, které používají nějaký fyzikální model, často nemusí být nutně časový krok výpočtu nějak svázán s rychlostí zobrazování snímků (frame-rate), nýbrž běží nezávisle, obyčejně pomaleji, neboť je většinou dost náročný, navíc fyzikální model by neměl být nějak závislý na rychlosti procesoru.

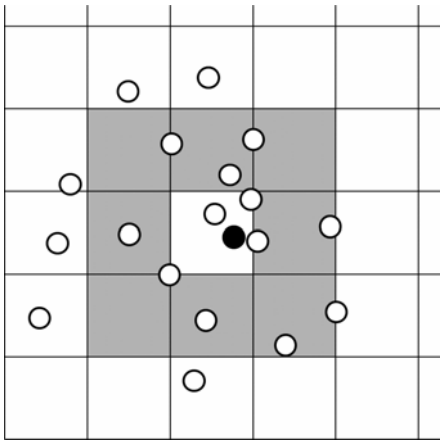
2.2.4 Algoritmická složitost výpočtu

V případě částicového systému bez vzájemných interakcí je algoritmická složitost zpracování všech částic přibližně O(n), kde *n* je celkový počet zpracovávaných částic. Pokud každá částice interaguje se všemi ostatními částicemi v sytému vzroste složitost výpočtu na $O(n^2)$, což při vyšším počtu částic bývá neúnosné, zejména pokud simulace má probíhat v reálném čase. Proto se používají různé techniky pro zjednodušení výpočtu, většinou založené na nějakém typu koherence, nejčastěji prostorové. Často se využívá dělení prostoru do segmentů konstantní velikosti nebo je možné použít hierarchickou strukturou jako např. oktalový strom (obr.2.7), blíže o oktalovém stromu se lze dočíst např. v [8].



Obr. 2.7: Konstantní dělení prostoru (vlevo) a dělení prostoru oktalovým stromem (vpravo)

Například detekce kolizí mezi částicemi při konstatním dělení se provádí tak, že testujeme kolize s částicemi ve stejném segmentu, ve kterém je právě zpracovávaná částice a dále ze sousedních segmentů, neboť některé částice můžou svým rozsahem přesahovat i do právě zpracovávaného segmentu, jak je ukázáno na obr. 2.8.



Obr 2.8: Přesah částic ze sousedních segmentů

Samotný problém detekce kolizí je ovšem ještě složitější, například při vysoké rychlosti pohybu částic se může stát, že částice, jejichž dráhy se protínají, se můžou „přeskočit“, čili v žádném kroku simulace nedojde k průniku jejich objemu. Z toho si lze představit, že jen samotná problematika detekce kolizí je velmi široké a složité téma, které přesahuje rozsah tohoto textu.

2.3 Další druhy částicových systémů

Vedle dosud ukázaných způsobů využití částicových systémů ještě existují další aplikace, ve kterých je výhodné použít zobrazení skupiny objektů pomocí částic a kontrolu jejich parametrů pomocí jednoduchého předpisu, tedy postupů charakteristických pro částicové systémy. Jednou z těchto aplikací je simulace složitého chování stáda či hejna, tak jak ji ve své práci poprvé popsal Craig W. Reynolds [10].

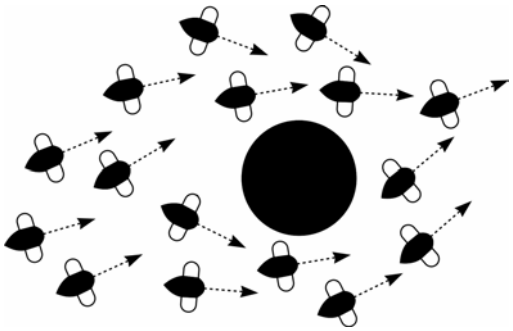
Při vytváření animace chování například hejna ptáků nastává ten problém, že hejno se skládá z mnoha pohybujících se ptáků, z nichž každý se pohybuje po vlastní často dost složité

dráze, a animovat ručně tento pohyb pro každého jedince by bylo velice náročné a pro praktickou práci neúnosné. Základem metody použití částicových systémů je ta myšlenka , že chování každého jedince v rámci hejna lze popsat relativně jednoduchým předpisem, který je definován několika pravidly chování, v podstatě podobným těm, kterými se řídí jedinec v hejnu v reálné přírodě. Pravidla, která Reynolds použil ve své práci, jsou tato:

1. *Vyhýbání se kolizím*: každý jedinec se snaží vyhnout srážce s ostatními jedinci v hejnu
2. *Přizpůsobení se pohybu*: každý jedinec se snaží srovnat směr a rychlost svého pohybu s jedinci ve svém nejbližším okolí
3. *Držení se v blízkosti hejna*: každý jedinec se snaží držet blízko jedinců ve svém okolí

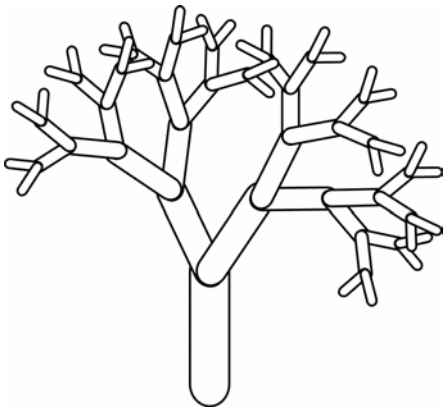
Vyhýbání se kolizím je nutkání změnit směr svého pohybu v případě že hrozí srážka s jiným jedincem. Toto chování je řízeno pouze pozicemi okolních jedinců. *Přizpůsobení se pohybu* je potřeba letět stejným směrem jako jedinci v okolí a tak v podstatě opět působí jako prevence proti srážkám s ostatními. *Držení se v blízkosti hejna* nutí jedince být v centru hejna a protože v Reynoldsově pojetí má jedinec pouze lokalizované vnímání, je to vlastně směřování do centra blízkých jedinců. Pokud je jedinec uvnitř hejna, kde je hustota rozmístění jedinců přibližně konstantní, centrum jeho blízkých jedinců je vlastně přibližně v místě, kde se zrovna nachází, takže nemá potřebu nijak v tomto ohledu měnit svou polohu.

Souhrnný efekt takto definovaného chování jednotlivých členů hejna ve výsledku velmi připomíná chování hejna ptáků v přírodě. Pokud je navíc realizována i detekce okolních překážek, hejno se při kolizi rozdělí, obteče překážku a za překážkou se opět spojí dohromady (viz obr. 2.9).



Obr 2.9: Chování hejna v okolí překážky

Částicové systémy je možné používat nejen k zobrazení pohybujících se objektů, ale lze je využít i k vyobrazení objektů se složitou strukturou, jejichž popis pomocí ploch by byl zbytečně komplikovaný a výpočetně náročný. V tomto ohledu se uplatňují při zobrazování modelů rostlin, které byly předtím vygenerovány nějakou používanou metodou, například pomocí L-systémů (příklad na obr. 2.10). Tato technika využívává přepisování řetězců (posloupnosti symbolů) podle určitého systému pravidel, kdy tyto řetězce jsou poté geometricky interpretovány nějakým způsobem. Pro bližší vysvětlení viz. např. [5], [11] a [12].



Obr. 2.10: Model stromu vytvořený pomocí L-systému

2.4 Implementace částicových systémů

Pokud se rozhodneme použít v aplikaci částicový systém, pak jeho implementace s sebou přináší rozmanité problémy, které je možné řešit více způsoby. Volba vhodného řešení záleží jednak na zvoleném typu částicového systému a dále pak na způsobu jeho použití. Například pokud navrhujeme částicový systém pro aplikaci, která je určena k vytváření vizuálních efektů pro film a animaci, použijeme spíše co nejpřesnější metody výpočtu fyzikálního modelu simulace, bez ohledu na zpomalení výpočtu scény, který to s sebou přináší. Naopak při použití pro modelování v reálném čase, například ve hře, je rozhodujícím faktorem rychlost výpočtu a na přesnosti příliš nezáleží, v tomto případě použijeme pouze primitivní metody, případně je fyzikální modelování nahrazeno předtím vypočítanou animací, která pouze přibližně ukazuje žádaný vizuální efekt. Aspekty implementace probrané v této kapitole se týkají zejména vhodných datových struktur a práce s pamětí.

2.4.1 Vhodné datové struktury

Informace o jednotlivé částici je většinou uložena v datové struktuře, která je navržena a přizpůsobena zvlášť pro modelování daného jevu. Například částice ve spring-mass systému a částice pro simulování chování hejna mají každá některé atributy, které spolu vůbec nesouvisejí. Není tedy vhodné vytvářet univerzální strukturu pro uložení libovolného druhu, protože by výsledná částice zabrala příliš mnoho paměti a volba způsobu zpracování částice by s sebou také přinesla komplikace. Nabízí se možnost implementace částicového systému pomocí nějakého objektového jazyka a pomocí dědičnosti odvozovat jednotlivé druhy částic, ale někteří autoři od použití dědičnosti odrazují, poněvadž s sebou může přinášet zpomalení výpočtu, což může být citelně znát při větším počtu částic. Pravděpodobně spíše záleží na platformě a na překladači, jak efektivním způsobem je v něm dědičnost vyřešena.

2.4.2 Práce s pamětí

V podstatě se nabízejí dvě možnosti, buď jsou částice systému uloženy v poli a každá částice obsahuje proměnnou, která značí, je-li částice aktivní či neaktivní, nebo je vytvořen jakýsi zásobník částic, z něhož jsou částice odebírány a přidávány do systému, po skončení své existence v systému se vracejí zpět do zásobníku. Obě metody mají své výhody i nevýhody, které zde popíšu.

Při použití pole částic není potřeba žádná velká režie pro aktivaci či deaktivaci částice, prostě je částice při přidání do systému označena jako aktivní a při odebírání je označena jako

neaktivní, zpracování částic pak v jednom kroku simulace probíhá tak, že se prochází celé pole a aktualizovány jsou pouze atributy těch částic, které jsou označeny jako aktivní. Nevýhodou je, že při malém počtu aktivních částic se vždy prochází celé pole částic, což může způsobovat zpomalení výpočtu.
Druhý způsob řeší tento problém, navíc dovoluje elegantní využití společné paměti pro více instancí částicového systému, jelikož však uložení částic v zásobníku a následné odebírání vyžaduje použití složitější datové struktury, alespoň spojového seznamu, vzniká režie spojená s odebíráním a přidáváním do zásobníku.
Pro jednodušší a menší implementace částicových systémů se jeví jako výhodnější použití jednoduššího pole částic, pro komplexnější aplikace je zřejmě výhodnější použití zásobníku.

3 Realizační část

Realizační částí této práce je program, který demonstruje některé techniky využití a implementace částicových systémů v praxi. Pro ukázkovou implementaci částicového systému jsem si vybral klasický systém bez vzájemné interakce částic. Částice mají tyto vlastnosti: barvu, váhu, věk, velikost a úhel, dále jsou samozřejmě charakterizovány svou polohou v prostoru, vektorem rychlosti. Samotný systém může obsahovat jeden nebo více emitérů (vrhačů) částic, přičemž každý z nich vytváří nové částice a přidává je do systému. V rámci systému je definován vektor gravitace a odpor prostředí (viskozita). Podle výběru uživatele můžou být částice zobrazeny několika různými způsoby, jako barevné čáry, jako bitmapy a také jako 3D objekty. Dále je v systému zavedena silové pole, realizované jako funkce polohy částice v prostoru a vektoru rychlosti, které může vytvářet další síly působící na částici. Například je takto možné vytvářet víření částic, turbulentní proudění nebo simulovat různá jiná silová pole.

Formulace úlohy

Vytvořit ukázkovou implementaci částicových systémů v prostředí MS Windows, demonstrující některé z popsaných technik a umožňující předvést některé způsoby použití částicových systémů v praxi.

3.1 Analýza úlohy

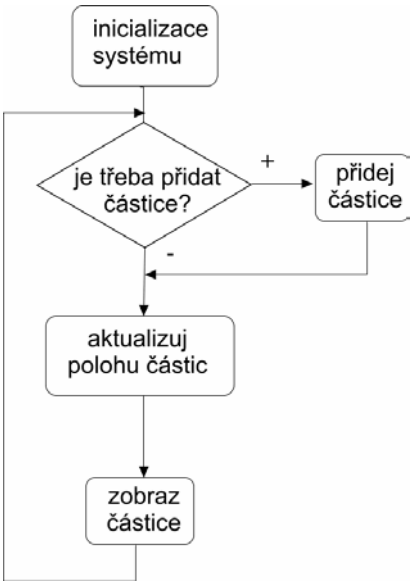
Pro ukázkovou implementaci je možné vybírat z několika možných druhů částicových systémů, popsaných v předchozích kapitolách, které jsou svou podstatou dosti rozdílné a jejichž současná implementace do jednoho systému by byla obtížná a nutila by k vytváření nešikovných konstrukcí a datových struktur. Program by měl nějakým vhodným způsobem umožňovat uživateli důkladně si prohlédnout fungující částicový systém, případně nějakým způsobem jej ovlivňovat tak, aby bylo vidět, jak se mění chování částicového systému v závislosti na nastavovaných parametrech.

V této implementaci jsem vytvořil částicový systém bez vzájemné interakce částic, kde částice jsou ovlivňovány pouze vnějšími silami, důvodem je zejména to že tento typ částicového systému je nejpoužívanější a na výsledném výstupu jsou nejlépe vidět základní principy částicových systémů, konečně i vlastní implementace tohoto typu systému je jednodušší. Uživatel může vybírat z několika ukázkových příkladů předdefinovaných částicových systémů a zobrazovanou scénu si prohlédnout ze všech stran, pohled je ovládán pomocí myši.

3.2 Popis algoritmu řešení

3.2.1 Vývojové schéma funkce částicového systému

Uvedené vývojové schéma na obr. 3.1 ukazuje princip funkce částicového systému, přidávání a odebírání částic a aktualizace jejich atributů, zejména polohy a rychlosti.



Obr. 3.1 Vývojové schéma částicového systému

3.2.2 Výpočet polohy částic

Nové částice jsou do systému přidávány pomocí tzv. *emitérů*. Částicový systém může těchto emitérů obsahovat více. Každý emitér je lokálně transformován vůči nadřazenému částicovému systému a z jeho polohy a natočení je odvozována poloha a směr pohybu nově vytvořených částic. Navíc podle typu je emitéru jsou částice přidávány do systému v jednom bodě nebo ve čtverci, případně v jiném geometrickém tvaru.
V každém kroku simulace je rychlost a poloha částice *i* vypočítávána podle vztahů

$$\mathbf{v}_i' = \mathbf{v}_i + \mathbf{a}_i \Delta t$$

$$\mathbf{p}_i' = \mathbf{p}_i + \mathbf{v}_i' \Delta t$$

kde $\mathbf{a}_i$ je zrychlení částice, vypočítané z celkového působení vnější síly, která je funkcí polohy a rychlosti částice, vydělené hmotností částice

$$\mathbf{a}_i = \mathbf{f}_{ext}(\mathbf{p}_i, \mathbf{v}_i)\frac{1}{m_i}$$

celková vnější síla je výslednicí několika nezávislých složek: gravitační síly $\mathbf{f}_g$, síly odporu prostředí $\mathbf{f}_r$, který je funkcí rychlosti částice a nakonec uživatelsky definovatelné síly prostředí $\mathbf{f}_{env}$, která je funkcí polohy a rychlosti částice.

$$\mathbf{f}_{ext} = \mathbf{f}_g + \mathbf{f}_r + \mathbf{f}_{env}$$

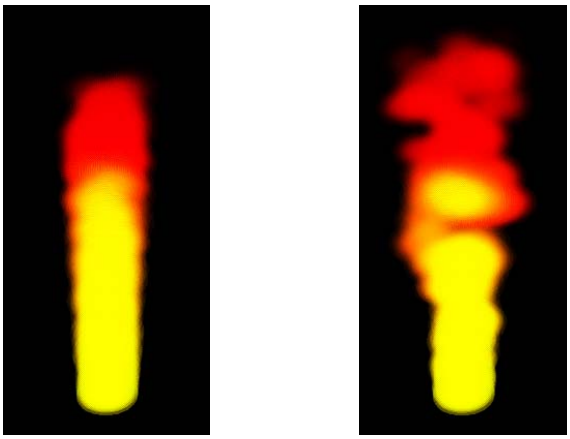
a jednotlivé složky jsou vypočítány podle vztahů:

$$\mathbf{f}_g = m_i \mathbf{g}$$

$$\mathbf{f}_r = -\varepsilon \mathbf{v}_i$$

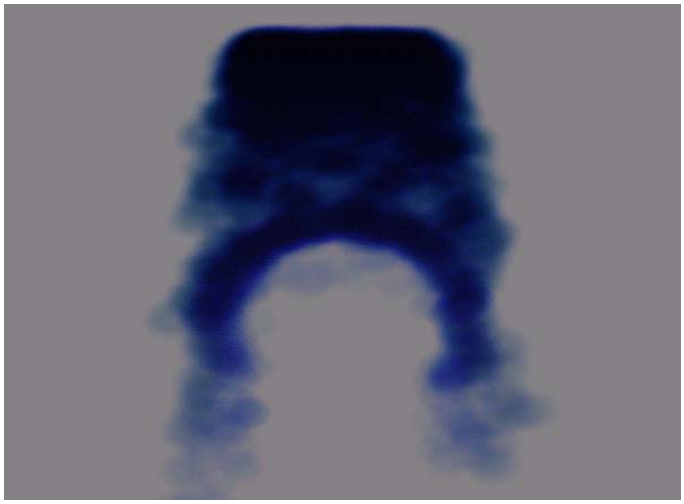
$$\mathbf{f}_{env} = \mathbf{f}_{env}(\mathbf{p}_i, \mathbf{v}_i)$$

kde $\mathbf{g}$ je vektor udávající směr a sílu gravitace, ε je odpor prostředí a funkce $\mathbf{f}_{env}$ je uživatelsky definovaný funkční předpis.



Obr. 3.2: Plamen bez použití šumové funkce(vlevo) a
plamen s šumovou funkcí(vpravo)

Funkčním předpisem může být například definována síla, která náhodně spojitě mění svůj směr, její jednotlivé složky jsou vypočítávány pomocí Perlinovy funkce 3D šumu. Takto vypočítaná síla poté při simulaci ohně způsobí nepravidelné vlnění plamene a shlukování částic do obláčků. Tento efekt do jisté míry připomíná turbulence plynu při hoření (viz obr. 3.2). Dále je například vytvořit funkci, která působí jako vírové pole a nutí částice obíhat okolo nějaké vybrané osy. V demonstračním programu je také využita funkce, která působí odpudivou sílu na povrchu koule, čímž nutí dopadající částice v gravitačním poli obtékat prostor ve tvaru koule (viz obr. 3.3).



Obr. 3.3: Obtékání částic okolo koule

3.3 Popis programu

Aplikace je napsána v programovacím jazyce C++, nicméně, i když se jedná o objektový jazyk, struktura programu není koncipována objektově, program je psán spíše ve stylu jazyka C a pouze využívá některé konstrukce jazyka C++. Zejména jde o přetěžování operátorů, možnost deklarovat proměnné uprostřed těla funkce a podobně. K zobrazení scény používá grafickou knihovnu OpenGL, a tak může využívat případnou hardwarovou akceleraci. Důvodem je zejména to, že program byl koncipován tak, aby se dal později použít také jako knihovna pro práci s částicovými systémy, a to i v jazyce C, v C++ jsou vytvořeny pouze neexportované funkce.
Pro práci s pamětí jsem zvolil spojový seznam se zásobníkem částic, zejména proto, že ačkoli tato technika není příliš obvyklá, je pro použití v částicovém systému celkem výhodná, protože dovoluje „recyklovat“ částice.
Problémem byla také geometrická transformace částic pro výsledné zobrazení na obrazovce. Pokud chceme částice zobrazovat jako plošky pokryté texturou, které jsou stále natočeny kolmo na kameru, existují v podstatě dva možné způsoby, jak dosáhnout natočení plošek na kameru bez ohledu na pohledové a modelové transformace scény. V tomto programu jsem použil vlastní systém transformací a knihovna OpenGL je využita pouze k projektivní transformaci a k rasterizaci scény.

3.3.1 Hlavní smyčka programu

Zde je zhruba sekvence akcí prováděných při běhu programu:

1. Spuštění programu: program načte textury a geometrické objekty z implicitních adresářů, vytvoří okno pro zobrazení scény
2. Zpracování událostí: program zpracuje zprávy od systému, reaguje na stisk klávesy, pohyb myši
3. Výpočet iterace částicového systému: program vypočte jednu iteraci částicového systému, aktualizuje polohy částic podle pohybových rovnic, vytvoří v systému potřebný počet nových částic a odstraní částice které překročily svou dobu života.
4. Transformace scény: program vypočte pohledovou transformaci a modelové transformace částic
5. Zobrazení částic: program transformuje polohu částic systému pomocí pohledové transformace, zvoleným způsobem zobrazí částice .
6. Další snímek: pokud uživatel neprovedl akci která vede k ukončení programu, program pokračuje opět od kroku 2, jinak krok 7
7. Konec programu: program uzavře okno pro zobrazení scény, uvolní alokovanou paměť a ukončí se.

3.3.2 Datové typy pro uložení částicového systému

Základní datovou strukturou částicového systému a potažmo celé aplikace je vlastní datová struktura částice `Particle`

```
struct Particle {
   Vector3 position;
   Vector3 prevPosition;
   Vector3 velocity;
   float mass;
   float age;
   float lifetime;
   float size;
   float growth;
   float angle;
   float rotation;
   Color4 color;
};
```

V této struktuře jsou uloženy všechny informace potřebné k vyobrazení částice a aktualizace její polohy v simulačním kroku. Vektorová hodnota `position` je udává pozici v prostoru, `prevPosition` je předchozí poloha v prostoru, používaná zejména při zobrazování, `velocity` je vektor rychlosti částice. Dále následují skalární hodnoty: `mass` je hmotnost částice, používá se při dosazení do pohybových rovnic, `age` je věk částice, v této simulaci je hodnota ve vteřinách, `lifetime` je věk, ve kterém bude částice odstraněna ze systému, `size` je velikost částice, `growth` je přírůstek velikosti částice za vteřinu, `angle` je úhel natočení částice a `rotation` je rychlost rotace částice a nakonec položka `color` udává barvu částice.
Všechny částice jsou v systému obsaženy ve struktuře `ParticleSys`, což je také struktura, která obsahuje samotný částicový systém.

```
struct ParticleSys {
   List *pool;
   Particle *p_array;
   int particle_total;
   List *emitter_list;
   int emitter_total;
   EnvForceFunc *env_force_fn;

   XForm xform;
   Vector3 gravity;
   float env_viscosity;
};
```

Položka `pool` je spojový seznam, vytvořený nad polem částic, které je alokováno při vytvoření systému, z tohoto seznamu jsou v průběhu odebírány vznikající částice, po zestárnutí a odstranění ze systému se tyto částice sem opět vracejí. `p_array` je pointer na vlastní pole alokovaných částic, slouží zde hlavně k dealokaci paměti při odstraňování systému. `particle_total` je celkový počet částic v systému, `emitter_list` je seznam emitérů v systému, k čemu slouží je vysvětleno dále, `emitter_total` je celkový počet emitérů v systému. `env_force_fn` je pointer na funkci typu `Vector3 EnvForceFunc(Vector3 position, Vector3 velocity)`, která funguje jako silové pole, které působí na částice v závislosti na jejich poloze a na jejich vektoru pohybu. `xform` je matice 4x4, která slouží k modelové transformaci částicového systému. Vektor `gravity` udává směr a sílu gravitace v systému a hodnota `env_viscosity` udává odpor prostředí.
Dále jsou součástí systému, jak už bylo naznačeno v předchozím odstavci, emitéry, jejichž vlastnosti jsou uloženy ve struktuře `ParticleEmitter`.

```
struct ParticleEmitter {
   unsigned int id;
   List *particles;

   int particle_count;
   int particle_total;

   double xform[16];

   float spread_angle;
   float birth_rate;
   ColorMap *cmap;

   float init_velocity;
   float init_lifetime;
   float init_mass;
   float init_size;
   float init_growth;
   float init_angle;
   float init_rotation;
};
```

Položka `id` je jednoznačný identifikátor emitéru v systému, `particles` je seznam aktivních částic, `particle_count` je okamžitý počet aktivních částic a `particle_total` je maximální počet částic, které může emitér vyprodukovat. `xform` je modelová transformace emitéru, `spread_angle` je úhel rozptylu částic, `birth_rate` je počet části generovaných za vteřinu a `cmap` je pole barev, kterými částice postupně prochází během svého života. Všechny hodnoty typu `init_*` inicializují odpovídající hodnoty ve struktuře `Particle` při tvorbě nové částice, s výjimkou `init_velocity`, která násobí počáteční vektor směru pohybu.

3.3.3 Další pomocné datové typy

Struktura Color4 a na ní založené struktura ColorMap je v programu použita ke specifikaci proměn barvy a průhlednosti částice v průběhu jejího života.

```
struct Color4{
   float r,g,b,a;
};
```

```
struct ColorNode {
   Color4 color;
   char is_node;
};
```

```
struct ColorMap {
   ColorNode nodes[256];
};
```

Dále je v programu použita třída Vector3, která slouží k práci s vektory v třírozměrném prostoru.

```
class Vector3 {

public:
   double x,y,z;

   Vector3(){ x = 0.0; y = 0.0; z = 0.0; }
   Vector3(double _x, double _y, double _z){ x = _x; y = _y;
z = _z; }

   Vector3& operator=(Vector3 &r);
   double Dot(Vector3 &v);
   Vector3 Cross(Vector3 &v);
   Vector3 Rotate(double pitch, double yaw);
   Vector3 XForm(double xmat[16]);
   Vector3 Normalize();
private:

};
```

Nabízí některé základní matematické operace s vektory, dále jsou pro ni definovány globálně přetížené operátory sčítání, násobení skalárem apod. ke zjednodušení zápisu výrazů, ve kterých jsou vektory používány.

3.3.4 Moduly, jejich funkce a rozhraní

Hlavní modul MAIN

Obsahuje hlavní funkci main programu, dále funkce pro vytvoření a zobrazení OpenGL okna pro rendering, SetupGLWindow a ShowGLWindow, funkci init pro inicializaci scény a redraw pro její vykreslení na obrazovku. Dále obsahuje obslužné funkce pro ovládání programu myší a klávesnicí.

```
int main(int argc, char **argv)
```
Hlavní funkce programu, vytvoří okno pro vykreslování, inicializuje scénu a přejde do smyčky zpráv.

```
HWND SetupGLWindow( Display *disp, int fullscreen )
```
Funkce pro vytvoření OpenGL okna, vrací handle na vytvořené okno, ve struktuře Display jsou specifikovány rozměry okna, podle parametru fullscreen je vykreslování v okně nebo přes celou obrazovku.

```
void ShowGLWindow( HWND hWnd, Display *disp )
```
Zobrazí OpenGL okno.

```
void init(void)
```
Inicializace scény, natažení textur, nastavení parametrů OpenGL a vytvoření předdefinovaných částicových systémů.

```
void redraw(HWND hwnd)
```
Vykreslení scény. Během toho volá pohledové transformace a následně vykreslovací proceduru zvoleného částicivého systému.

Modul PARTICLE_CLASS

Tento modul je souborem funkcí pro vytvoření a ovládání částicového systému. Obsahuje jednoduché příklady emitérů částic, dále obsahuje funkce pro práci s barevnou mapou.

```
ParticleSys *psysNew( int particle_total )
```
Vytvoří nový částicový systém a alokuje pamět pro počet částic, daný parametrem particle_total.

```
int psysLoadIdentity( ParticleSys *ps )
```
Funkce pro modelovou transformaci, vynuluje transformaci částicového systému nastavení transformační matice na jednotkovou.

```
int psysTranslate( ParticleSys *ps, double x, double y,
double z )
```
Posunová transformace částicového systému. Parametry x, y, z udávají o kolik se poloha změní v jednotlivých osách.

```
int psysRotate( ParticleSys *ps, double theta, double x,
double y, double z )
```
Rotace částicového systému, theta udává úhel natočení, parametry x, y, z určují směrnici osy, podle které bude otáčení probíhat.

```
int psysSetGravity( ParticleSys *ps, Vector3 gravity )
```
Nastavení vektoru gravitace v částicovém systému.

```
unsigned int psysAddEmitter( ParticleSys *ps, EmitterStruct
*es )
```

Přidání emitéru do částicového systému, parametr EmitterStruct je struktura, která obsahuje všechny parametry emitéru (viz. výše struktura ParticleEmitter), kromě modelových transformací. Vrací číslo, které jednoznačně identifikuje emitér v rámci částicového systému.

```
int psysTranslateEmitter( ParticleSys *ps, unsigned int
pe_id, double x, double y, double z )
```

Posun emitéru vzhledem k částicovému systému, parametr pe_id identifikuje emitér v rámci systému, parametry x, y, z určují posun.

```
int psysRotateEmitter( ParticleSys *ps, unsigned int pe_id,
double theta, double x, double y, double z )
```

Rotace rotace emitéru vzhledem k částicového systému, parametr pe_id identifikuje emitér v rámci systému, theta udává úhel natočení, parametry x, y, z určují směrnici osy, podle které bude otáčení probíhat.

```
int psysIterate( ParticleSys *ps, double delta_t )
```

Výpočet jedné iterace částicového systému. Vypočte nové polohy částic v rámci systému, pokud je potřeba, přidá nové částice do systému a částice, které překročily svou dobu života, ze sysému odebere a vrátí zpět do zásobníku částic. Podrobně viz. vývojové schéma Iterace částicového systému. Parametr delta_t určuje velikost časového kroku iterace.

```
int psysRenderTexturedQuads( ParticleSys *ps, int
glTextureId )
```

Funkce pro vykreslení částicového systému. V tomto případě jsou částice vykresleny jako texturované obdélníky, parametr identifikuje texturu OpenGL, kterou bude obdélník potažen.

```
int psysRenderLines( ParticleSys *ps )
```

Funkce pro vykreslení částicového systému. Zde je jsou částice vykreslovány jako čáry mezi aktuální a předchozí polohou částice.

```
unsigned int psysAddPointSpray(ParticleSys *ps,
EmitterStruct *es, float spread_angle )
```

Pomocná funkce pro přidání emitéru do systému, přidává směrový bodový emitér. Parametr spread_angle udává úhel rozptylu částic.

```
unsigned int psysAddRectangleSpray(ParticleSys *ps,
EmitterStruct *es, float spread_angle, float width, float
length )
```

Pomocná funkce pro přidání emitéru do systému, přidává směrový čtvercový emitér. Parametr spread_angle udává úhel rozptylu částic, width je šířka emitéru a length je délka emitéru.

Dále jsou v modulu pomocné funkce pro práci s barevnou mapou.

```
ColorMap* cmap_new( Color4 *start_color, Color4 *end_color )
```

Vytvoří novou barevnou mapu, která je inicializována jako lineární přechod mezi barvou danou parametrem start_color a barvou danou parametrem end_color.

```
int cmap_add_node( ColorMap *cm, int index, Color4 *color )
```

Přidá nový barevný uzel do barevné mapy a přepočte hodnoty barev v barevné mapě s ohledem na nově přidaný uzel.

```
cmap_get_color_t(cm,t)
```

Makro mapující reálný parametr t v rozsahu 0 až 1 na indexy v barevné mapě, vrací barvu odpovídající relativní poloze parametru t v barevné mapě.

Modul VECTOR_CLASS

Tento modul obsahuje zejména třídu Vector3 pro práci s třírozměrnými vektory a pomocné funkce pro přetížené operátory pracující s tímto datovým typem. Dále je skupina funkcí pro pohledové a modelové transformace v třírozměrném prostoru.

```
Vector3& Vector3::operator=(Vector3 &r)
```

Operátor přiřazení, jednoduše zkopíruje hodnoty x,y,z z vektoru r do aktuální instance, vrací referenci na aktuální instanci (k tomu aby bylo možné provádět operace jako: Vector3 c = b = a;).

```
double Vector3::Dot(Vector3 &v)
```

Skalární součin aktuálního vektoru s vektorem v. Vrací výslednou hodnotu.

```
Vector3 Vector3::Cross(Vector3 &v)
```

Vektorový součin aktuálního vektoru s vektorem v. Vrací výsledný vektor, aktuální vektor s nemění.

```
Vector3 Vector3::XForm(double xmat[16])
```

Transformace vektoru transformační maticí danou paramtrem xmat. Vrací tranformovaný vektor, aktuální vektor se transformací nezmění.

```
Vector3 Vector3::Normalize()
```

Vrací normalizovaný aktuální vektor. Samotný aktuální vektor se přitom nemění.

```
Vector3 operator +(Vector3 &l, Vector3 &r)
Vector3 operator +=(Vector3 &l,Vector3 &r)
Vector3 operator -(Vector3 &l, Vector3 &r)
Vector3 operator -=(Vector3 &l,Vector3 &r)
Vector3 operator *(Vector3 &l, double r)
Vector3 operator *(double l,Vector3 &r)
Vector3 operator *=(Vector3 &l,double r)
Vector3 operator /(Vector3 &l, double r)
Vector3 operator /(double l, Vector3 &r)
Vector3 operator /=(Vector3 &l,double r)
```

Přetížené operátory pro sčítání a odečítání vektorů, násobení a dělení vektoru skalárem, v uvedeném pořadí.

```
void xformLoadIdentity( XForm xf)
```

Nastaví transformaci xf na identitu.

```
void xformRotate( XForm xf, double theta, double x, double
y, double z )
```
Rotační tranformace xf, theta je úhel otočení ve stupních, x, y, z udávají směrnici osy otáčení.

```
void xformTranslate( XForm xf, double x, double y, double z)
```
Translační transformace xf. Parametry x, y, z udávají posun v jednotlivých osách.

```
void xformScale(XForm xf, double x, double y, double z)
```
Transformace změny měřítka. Parametry x, y, z udávají koeficienty změny měřítka v jednotlivých osách.

Modul TEXTURES

Modul pro práci s texturami.

```
void LoadTextures(char *path)
```
Načte všechny nalezené soubory s bitmapami, nalézající se v adresáři zadaném parametrem path, vytvoří z nich OpenGL textury, popřípadě změní jejich rozměry tak, aby odpovídaly formátu OpenGL, a naplní jimi globální pole textur textures.

Modul LLIST

Modul pro práci obousměrným spojovým seznamem. Obsahuje větší množtví funkcí, uvedu jen ty, které jsou využity v této aplikaci.

```
List *list_prepend(List *list,void *data)
```
Přidá nový prvek data na začátek spojového seznamu list. Vrací ukazatel na aktualizovaný seznam.

```
List *list_free(List *list)
```
Uvolní spojový seznam z paměti. Neuvolňuje paměť pro data uložená v seznamu.

Modul PERLIN

V tomto modulu je implementace Perlinova šumu.

```
double noise1(double arg);
float noise2(float vec[2]);
float noise3(float vec[3]);
```

Jedno-, dvou- a třírozměrná verze Perlinovy nízkofrekvenční šumové funkce. Jsou využívány například při definování uživatelského předpisu síly pro simulaci turbulence plamene.

4 Modelování v 3DStudiu MAX

Částicové systémy jsou nějakým způsobem zakomponovány do velkého množství modelovacích a animačních programů. V jednodušší podobě jsou částicové systémy používány také v jednom z nejrozšířenějších programů pro 3D animaci, ve 3DStudiu MAX od firmy Kinetix [13]. V základní verzi je pouze několik druhů základních částicových systémů, nicméně i ty stačí k vytvoření zajímavých animací. Komplexnější implementace částicových systémů nainstalovatelné do 3Dstudia MAX lze pořídit v podobě zásuvných modulů (pluginů). V této kapitole krátce popíšu možnosti animace pomocí částicových systémů v 3DStudiu MAX verze 3, jejich jednotlivé typy a nakonec způsob jakým byly vytvářeny scény, které jsou součástí této práce.

4.1 Možnosti a typy částicových systémů v 3DS MAX

Základním prvkem částicového systému v 3DS MAX je emitér částic. Slouží jako objekt, na který je vázáno vytváření nových částic do systému a nastavení jeho parametrů udává jaké částice budou v sytému a jak se budou chovat. Emitérů je celkem 6 druhů.

Spray
Nejjednodušší emitér, má jednoduché nastavení, slouží pouze k vrhání částic jedním směrem. Částice mohou být pouze jednoduchá geometrická primitiva jako body a čáry.

Snow
Podobný emitéru *Spray*, dovoluje nastavovat některé speciální parametry, vhodné pro simulaci sněhu nebo deště.

SuperSpray
Výrazně komplexnější varianta emitéru *Spray*, umožňuje například na místě částic zobrazovat uživatelem definované geometrické objekty nebo takzvané metaparticles (částice chovající se jako kapalina). Dále umožňuje široké nastavení parametrů pohybu a otáčení částic a mnoha dalších parametrů.

Blizzard
Komplexní varianta emitéru *Snow*, velmi podobné možnosti jako *SuperSpray*.

PArray
Emitér určený k rozmísťování částic po povrchu geometrického objektu. Lze jej použít například k modelování zatravněné plochy.

PCloud
Podobný jako *PArray*, rozmísťování částic však neprobíhá na povrchu objektu, nýbrž v celém objemu. Používá se například, jak název napovídá, k simulaci mraků či kouře.

Pohyb částice po vypuštění z emitéru lze v 3DStudiu dále ovlivňovat pomocí takzvaných space-warpů, což jsou jakási prostová pole, která deformují všechny objekty v prostoru ve kterém působí. Pro částice je významný například *Gravity*, který v prostoru vytvoří orientovanou gravitační sílu, ta působí na částice a ovlivňuje jejich pohyb. Space-warpů je celá řada, některé z nich na částice nepůsobí a naopak některé působí pouze na částice. Další významný space-warp je Deflektor a jeho odvozeniny, který působí jako „odražeč“ částic a používá se tam, kde je potřeba aby se částice odráželi od nějakého povrchu.

Částicím lze také přiřadit různé materiály, které určují, jak budou vyzobrazeny při finálním výpočtu scény. Nastavení materiálů, barev a textur částic (a obecně objektů) je v 3DStudiu poměrně složité a dovoluje nastavit obrovské množství detailů co se týče výsledného vzhledu částice a jsou k dispozici některá speciální nastavení, která se týkají výhradně částic. Například jedním z nich je materiál ParticleAge, který dovoluje zvolit změnu vzhledu (textury či barvy) částice v závislosti na jejím věku.

4.2 Vytvořené scény

Oheň

Tento jednoduchý model ohně ukazuje zejména jak je v 3DStudiu možné vytvořit oheň, ve kterém se plameny proměňují v kouř. Vlastní realizace obsahuje vlastně dva částicové systémy typu *SuperSpray*, jeden slouží pro simulaci plamenů a druhý vytváří kouř. Důvod tohoto rozdělení je ten, že plameny jsou do scény vykreslovány tak, že hodnota jejich barevné intenzity se přičítá k pozadí, což má za následek ten efekt, že plameny se navzájem zesvětlují. Naopak intenzita kouře se od pozadí odečítá, což má za následek ztmavení. Oba druhy barevné kompozice nelze použít v jednom materiálu, proto je model ohně rozdělen na dva částicové systémy.

Vodopád

Velmi jednoduchá simulace tekoucí vody. Ukazuje hlavně odrážení částic od objektu (voda od skály) pomocí space-warpu typu *Deflektor*. Využívá emitér typu *Spray*.

Sníh

Demostruje použití uživatelem definovaných objektů na místě částic. Jednotlivé vločky jsou zobrazovány jako instance předem vytvořeného objektu ve tvaru sněhové vločky. Velikost částic je výrazně zvětšena tak, aby byly vidět jednotlivé objekty. Částice jsou generovány emitérem typu *Blizzard*.

5 Závěr

V této práci jsem se pokusil popsat některé nejpoužívanější druhy částicových systémů a jejich použití pro modelování přírodních jevů a obecně animovaných geometrických objektů. Částicové systémy nejsou nijak zvlášť pevně svázány nějakou podloženou teorií, ani jejich implementace nevyžaduje využívat specifické, přesně dané algoritmy a jejich použití je velmi široké. Z těchto důvodů jsem částicové systémy v textu neklasifikoval podle použití, ani podle způsobu implementace. Použitá klasifikace je pouze jedna z možných a pouze zdůrazňuje ty charakteristiky částicových systémů, které jsem chtěl vyzdvihnout.
Jelikož tato práce se zabývá zejména dynamickým chováním částic a spíše okrajově se zabývá rozmanitostí vzhledu a způsobu zobrazení částic, při tvorbě demonstračního programu se to odráží větším výběru možností chování částic oproti pouze základním způsobům zobrazení částice (čára nebo bitmapa). Vytvořená implementace podle mého názoru postačuje k základní demonstraci modelování pomocí částicových systémů a může sloužit jako výchozí kostra pro vytvoření komplexnější implementace, případně knihovny pro vytváření částicových systémů.
Namodelované příklady částicových systémů v 3DStudiu MAX slouží pouze k prozkoumání a demonstraci možností, jaké tento program nabízí pro modelování pomocí částicových systémů. Vytvoření realističtějších scén, použitelných v praxi, by si vyžádalo daleko více času a zejména zkušeností, neboť výsledný efekt se tomto případě dosti těžko odhaduje.

Literatura

[1] Reeves, W. T.; *Particle Systems – A Technique for Modeling a Class of Fuzzy Objects*. ACM Transactions on Graphics, Vol. 2 Apr. 1983.

[2] Paramount; *Star Trek II: The Wrath of Khan*. (film), 1982.

[3] Blinn, J.*; Light Reflection Functions for the Simulation of Clouds and Dusty Surfaces*, SIGGRAPH 82, pp 21-29.

[4] id Software; *www.idsoftware.com*

[5] Reeves, W. T., Blau R.; *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems*. SIGGRAPH 85. pp 313-322.

[6] Sims, K.*; Particle Animation and Rendering Using Data Parallel Computation,* Computer Graphics, Volume 24, Number 4, 1990.

[7] Angel E.*; Interactive Computer Graphics with OpenGL*, Adison-Wesley, 2000, pp 467-481.

[8] Beneš B., Felkel P., Sochor J., Žára J.; *Vizualizace*. Skripta FEL ČVUT Praha, 1997, str.21-22.

[9] Přikryl P.; *Numerické metody – Aproximace funkcí a matematická analýza*. skripta FAV ZČU Plzeň, 1996.

[10] Reynolds C.W.; *Flocks, Herds, and Schools: A Distributed Behavioral Model*. Computer Graphics, Volume 21, Number 4, 1987, pp 24-31.

[11] Prusinkiewicz P., Lindenmayer A.; *The algorithmic beauty of plants*. Springer-Verlag, 1990.

[12] Smith A. R.; *Plants, fractals, and formal languages*, SIGGRAPH 84, Computer Graphics 18, 1984, pp 1-10.

[13] Kinetix Ltd.; *www.ktx.com*

Přílohy

Popis ovládání programu

Jelikož se jedná o demonstrační aplikaci, je ovládání programu velmi jednoduché, uživatel vlastně pouze přepíná jednotlivé předdefinované ukázky částicových systémů a myší otáčí kameru okolo vykreslovaného částicového systému.

Přepínání zobrazovaného částicového systému

řadou kláves 1 až 9 a 0.

Změna úhlu pohledu na částicový systém

pohybem myši při stisknutém levém tlačítku, horizontální pohyb mění azimut pohledu, vertikální mění zenit pohledu

Změna pozice pozorovatele

vertikálním pohybem myši při stisknutém pravém tlačítku, mění vzdálenost pozorovatele od částicového systému