VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

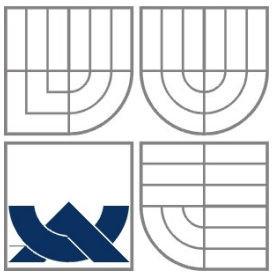
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REALISTICKÉ ZOBRAZENÍ MRAKŮ A KOUŘE

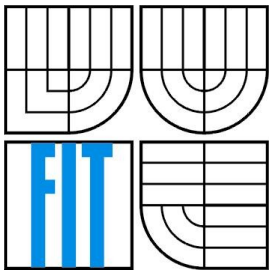
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Jan Kopidol
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REALISTICKÉ ZOBRAZENÍ MRAKŮ A KOUŘE

REALISTIC RENDERING OF SMOKE AND CLOUDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Kopidol
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radovan Jošth
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce se zabývá metodami pro vykreslování objemových dat jako jsou mraky nebo kouř v počítačové grafice. V úvodu jsou shrnuty techniky a triky používané v počítačové grafice pro zobrazení těchto těles ve scéně, především ty, které využívají objemovou reprezentaci, jejich výhody a nevýhody. Následující část je hlouběji věnována vybrané technice pro zobrazování objemových dat – vykreslování mraků a kouře s ohledem na chování světla uvnitř jejich objemu (nazývané participating media), jejímu principu a vztahům ze kterých vychází. Je uveden krátký přehled aplikací, sloužících pro realistické zobrazování scény, které by byly vhodné pro implementaci vybraného algoritmu. Blender, do kterého je algoritmus implementován je popsána podrobněji, včetně jeho vnitřní struktury, obzvláště renderovacího engine. Poslední část práce je věnována návrhu algoritmu, jeho integrace do aplikace a postupu při jeho implementaci.

Klíčová slova

počítačová grafika, objemová data, mraky, kouř, participating media, realistické zobrazování, Blender

Citace

Kopidol Jan: Realistické zobrazení mraků a kouře v češtině. Brno, rok, diplomová práce, FIT VUT v Brně.

Abstract

This work discourses about methods of rendering volumetric data such as clouds or smoke in computer graphics and implementation of this feature to existing application. The first part is summary of techniques and tricks used in computer graphics to display such objects in scene, their pros and cons and the most used techniques of displaying volumetric data. Next part is more closely focused to choosed technique of rendering volumetric data with consideration of light behavior inside the volume (also called participating media) and basic relationships used used in computation. In following part of work there is short list of applications – renderers used to realistic rendering of scene, which are suitable for implementation of selected volumetric data rendering algorithm. Selected application – Blender is described more deeply including its inner structure, especially rendering engine. Last part of work is dedicated to design, implementation and integration of rendering algorithm itself.

Keywords

computer graphics, volumetric data, clouds, smoke, participating media, photorealistic rendering, Blender

Realistické zobrazení mraků a kouře

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radovana Joštha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Mé poděkování patří především Ing. Radovanu Joštovi za jeho podporu a inspiraci při tvorbě této práce.

Obsah

1 Úvod

Informační technologie a počítače nalézají v dnešní době uplatnění v nejrůznějších odvětvích lidské činnosti. Výjimkou není ani počítačová grafika, která se začala rozvíjet před více než třiceti lety. Díky tehdejšímu slabému výkonu počítačů a nedostatku paměti se zpočátku používala jenom na schematické zobrazování - grafický výstup tvořily pouze rovné čáry. Rostoucí výpočetní výkon však umožnil náš trojrozměrný svět simulovat stále přesněji a podrobněji a tak se dnes setkáváme s počítačem generovanými obrázky které působí stejně realisticky jako fotografie. Počítačová grafika má v dnešní době uplatnění v několika průmyslových odvětvích a bývá spojována se systémy pro projektování (CAD), zobrazování statistických údajů, úpravu obrazu a mnoha dalšími. Pro účely této práce je však nejzajímavější její použití při realistickém zobrazování scén.

Pro vytváření nových obrazů na počítači lze zvolit několik přístupů. Nejpřirozenější cestou je kreslení nebo malování s použitím virtuálních nástrojů které je svou podstatou velmi podobné klasické umělecké tvorbě například malbě na plátno nebo kresbě. Pokud má být výsledek realistický, je od autora vyžadován cit pro perspektivu a schopnost správně zachytit nasvětlení objektů, jejich stíny a odrazy.

Počítače však nabízí také další možnost – vymodelování scény ve virtuálním prostoru a její následné zobrazení. Při tomto přístupu autor díla vytvoří trojrozměrné modely objektů které mají být zobrazeny a ty umístí do scény. Těmto objektům pak nadefinuje materiál který má být použit pro jejich zobrazení. Tento způsob práce je výhodný díky své přesnosti, umožňuje použít již vytvořené objekty (např. z CAD) a především dovoluje snadné vytváření animací. Také umožňuje snadnou úpravu výsledku například pouhým přesunutím objektu a znovuvyužití již vymodelovaných prvků v jiných scénách.

Zobrazování 3D objektů na počítači lze rozdělit podle způsobu vykreslování na akcelerovanou a neakcelerovanou počítačovou grafiku. Akcelerovaná 3D grafika umožňuje prakticky okamžité zobrazení scény a tedy malou odezvu – využívá se zejména v interaktivních aplikacích (3D editory, 3D prohlížeče, počítačové hry). Tento pojem se používá proto, že při zobrazování je použit grafický akcelerátor a specializované výpočetní jednotky GPU. Neakcelerovaná grafika naproti tomu používá pro výpočet scény většinou procesor. Díky tomu, že je CPU univerzální, méně omezený a umožňuje také snadnější přístup k operační paměti, bývají výsledky vykreslované procesorem realističtější, ovšem za cenu delšího času výpočtu. Software který zajistí výpočet zobrazení se nazývá renderer nebo také renderovací engine. Kvalita výsledku pak závisí především na rendereru který je použit pro vykreslení scény. Pro dosažení realistických výstupů slouží techniky jako raytraycing, radiosita, ambient occlusion nebo global illumination které jsou schopny simulovat reálné chování světla, většinou jsou ale určeny pouze pro objekty s pevným povrchem. S vývojem grafických akcelerátorů se poslední dobou začíná pro výpočet neakcelerovaných scén používat také výpočetní výkon

grafických karet. Pomocí speciálních programových knihoven lze do výpočtu zapojit také GPU (především pro vektorové operace), ve výsledku je tak pro vykreslení scény použit jak CPU tak GPU.

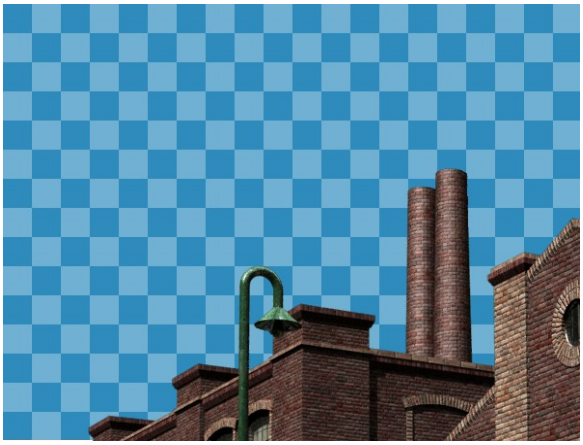
V mnoha případech je však ve scéně potřeba zobrazit mraky, oheň nebo kouř, což jsou tělesa která nemají povrch (říká se jim objemová tělesa) a nelze na ně aplikovat běžně používané metody. Světlo se totiž neodráží od povrchu, ale proniká dovnitř, kde dochází lomům a odrazům od mikroskopických částic ze kterých se skládá. Je třeba podotknout, že zobrazování a simulace chování světla v těchto médiích nachází uplatnění také jinde, než při fotorealistickém zobrazování uměleckých děl nebo ve filmech. Dalšími oblastmi nasazení je bezpečnost - např. V dopravě (především letecké) nebo návrh osvětlovacích prvků (světelných reflektorů). Problematikou vykreslování těchto objemových těles se zabývá tato práce, důraz je však kladen především na vizuální kvalitu a fyzikální přesnost není důležitá.

Jelikož je zobrazováním trojrozměrné scény poměrně složitý proces, je výsledná implementace zakomponována do již existujícího software sloužícího pro zobrazování trojrozměrných dat. To významným způsobem usnadnilo práci při implementaci a umožnilo tak využít již implementované algoritmy např. pro transformaci objektů nebo texturování. Navíc má tak má výsledek větší potenciál pro své využití, protože není omezen na zobrazování objemových dat.

2 Současné metody zobrazování mraků a kouře v realistické 3D grafice

2.1 Doplnění v postprocesingu

Postprocesing v počítačové grafice je závěrečná fáze tvorby scény nebo animace. Vychází se z již vyrenderované (vykreslené) scény které se říká render. Zpravidla se jedná o barevné úpravy, přidání detailů které by bylo příliš náročné vymodelovat nebo o doplnění efektů, které daný renderer nepodporuje nebo by byly příliš náročné na vytvoření či vykreslení. Takovými efekty může být například hloubka ostrosti nebo právě domalování mraků či kouře. Tyto úpravy se většinou provádí v 2D editorech (Photoshop, Gimp) nebo specializovaných programech – kompozitorech (Combustion), které umožňují pracovat např. se z-bufferem a dalšími daty scény která nejsou přímo vidět. U náročnějších scén, se výsledný render rozděluje do několika vrstev a kompozitor slouží ke „sloučení“ (poskládání) těchto vrstev, současně umožňuje aplikovat na ně různé efekty. Použití kompozitorů navíc umožňuje snadnou práci s animacemi – definuje proces úprav univerzálně pro každý snímek animace. Někdy bývají kopozitory přímo součástí grafiké aplikace (např. Blender Node Editor).



Obrázek 1: Scéna vykreslená s průhledným pozadím

Obrázek 2: Scéna doplněná o pozadí v 2D editoru

Realistické vykreslování mraků a kouře bylo až do nedávna velmi komplikované, proto se tento problém různě obcházel. V případě mraků je nejjednodušším řešením dát na pozadí scény vhodnou fotografii oblohy nebo scénu vyrenderovat s průhledným pozadím a oblohu i s mraky doplnit v 2D editoru (Obrázek 1 a 2). Je však třeba brát v potaz nasvětlení oblohy i scény, aby výsledek vypadal věrohodně. Podobným způsobem lze do scény doplnit oblaka kouře. Je velmi komplikované izolovat kouř z fotografie od pozadí tak aby se dal použít na jinde. Proto bývá většinou

domalován ručně pomocí speciálních štětců (Obrázek 3). Stíny kouře by šly domalovat podobně jako kouř samotný, ale pokud by se ve stínu nacházely složitější objekty, bylo by to velmi náročné, proto je někdy vhodnější vytvořit pro stín speciální texturu a vykreslení stínu přenechat rendereru. Blíže je tento způsob tvorby stínu popsán v následující podkapitole.

Obrázek 3: Domalován kouř v 2D editoru - konečný výsledek

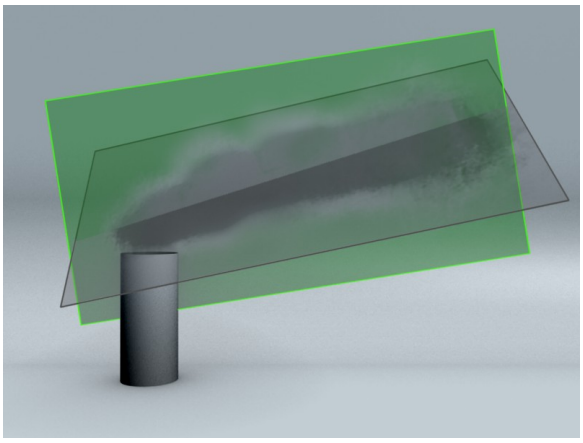
Vykreslování objemových dat je v mnoha dnešních redererech poměrně dobře podporováno, přesto má malování mraků a kouře v postprocesingu stále své místo. Hlavním důvodem je především dobrá kontrola nad výsledkem, efektivita a rychlost (v některých případech).

2.2 Použití spritů

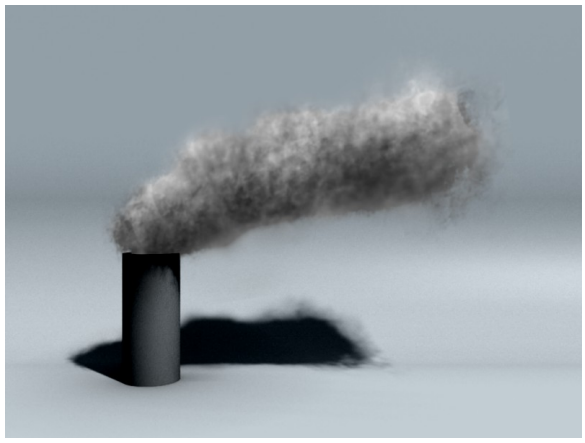
Někdy je potřeba kouř zakomponovat do scény již v průběhu rendrování. Například pokud je před kouřem nějaký předmět scény, který jej překrývá. V tomto případě by byl postprocesing velmi komplikovaný. Z toho důvodu lze ve scéně pro kouř vytvořit speciální objekt, na který je namapována textura kouře (Obrázky 4 a 5). Tím pádem je kouř přímo vyrendrován a není třeba řešit překrývání objektů. Na druhou stranu autor nemá nad kouřem tak dobrou kontrolu jako v případě postprocessingu a pokud jej potřebuje nějak upravit, vyžaduje to opětovné vyrendrování celé scény (což může být v některých případech velmi časově náročné). Stín lze řešit opět pomocí nějaké speciální stínové textury. Princip této metody vystihuje obrázek 4 – zeleně je zobrazen sprite pro kouř samotný, šedě pak sprite pro stín, který je nakloněn přímo naproti světlu (aby nedocházelo ke zkreslení stínu). Toto řešení je použitelné pouze v případě, že ve scéně vrhá stíny pouze jedno svetlo.

Dalším úskalím tohoto přístupu je zobrazování předmětů které se nachází v oblasti do které kouř zasahuje. Jelikož kouř i stín jsou reprezentovány pouze jednou plochou, musejí předměty zákonitě být na jedné nebo druhé straně nebo budou touto plochou rozděleny. Takže ve výsledku se budou jevit buď tak, že v kouři vůbec nejsou, že jsou za ním nebo je objekt „rozpůlí“ (obdobný

problém nastává v také v případě objektu pro stín). Oba případy je třeba řešit v postprocesingu doretušováním kritických oblastí.



Obrázek 4: Znázornění tvorby kouře a jeho stínu pomocí spritů



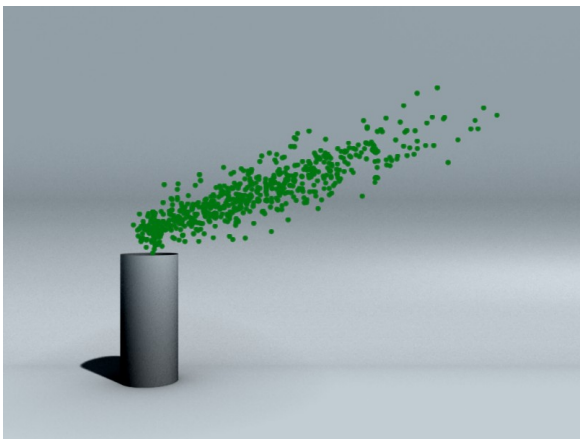
Obrázek 5: Kouř vytvořený pomocí spritů - výsledek

2.3 Použití částicových systémů

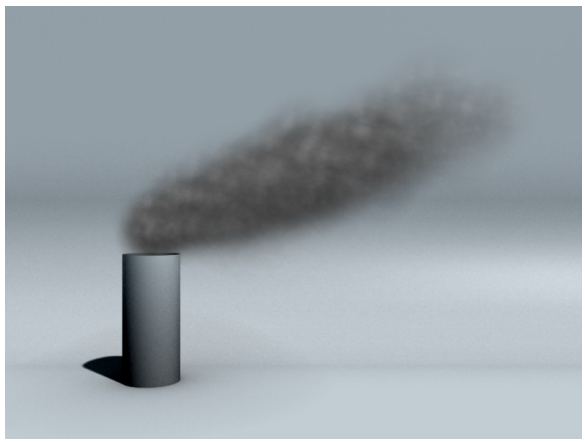
Výše uvedené postupy jsou vhodné především pro statické scény. Problém ovšem nastává, je-li vytvářena animace. Pokud se nemění úhel pohledu na nějaký zobrazovaný oblak kouře, lze použít trik s texturou (v tomto případě animovanou – např. videosekvencí). Je-li v animaci na nějaký oblak kouře nebo páry pohled z různých stran, je třeba zvolit zcela jiný přístup.

Nejjednodušší možností pro tyto případy je zobrazení kouře pomocí částicového systému (Obrázky 6 a 7). Tato metoda je svou podstatou již algoritmická a od autora vyžaduje jenom vhodné nastavení parametrů – mraky nebo kouř jsou kompletně generovány počítačem. Celý oblak je simulován v prostoru a tak lze zobrazovaný oblak pozorovat ze všech stran aniž by se jevil ploše. Navíc problém s objekty uvnitř kouře není tak viditelný jako v případě spritů. Při vykreslování je ale stále částicový systém převeden na skupinu plošek s texturou a při použití větších částic dochází ke stejnému problému jako u spritů. Textura bývá většinou generovaná – jedná se často o nějaký druh šumu.

Hlavní nevýhodou je nevěrohodnost zobrazení - to je způsobeno zejména špatnou kontrolou nad osvětlováním. Částicové systémy bývají často speciálními objekty a výpočet světelných podmínek v rámci částicových systémů by byl při nejmenším velmi komplikovaný. Dalším významným problémem jsou stíny, které v podstatě úzce souvisí také s problémem osvětlení. Na rozdíl od běžných objektů, kde k výpočtu osvětlení modelu stačí většinou normála povrchu a k výpočtu stínů jednoduchý raytraycing nebo shadow buffer, je třeba v obou případech zjistit útlum světla všemi ostatními částicemi (navíc pro každou částici zvlášť) .



Obrázek 6: Znázornění tvorby kouře pomocí částicového systému



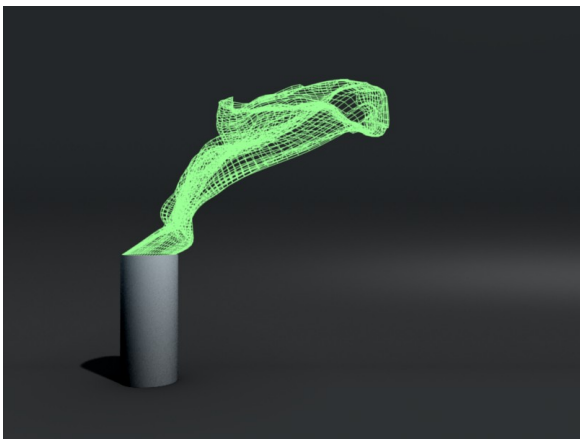
Obrázek 7: Kouř vytvořený pomocí částicového systému - výsledek

Stíny by u částicových systémů šly řešit texturou, podobně jako u metody se sprajty, ale pak nepůjde vytvořit věrohodnou animaci. Další možností by bylo vytvořit nový částicový systém se speciálními částicemi které budou pouze vrhat stín, to může být ovšem v některých aplikacích problematické. V Blenderu by tento problém šel řešit tak, že by byly částicovým systémem generovány objekty schopné vrhat stín – např. plošky s texturou. Tento postup je však poměrně komplikovaný a málo pružný – je třeba kontrolovat souvislosti mezi původním a stínovým částicovým systémem. Nejlépe se hodí částicové systémy pro simulaci ohně, malých oblaků páry a podobných jevů, u kterých není třeba starat se o nasvětlení ani stíny.

2.4 Vytvoření objektu se speciálním materiálem

Ve speciálních případech lze u některých druhů kouře pozorovat jasně viditelný “povrch” - například u cigaretového kouře nebo pomalu stoupající páry. V tomto případě je možné celý objekt kouře vymodelovat jako jakýkoliv jiný 3D objekt – např pomocí polygonů nebo NURBS. Místo modelování může autor použít například simulaci oděvu aplikovanou na kužel - tato metoda byla použita v ilustračním obrázku (Obrázky 8 a 9).

Po vytvoření modelu mu stačí přiřadit vhodný materiál a vyrendrovat. U použitého materiálu je důležité zejména to, jak svou průhledností dokáže napodobit struktury které v kouři vznikají. Pro ukázkové obrázky byl materiál vytvořen v Materiálovém Node Editoru Blenderu (nástroj Blenderu pro manipulaci s materiály) a jeho průhlednost je určena normálou povrchu - čím více je materiál nakloněn na kameru, tím více je průhledný. V tomto případě lze za algoritmus použitý pro vykreslování považovat shader materiálu aplikovaného na objekt. Pokud je vyrenderovaný obrázek ještě upraven v postprocesingu, může být výsledek velmi realistický.



Obrázek 8: Znázornění tvorby kouře pomocí modelu a speciálního materiálu

Obrázek 9: Kouř vytvořený pomocí modelu a speciálního materiálu - výsledek

Nevýhodou tohoto přístupu je problematická animace takto vytvořeného kouře, jelikož se stále jedná o mesh. Navíc tento druh kouře se často chová odlišným způsobem než hustší kouř – lze pozorovat různé deformace a rozpad na počátku snadno identifikovatelných struktur. To značně komplikuje možnost animace a proto je tato metoda vhodná především pro statické scény.

3 Vykreslování materiálů ovlivňujících průchod světla

3.1 Modelování média v 3D editoru

Pro vykreslování média je důležité, jak bude reprezentováno. Před samotným vykreslením je třeba nástroji pro zobrazování scény popsat, kde a jak se má médium zobrazit. Při tomto úkolu je kladen důraz na snadnost definování média, která umožní efektivní tvorbu scény grafikovi, který ji bude vytvářet. Zároveň však musí být k dispozici nástroje, které dokáží dané médium co nejpřesněji nadefinovat a vymodelovat.

Jelikož hustota média nemusí být ve všech místech jeho objemu stejná, je potřeba ji reprezentovat nějakou funkcí, která ji bude jednoznačně definovat pro všechny body prostoru. Tuto funkci je třeba volit s ohledem na možnosti modelovacího nástroje, který bude sloužit pro definování média – jeho rozměrů, polohy a vlastností. Za tímto účelem se hodí použití kombinace procedurálních 3D textur (které jsou schopny simulovat složitou strukturu některých typů kouře, mraků nebo páry) a nějakých vhodných modelovacích nástrojů, které jsou schopny snadno definovat objem a polohu daného objektu. Nejpodstatnější informací, která je nutná pro vykreslení média je zřejmě jeho hustota (množství částic v jednotlivých oblastech objemu). Dále je to barva a další vlastnosti, které mají vliv na chování světla uvnitř objemu. Pro tyto účely lze použít například další textury které budou definovat tyto vlastnosti. Často je pro zobrazení rozhodující hlavně intenzita, případně barva, ostatní vlastnosti lze většinou určit z intenzity.

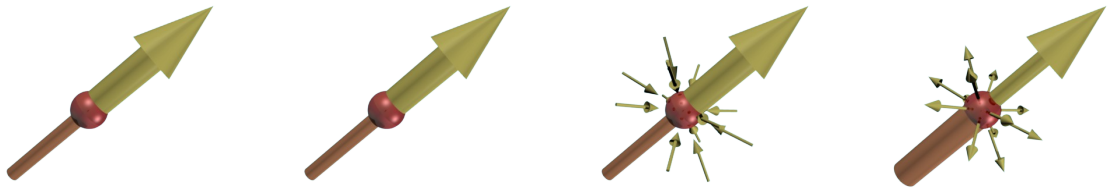
Pro účely modelování lze s výhodou použít klasické modelovací nástroje sloužící pro modelování objektů s pevným povrchem, pomocí nichž lze určit objem, ve kterém se médium nachází. V některých pokročilých aplikací je pro tento typ prvku scény definován speciální druh grafického objektu, který se umisťuje do scény podobně jako ostatní objekty. Výhodou tohoto speciálního objektu je možnost nastavení parametrů kouře pro konkrétní objekt – například plynulosti přechodu. Navíc je také zajištěna konzistence – u klasického meshe může být například jeho objem uvnitř rozdělen stěnou nebo naopak nemusí být vůbec uzavřen (a tak nelze určit co je uvnitř a co je vně). Podrobněji je problematika reprezentace objemových dat popsána v podkapitole 3.3.

3.2 Chování světla které prochází skrze participating media

Při renderování kouře nebo mraků se setkáváme s několika problémy. Na rozdíl od obyčejných objektů nestačí spočítat pouze osvětlení hranic média (oblaku kouře nebo páry). Při průchodu světla materiálem dochází k interakci s materiálem v celém objemu média, a proto je třeba znát vlastnosti materiálu nejen na hranicích ale ve všech bodech objemu. Dochází k několika optickým jevům, ale pro účely této práce bude důležité zejména osvětlení jednotlivých bodů. Jevy jako rozklad světla nebo změna spektra nejsou při pozorování kouře nebo mraků příliš časté a při implementaci na ně nebude brán zřetel.

Paprsek, který prochází médiem lze rozdělit do několika složek (Obrázek 10) [1]:

- světlo přímo ze světelného zdroje a světlo vyzářené materiálem (emise)
- světlo, jež je materiálem pohlceno (absorbce)
- světlo, jež je materiálem odraženo z okolí zpět do směru paprsku (in-scattering)
- světlo, jež je materiálem odraženo do okolí (out-scattering)



emise absorbce in-scattering out-scattering

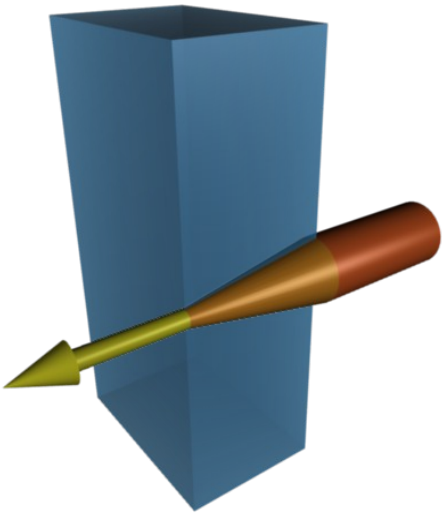
Obrázek 10: Složky paprsku procházející médiem

Ve skutečnosti k absorpci a odrazu světelného paprsku dochází při jeho průchodu nějakou mikroskopickou částečkou kouře nebo páry. Simulovat přímo interakci s těmito částicemi by bylo výpočetně velmi náročné. Při modelování média proto nejsou modelovány jednotlivé částice ale pouze jejich hustota. Podle hustoty částic lze dále určit pravděpodobnost s jakou bude paprsek odražen nebo pohlcen a to postačí pro simulaci chování světla v médiu.

Optickou hustotu lze určit jako množství částic na jednotku objemu. Tato je pro účely výpočtu označena jako koeficient absorpce κ_a – pro nějaký úsek (diferenční vzdálenost) je to $\kappa_a(x)dx$. Obdobně lze uvažovat pro rozptyl paprsku do okolí (v podstatě je také jeho energie "ztracena", není však pohlcena ale rozptýlena do okolí). Pro tuto veličinu je zaveden koeficient rozptylu (scattering) κ_s. V obou případech dochází k úbytku světla – přesněji řečeno radiance L. Proto je zaveden společný pojem – koeficient úbytku $\kappa_t = \kappa_a + \kappa_s$. Vztah pro úbytek radiance pro nějaký úsek je následující (1):

$$dL(x) = -\kappa_t(x)L(x)dx \quad (1)$$

Situaci ilustruje obrázek 11:



Obrázek 11: Úbytek světla při jeho průchodu materiálem

Výsledkem této diferenciální rovnice je Beer-Lambertův zákon [1], který popisuje absorpci elektromagnetického záření (v tomto případě světla) látkou.

$$L(x) = L(x_0)\,\mathrm{e}^{-\int_{x_0}^{x} \kappa_t(u)\,du} = L(x_0)\,\tau(x_0, x) \quad (2)$$

$$\tau(x_0, x) = \mathrm{e}^{-\int_{x_0}^{x} \kappa_t(u)\,du} \quad (3)$$

$$\int_{x_0}^{x} \kappa_t(u)\,du \quad (4)$$

(4) označuje optickou hustotu (tedy hustotu částic které odrážejí a pohlcují světlo), $\tau(x_0,x)$ označuje propustnost materiálu (transmittance) v úseku mezi body x_0 a x.

Další část tvoří složky, které intenzitu světla naopak zvyšují. Mezi ty patří vlastní vyzařování materiálu (emise) a paprsky, které se odrazí do směru paprsku (in-scattering). Zatímco u rozptylu paprsku do okolí bylo jedno, do jakého směru se paprsek odrazí, v opačném případě (in-scattering) je potřeba brát v potaz pravděpodobnost, s jakou se paprsek odrazí právě do směru zkoumaného paprsku – nebo jinými slovy množství světla, které se do daného směru odrazí. Pro popis této složky slouží fázová funkce $p(\omega_0, \omega_i)$ – ω_0 označuje směr zkoumaného paprsku, ω_i směr vnějšího paprsku, který je odrážen. Název fázové funkce vznikl z podobnosti k fázím měsíce. Funkce ve skutečnosti popisuje podíl mezi množstvím světla, které je skutečně odraženo a mezi množstvím, které by bylo odraženo, kdyby funkce byla nezávislá na směru paprsku (a příchozí paprsek rovnoměrně rozptýlen). Parametrem této funkce může být také pouze úhel mezi směrem paprsku vstupujícího a paprsku zkoumaného (tedy úhel mezi ω_0 a ω_i), jelikož u většiny fázových funkcí nehrají skutečné směry roli (funkce je symetrická podle směru zkoumaného paprsku) a důležitý je právě úhel, který paprsky svírají (označuje se fázový úhel) [3].

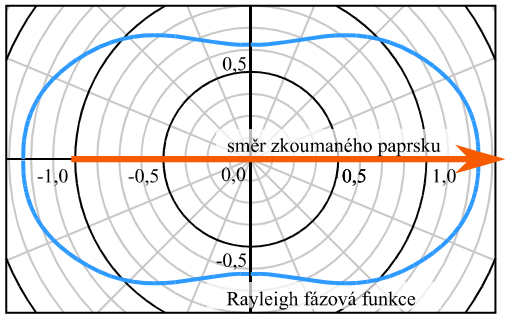
Fázová funkce může být popsána několika způsoby – ty se liší zejména typem vykreslovaného média. Nejjednodušší fázovou funkcí je isotropická fázová funkce – ta má následující tvar [3]:

$$p(\cos\theta)=\frac{1}{4\pi} \quad (5)$$

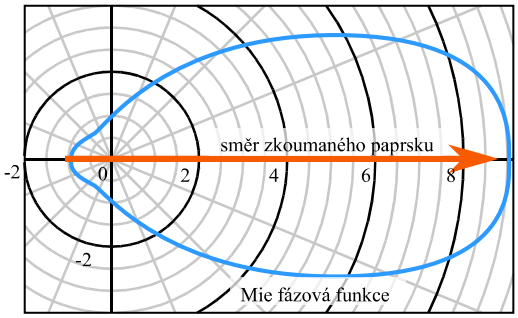
Jak je vidět, funkce není vůbec závislá na směru paprsku a má stejnou hodnotu pro všechny jeho směry.

Další funkcí je Rayleigh – ta se hodí zejména pro simulaci látek skládajících se z částic menších jak desetina vlnové délky světla (přibližně velikost molekul). Například cigaretový kouř nebo atmosféra. Funkce má následující tvar [3]:

$$p(\cos\theta)=\frac{3}{\xi}\frac{1+\cos^2\theta}{\lambda^4} \quad (6)$$



Obrázek 12: Znázornění Raileigh fázové funkce



Obrázek 12: Znázornění Mie fázové funkce

Pro zobrazení objemů, ve kterých jsou částice větší slouží Mie fázová funkce (její průběh ilustruje obrázek 13). Tato funkce je vhodná například pro zobrazování mraků a mlhy [4], [1]. Jelikož se jedná o velmi komplexní funkci, je Mie fázová funkce často aproximována jinou funkcí, v některých případech v závislosti na typu zobrazovaného média. Mezi tyto funkce patří například Henyey-Greensteinova fázová funkce nebo Schlickova fázová funkce. Poslední zmíněná je vhodná díky své rychlosti a snadné implementaci při raytracingu pomocí Monte-Carlo metod [1].

Nyní, když jsou definovány všechny složky, které se podílejí na osvětlení částic uvnitř média, je možné sestavit funkci, která toto osvětlení popisuje [1]:

$$\frac{dL(x)}{dx}=\underbrace{\kappa_a(x)L_e(x)}_{emise}+\underbrace{\frac{\kappa_s(x)}{4\pi}\int_S L(x,\omega_i)\,p(\omega_o,\omega_i)\,d\sigma_{\omega_i}}_{in-scattering}-\underbrace{\kappa_a(x)L(x)}_{absorbce}-\underbrace{\kappa_s(x)L(x)}_{out-scattering} \quad (7)$$

S reprezentuje směry, z nichž do bodu x přihází světlo. Rovnici lze dále zapsat jako

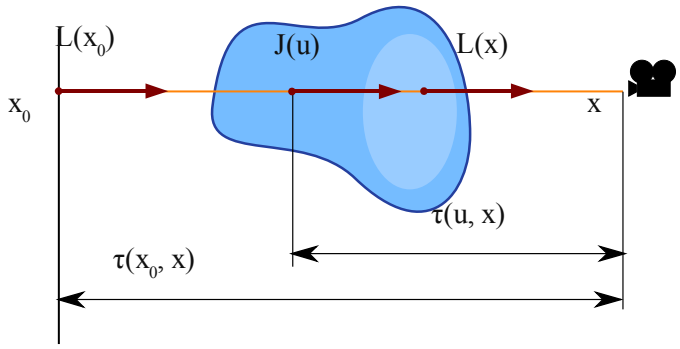
$$\frac{dL(x)}{dx}=\kappa_t(x)J(x)-\kappa_t(x)L(x) \quad (8)$$

J(x) je radiance, která v daném bodě „vzniká“ buďto vyzařováním materiálu nebo odrazem světla z jiného směru. L(x) je naopak radiance, která v daném bodě „zaniká“ absorbcí materiálu nebo rozptylem do jiného směru. Tyto dvě radiance lze dále rozepsat:

$$J(x)=(1-\Omega(x))L_e(x)+\frac{(\Omega(x))}{4\pi}\int_S L(x,\omega_i)\,p(\omega_o,\omega_i)\,d\sigma_{\omega_i} \quad (9)$$

$\Omega = \frac{\kappa_s}{\kappa_t}$ označuje tzv scattering albedo – vlastnost popisující úbytek radiance odrazem. Integrováním (8) vznikne následující rovnice (nazývaná také integral transport equation, viz obrázek 14) [1]:

$$L(x) = \tau(x_0, x) L(x_0) + \int_{x_0}^{x} \tau(u, x) \kappa_t(u) J(u) du \quad (10)$$



Obrázek 14: Ilustrace k "Integral transport equation"

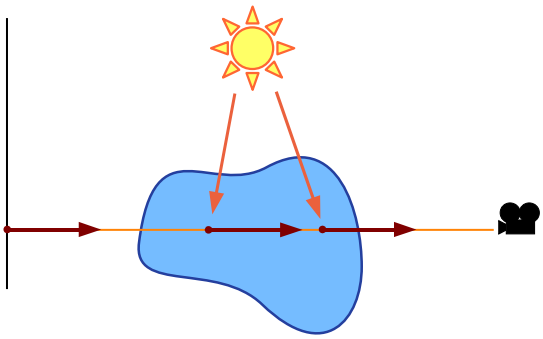
Světlo by se mohlo uvnitř média odrážet donekonečna, to by vedlo ale k velmi složitým výpočtům. Proto se pro je při výpočtu počet odrazů omezen.

Pokud nedochází k rozptylu, je $\kappa_s = 0$ – tento případ nastává zejména v situacích kdy vykreslovaná látka produkuje vysoké množství světla, a tak lze vnitřní odrazy zanedbat, aniž by se to projevilo ve výsledku nebo k vnitřním odrazům vůbec nedochází. Příkladem takové látky je oheň. Za uvedeného předpokladu $\kappa_s = 0$ lze uvažovat $\kappa_t = \kappa_a$, $\Omega(x) = 0$ a $J(x) = L_e$ (radiance vyzařovaná látkou). Rovnice osvětlení v daném bodě pak vypadá takto [1]:

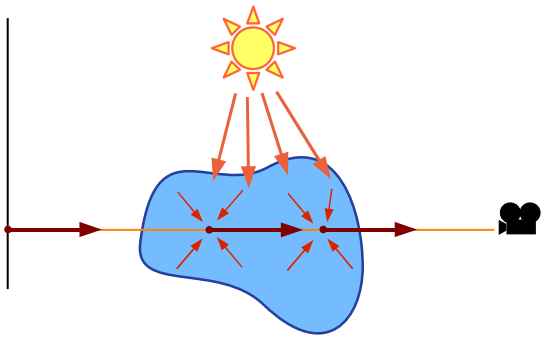
$$L(x) = \tau(x_0, x) L(x_0) + \int_{x_0}^{x} \tau(u, x) \kappa_a(u) L_e(u) du \quad (11)$$

Pokud se navíc jedná o homogenní těleso bez vlastního vyzařování, lze rovnici zjednodušit až na tento tvar [1]:

$$L(x) = e^{-\kappa_a \|x_0 - x\|} L(x_0) \quad (12)$$



Obrázek 15: Jednonásobný odraz paprsku



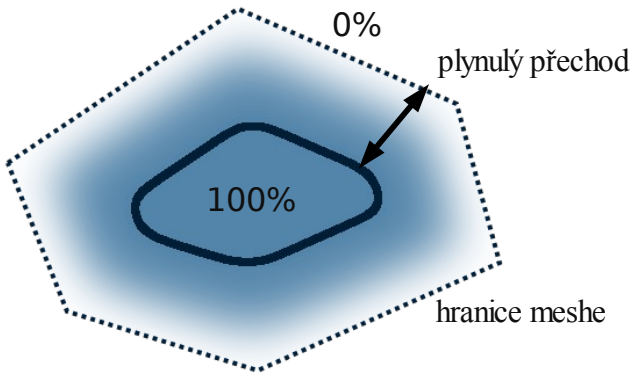
Obrázek 16: Vícenásobný odraz paprsku

Jednonásobný odraz lze uvažovat u látek, které jsou velmi řídké (téměř průhledné) nebo málo dochází k vnitřním odrazům. U ostatních těles je často nutné použít vícenásobné odrazy, občas však i výpočet jednonásobného odrazu s vhodnými parametry poskytne dostatečně uspokojivé výsledky. Rozdíl mezi jednonásobnými a vícenásobnými odrazy ilustrují obrázky 15 a 16.

3.3 Reprezentace objemových dat

Jak bylo v předchozí podkapitole řečeno, je nutné zvolit vhodnou reprezentaci média při jeho modelování. Nejjednodušší variantou je jednoduchá látka s konstantní hustotou částic ve všech bodech jeho objemu. S tímto přístupem ovšem nelze modelovat složitější struktury (většina mraků nebo kouře) a hodí se spíše např. na jednoduchou mlhu. Proto se pro definování hustoty a dalších vlastností používá funkce která pro každý bod prostoru vrací jeho optickou hustotu, případně barvu nebo fázovou funkci. Jednou z možností je dodání dat explicitně – tedy formou připravené 3D textury uložené v souboru, výhodou je jednoznačnost této reprezentace, často to však vede k objemným souborům a mnohdy není tato jednoznačnost důležitá. Nejvhodnějším způsobem reprezentace pro účely počítačové grafiky je implementace kombinací procedurálních textur nebo jiných fraktálních funkcí pro strukturu a definování objemu média pro určení jeho tvaru a pozice. Procedurální textury jsou již ve vybraném modelovacím nástroji přítomny – jsou k dispozici různé druhy šumů a o přechody.

Dále je potřeba najít funkci určující hustotu látky bez ohledu na její strukturu – tedy lépe řečeno místa ve scéně ve kterých se bude kouř nebo mrak nacházet. K tomu poslouží v tomto případě obyčejné objekty, je však třeba vytvořit funkci, která převede jejich plošnou reprezentaci na reprezentaci objemovou, tak, aby byl přechod mezi hranicemi objektu a médiem plynulý (tedy aby nebyl mrak nebo kouř ostře ohraničen polygony daného objektu). To lze udělat například výpočtem vzdálenosti bodu v prostoru od nejbližší hrany objektu a v závislosti na této vzdálenosti lze potom určit hustotu látky v daném místě. Situaci ilustruje obrázek 17.



Obrázek 17: Hustota látky uvnitř meshe

Hustotu vypočítanou pro strukturu pak stačí vynásobit hustotou vypočítanou pro objem a výsledkem je konkrétní hustota částic v daném bodě.

Dalšími možnostmi je například použití blobů – tedy bodů v prostoru které mají dánu intenzitu a okolní intenzita v prostoru je vypočítána z těchto bodů. Při vytváření složitějších objektů je potřeba požadovaný tvar vymodelovat vhodným umístěním několika blobů do prostoru.

3.4 Metody používané pro vykreslování objemových dat

Pro samotné vykreslování média se používá několik metod. Ty se liší zejména rychlostí, náročností na paměť a fyzikální přesností. Mezi nejrychlejší patří tzv. Fake Media metody, které jsou orientované především na efektní výsledek a nemusí vycházet z fyzikálních zákonů. Daný objekt není zobrazován jako objemová data, ale je něčím nahrazen. Mezi tyto metody patří například částicové systémy (kde je objem nahrazen velkým množstvím plošek).

Dalšími přístupy jsou metody analytické, u nichž se předpokládá výrazné zjednodušení matematického modelu a následné řešení vzniklých rovnic. Dále deterministické, které řeší vztahy uvnitř tělesa numerickými metodami. Posledními používanými metodami jsou stochastické, které využívají náhodného vzorkování a interpolace. Pro účely práce se nejlépe hodí stochastické metody, jelikož je lze považovat za nejobecnější – analytické a deterministické postupy jsou často vázány na určitý druh média (mraky, kouř, mlha...) [1]. Analytické metody také nejsou uzpůsobeny pro práci s vícenásobnými odrazy (vedlo by to ke složitým výpočtům). Vzorkování je navíc v počítačové grafice hojně používáno pro řešení dalších problémů spojených např. s osvětlováním, výpočtem barvy povrchu nebo odrazu.

U metod které pracují s vícenásobným odrazem paprsku, bývá proces vykreslování rozdělen do dvou částí – výpočet samotného osvětlení uvnitř objemu média a následné vykreslení média. Tento způsob je náročný na paměť, jelikož je nutné uchovat informace o osvětlení jednotlivých bodů v prostoru. Proto je nutné zvolit vhodné rozlišení média, tak aby se tyto informace vešly do operační paměti počítače. Na druhou stranu je toto řešení vhodné pro animace (pokud se médium nepohybuje), jelikož vypočítaná data lze uložit a použít při dalších snímcích animace.

Stochastické metody se dále dělí pode toho, jakým způsobem se vzorky vytvářejí [1] – buďto pravidelně nebo nepravidelně (tedy vzdálenosti mezi vzorky jsou náhodné). Pro výpočet osvětlení a barvy bodu se v obou případech vychází ze vztahů uvedených v podkapitole 3.2. Tato metoda je použita při implementaci zobrazovacího algoritmu v této práci – je použito pravidelné vzorkování.

4 Volba nástroje pro implementaci

4.1 Kritéria

V zadání práce je uvedeno, že do zvoleného nástroje má být doplněna nová funkcionalita. Proto je v prvé řadě nutné, aby byly k vybranému nástroji dostupné zdrojové kódy nebo aby byl modulární (podporoval příslušný typ pluginů). K tomuto účelu se hodí zejména aplikace vydané pod licencí GNU-GPL nebo jinou otevřenou licencí.

Dalším podstatným kritériem je dostupnost vhodného modelovacího nástroje, schopného s vybraným rendererem spolupracovat. Tyto data by sice bylo možné popsat např. V textovém souboru, tím by se ale výrazně snížila použitelnost výsledného řešení – nevizuální začlenění do scény by bylo značně problematické. Tento nástroj by měl být schopen poskytnout možnost data modelovat a připravit k vyrenderování, důraz je kladen především na schopnost pracovat s volumetrickými daty, která budou reprezentovat kouř nebo mraky.

Kromě dvou výše uvedených hlavních požadavků by měl být zvolený nástroj dostatečně aktuální a dobře podporovaný. Dalším předpokladem je dobrý stav zdrojového kódu a také jeho způsobilost začlenit do sebe nové části, které budou implementovat přidanou funkcionalitu. Dalším požadavkem je podpora pro import scény z nějakého vhodného modelovacího nástroje. Mezi, v současné době nejkvalitnější volně dostupné nástroje pro modelování patří Blender navíc s ním má autor práce zkušenosti, proto bude požadavkem také schopnost rendereru spolupracovat s tímto nástrojem.

V následujících podkapitolách budou krátce shrnuty vlastnosti několika nástrojů pro realistické zobrazování scén, které jsou vhodné pro účely této práce. Byly vybrány:

- POV-ray
- YafRay a YafaRay
- Blender Internal Renderer (součást aplikace Blender)

Všechny výše uvedené aplikace podporují renderování scény z aplikace Blender a jsou schopny pracovat s jeho procedurálními texturami. Podrobněji jsou jejich vlastnosti popsány v následujících podkapitolách.

4.2 POV-Ray

Tento nástroj je ze všech uvedených nejstarší, to s sebou nese výhody i nevýhody. Hlavními výhodami je dobrá odladěnost kódu a poměrně dobrá podpora v mnoha aplikacích. Mezi nevýhody patří především jeho relativní zastaralost. POV-Ray je sice stále schopen podávat dobré výsledky, ale v dnešní době je již překonán novějšími a modernějšími nástroji.

Tento renderer scénu načítá s textového souboru, a proto ani není potřeba mít k dispozici nějaký modelovací nástroj. Vytváření scény pomocí skriptu je však vhodné spíše pro generování komplexních těles (např. zobrazení matematických funkcí). Export scény z Blenderu je řešen pomocí skriptu PovAnim. Jsou k dispozici také speciální verze Blenderu které umožňují scénu renderovat přímo z aplikace podobně jako v případě YafRaye. Díky tomu že POV-Ray zpracovává textové soubory, lze scénu snadno upravit a přidat do ní např. prvky které modelovací nástroj nepodporuje.

4.3 YafRay a YafaRay

Z uvedených nástrojů je YafRay nejmladší. Jeho hlavní předností je vysoká kvalita výsledného obrázku a dobrá spolupráce s Blenderem. Na rozdíl od ostatních externích rendererů není export do YafRaye řešen pomocí skriptů, ale je přímo implementován do Blenderu. Také ve vlastnostech scény, světel a kamery lze nastavit některé parametry specifické pouze pro YafRay. Nevýhodou je poslední dobou poměrně pomalý vývoj – YafRay je od začátku vyvíjen pouze malou skupinou vývojářů a rychlost vývoje je tedy velmi závislá na volném osobním čase několika málo nadšenců. Asi před rokem započala práce na nové verzi, která nese název YafaRay. Bohužel není jisté kdy bude tato verze stabilní a jak široce bude spolupracovat s Blenderem. Obě verze disponují pokročilými osvětlovacími technikami jako Ambient Occlussion a Global Illumination a proto lze snadno dosáhnout realisticky vypadajících výsledků. Díky tomu je výsledný render scény také poměrně časově náročný. Obdobně jako u POV-Raye lze scénu načítat z textového (v tomto případě XML) souboru, což umožňuje snadnou editaci scény a zpřístupnění funkcí, které nejsou dostupné z uživatelského rozhraní Blenderu.

4.4 Blender Internal Renderer

Tento renderovací nástroj je přímo součástí aplikace Blender. Z toho plyne především výborná podpora všech jeho vlastností přímo v editoru při vytváření scény. Za vývojem této aplikace stojí organizace Blender Foundation, která získává prostředky pro vývoj a poskytuje dobré zázemí pro vývojáře. Díky tomu, že je Blender neustále vyvíjen jako celek, dochází také k postupnému vylepšování jeho interního rendereru. Navíc je vyvíjen větší skupinou lidí, takže lze očekávat velký důraz na dobrou čitelnost zdrojového kódu. Na druhou stranu se zdrojový kód díky stálému vývoji často mění a s tím je třeba také počítat při doplňování nových funkcí.

Hlavní nevýhodou jsou méně propracované osvětlovací modely, než nabízí výše zmíněný YafRay nebo POV-Ray. Důsledkem je náročnější osvětlování scény z pohledu počítačového grafika – pro dosáhnutí realistického výsledku je potřeba vynaložit větší úsilí a použít více světel, což vede ke zpomalení celého procesu tvorby, ladění a renderování scény. Na druhou stranu je tento renderer v mnoha situacích oproti ostatním rychlejší.

4.5 Výběr nástroje pro implementaci

V předchozích odstavcích byly popsány některé nástroje které by byly vhodné pro účely této práce. Autor se všemi nástroji zběžně seznámil. Z těchto nástrojů byl pro výslednou implementaci nakonec zvolen Blender Internal Renderer. Hlavní motivací bylo zejména to, že je integrován do Blenderu a tak umožňuje dobře provázat editor, ve kterém budou mraky nebo kouř modelovány a renderer, ve kterém bude algoritmus implementován. Další výhodou je také neustálý vývoj a rozšířenost této aplikace. Navíc s ním má autor zkušenosti, což dovolí co nejlépe využít jeho stávající potenciál při implementaci.

5 Blenderu, jeho architektura a Renderovací Engine

5.1 Blender – 3D animační software

Blender byl vybrán jako nástroj, do kterého bude výsledný algoritmus implementován a proto je třeba se s ním blíže seznámit [8][10]. Původně byl vyvíjen jako modelovací a animační nástroj pro interní potřeby grafického studia NeoGeo. Po zániku studia byly práva převedena na společnost NaN a časem byl Blender uvolněn jako freeware, vývoj však ustal. V roce 2002 byla hlavním programátorem a tvůrcem Blenderu – Tonem Roosendaalem založena nadace Blender Foundation. Jejím prvním cílem bylo získat prostředky na odkoupení práv na zdrojové kódy – to se povedlo koncem roku 2002 a Blender byl uvolněn pod licencí GPL.

Blender je tedy vyvíjen již 13 let a byl od počátku koncipován jako velmi komplexní, robustní a víceúčelový nástroj. Podporovány jsou všechny rozšířené platformy - Windows, Linux (i386, amd64, PowerPC), Mac OS, FreeBSD a několik dalších. V aplikaci jsou obsaženy mimo jiné následující nástroje[1].

- 3D editor – Toto je původní a také primární funkce Blenderu. Disponuje nástroji pro práci s polygoniálním modelem, jeho animaci, editor materiálu, umožňuje jednoduché logické operace a mnoho dalších funkcí.
- Blender Python API – Za pomoci tohoto API lze v Blenderu vytvářet skripty které přímo ovlivňují scénu, její objekty nebo chování Blenderu. Skripty v Pythnou jsou také přímo součástí Blenderu – zejména se jedná o prvky GUI, nástroje pro generování meshů nebo nově přidávané a testované nástroje a schopnosti.
- Blender Game Engine – Umožňuje tvorbu interaktivních aplikací – například her nebo prezentací. Jeho součástí je také fyzikální engine, který také slouží jako prostředek pro tvorbu

komplexnějších animací. K dispozici je Sumo nebo Bullet Physics, které umožňují simulování pádu a dalšího chování fyzikálních těles a jejich kolizí.

- Nástroje pro sculpting – Slouží k definování složitější struktury povrchu pomocí simulace přidávání nebo odebírání materiálu – používá se například při tvorbě vrásek nebo hrubé struktury povrchu u které by bylo modelování náročné a použití výškové mapy nevhodné.
- Materiálový Node Editor – Je určen pro tvorbu komplexních materiálů – umožňuje definovat materiál a jeho barvu složením ostatních materiálů, ale také s využitím informací o aktuálních vlastnostech povrchu (např normály, UV koordinátů apod.).
- Kompozitní Node Editor – Nástroj pro zpracování výsledku renderování (který může mít několik vrstev). Nabízí nástroje pro barevné korekce a další filtry, umožňuje pracovat nebarevnými informacemi renderu (hodnota Z-bufferu, normála, čas …).
- Částicové systémy – kromě jednoduchých částicových systému jsou k dispozici tzv statické částice (chlupů a vlasů) a pokročilé druhy částic jako jsou boids (simulace hejna – např. ryb)
- Soft Body – Simulaci chování látky – umožňuje interakci s vnějšími silami (např. Větrem) a s ostatními objekty - kolize.
- Fluids – Simulace chování tekutin.
- Blender Internal Rendering Engine – Renderovací engine s podporou několika druhů světel a stínů, raytraycingu, radiosity, ambient occlusion, subsurface scattering, zapékání a dalších metod pro podporu realistického zobrazování scény.

5.2 Vývoj Blenderu

Jak bylo v předchozí kapitole uvedeno, Blender byl původně vyvíjen jako aplikace pro vnitřní potřeby studia. Původním vývojářem je Ton Roosendaal (pozdější zakladatelem Blender Foundation). Po zániku grafického studia NeoGeo (jež bylo zaměřeno na tvorbu 3D animací) převzala Blender společnost NaN, která se orientovala především na jeho vývoj a prodej. Po jejím zániku následovala krátká prodleva způsobená převodem autorských práv a poté načas pokračoval s vývojem pouze původní autor.

V současné době je Blender vyvíjen především v rámci realizace tzv. „Open Movie“ - tedy vytváření animovaných filmů s licencí Creative Commons. Tyto animované filmy jsou financovány sponzory a prodejem DVD (které však lze také legálně stáhnout z Internetu) a pro vývoj Blenderu jsou velmi důležité. Součástí vzniků těchto filmů je totiž také přidávání nových vlastností, které jsou ve filmu použity (např. pro animaci, renderování, postprocesing apod.). V současné době jsou tyto filmy prozatím dva:

- Elephants Dream (Září 2005 - Březen 2006) – Hlavním přínosem byl materiálový a kompozitní Node Editor.

- Big Buck Bunny (Říjen 2007 - Květen 2008) – Při vývoji byl implementován nový částicový systém určený pro práci s „vlasovými“ materiály jako srst, vlasy apod., podpora pro nové druhy stínů, matné reflexe a další vlastnosti.

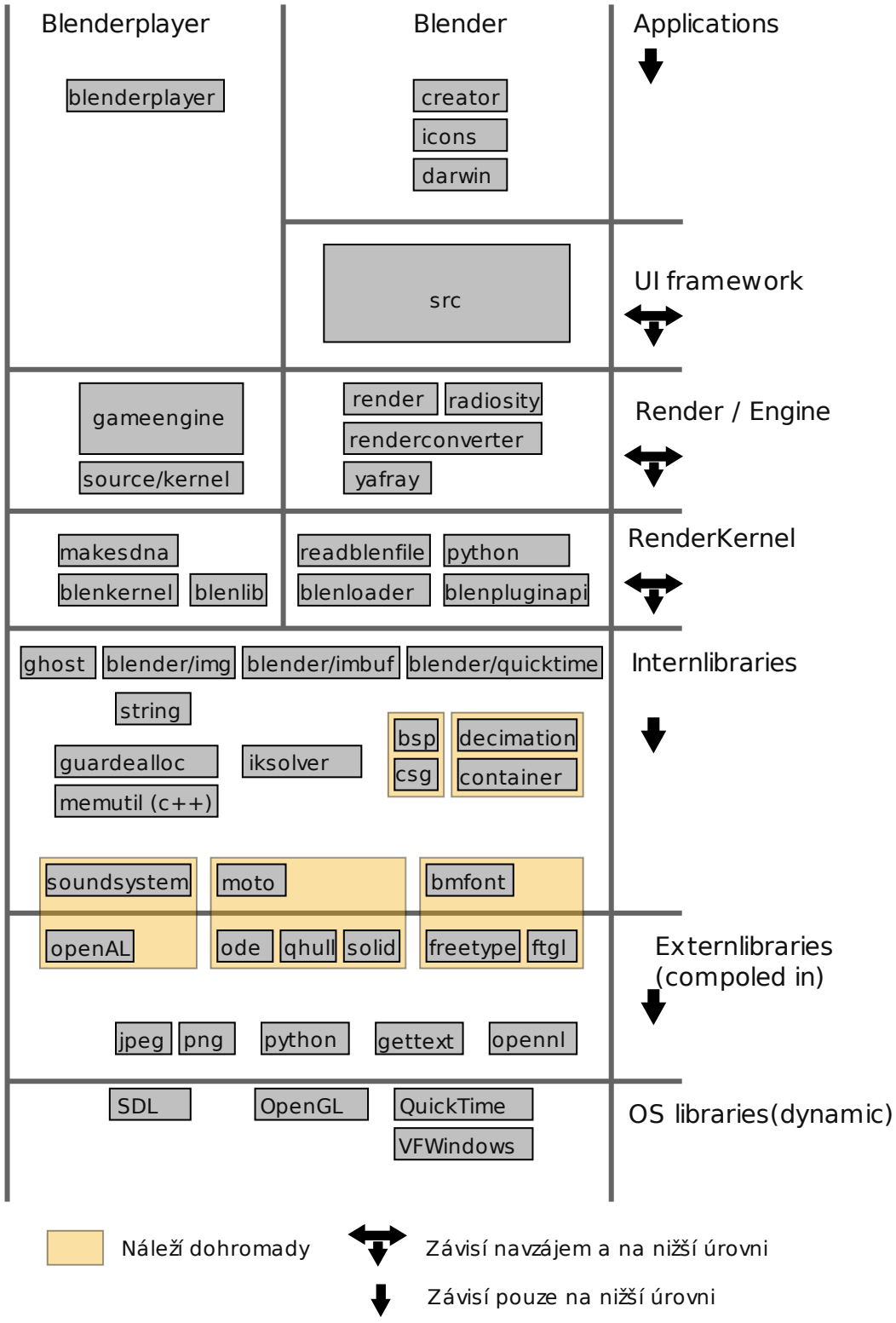
5.3 Vnitřní struktura aplikace Blender

Jak je z předchozí podkapitoly patrné, je Blender velmi rozsáhlá aplikace a tomu také odpovídá jeho vnitřní struktura. Téměř nepřetržitý 13ti-letý vývoj svědčí o dobrém původním návrhu a snadné rozšiřitelnosti. Přesto že byly a jsou některé jeho části přepisovány, podstata jádra a princip fungování se nemění.

Všechny části popsané v kapitole 5.1 jsou do Blenderu plně integrovány a navzájem provázány. Vzhledem k tomu, že jsou kladeny požadavky na malou odezvu programu a rychlý výstup, je Blender z převážné části napsán v jazyce C. Externí knihovny a některé méně náročné součásti (např. GUI, nebo některé nástroje pro editaci) jsou napsány v Pythonu nebo C++. Díky použití jazyku C u výpočetně náročných operací a v jádře je zajištěna vysoká rychlost, oproti C++ jsou však zdrojové kódy méně přehledné a intuitivní. Použití objektů se při návrhu 3D editoru přímo nabízí, je však pro některé situace méně efektivní. Jelikož se jedná o výpočetně náročnou aplikaci, mnoho operací podporuje více vláken a proto má Blender také svou vlastní správu paměti a pro některé součásti také garbage collector. Schematické znázornění struktury aplikace je zobrazeno na obrázku 18 []. Místo objektů jsou v Blenderu použity struktury – často velmi rozsáhlé a navzájem propojené. Při práci s daty scény (např. meshem nebo materiálem) jsou pak funkci předávány ukazatele na tyto struktury.

Na tak rozsáhlý projekt, jakým Blender bezesporu je, je jeho dokumentace poměrně stručná a neaktualizovaná (to je způsobeno především rychlým vývojem). Nabízí jen velmi slabou podporu pro orientaci v aktuálních zdrojových kódech, a proto je seznamování se s principem činnosti poměrně obtížné. Hlavním vodítkem při porozumění fungování Blenderu jsou tak hlavně zdrojové kódy. Ze začátku byl největší problém se orientovat v rozsáhlých a propojených strukturách.

Blender má velmi originální a sofistikovaný způsob ukládání svých pracovních souborů. V *.blend* souboru je uložena kompletní scéna včetně objektů, materiálů, animací, skriptů a dalších součástí projektu a podporuje také připojení souborů které scéna využívá (např. textury nebo fonty). Každé sestavení Blenderu a každý *.blend* soubor v sobě mají uloženy tzv. DNA – v té jsou obsaženy metadata které daná verze využívá (např. délky číselných datových typů, položky struktur apod.) a tak je zajištěna obousměrná kompatibilita Blenderu s jeho pracovními soubory v různých verzích aplikace. Při ukládání do souboru konkrétní verze Blenderu zkopíruje své DNA. Při otevírání si ji opět přečte a díky tomu je zajištěna jednoznačná interpretace souboru (neznámá data jsou ignorována) [8].



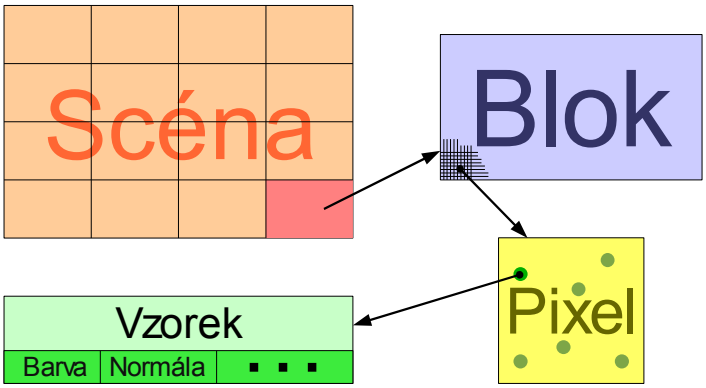
Obrázek 18: Schéma závislostí součástí Blenderu

Jednotlivé logické celky jsou v Blenderu často rozděleny. Jednou částí je samotný algoritmus nebo funkční úsek (např. GUI, práce s materiály, renderer) a druhou částí je definice typů. Kvůli tomu jsou často datové typy, které se používají při práci s Blenderem nebo pro ukládání, odděleny od samotného kódu který implementuje výpočet. To na jednu stranu znepřehledňuje konkrétní

algoritmus (jelikož se musí některé datové typy dohledávat jinde), na druhou stranu to ale umožňuje snadno kontrolovat data ukládaná do *.blend* souboru a obecné funkční celky mohou využívat všechny struktury se kterými se při práci s Blenderem pracuje.

5.4 Blender Internal Rendering Engine

Největší pozornost je potřeba věnovat rendrovacímu engine Blenderu, jelikož do něj bude algoritmus integrován. Renderer Blenderu je plně škálovatelná součást aplikace. Vykreslování probíhá hierarchicky - viz obrázek 19:



Obrázek 19: Hierarchie renderu při jeho vykreslování

Výstupní obraz je nejprve pravidelně rozdělen na stejně velké oblasti a každá z těchto částí je nezávisle na ostatních vykreslena – v případě povolení škálování je použito více nezávislých vláken. Každá část se skládá z jednotlivých pixelů a ty jsou složeny ze vzorků – pokud není povoleno nadvzorkování (OSA – OverSAmpling), má každý pixel pouze jeden vzorek. V opačném případě může mít 3, 5, 8 nebo 16 vzorků. Ty většinou obsahují informace o barvě, pokud se výstup dále zpracovává v kompozitoru, můžou obsahovat také informace o:

- hloubce pixelu (z-buffer)
- směru pohybu
- normále, texturovacích koordinátech
- indexu objektu
- barvě povrchu – bez shaderu, diffuse shader a specular shader
- stínu
- odrazech a lomech
- ambient occlusion a radiositě

Každý z bloků je vykreslován postupně – v případě použití vrstev po vrstvách, pixel po pixelu, každý pixel po vzorcích. V každém bloku je alokována speciální pomocná struktura, která obsahuje informace o právě zpracovávaném vzorku a je inicializována na začátku vykreslování bloku. Pro každý vzorek je naplněna informacemi o jeho pozici, objektech a trojúhelnících které se do něj mohou promítnout při vykreslování, jejich normálách pozicích, texturovacích koordinitech a dalších

informacích potřebných k jeho zobrazení. Potom jsou pro každý vzorek na základě těchto informací dopočítány postupně:

- texturovací koordináty
- barvy textur (buďto procedurálních nebo ze souboru – obrázku) a jejich kombinace
- normála (která může být texturou ovlivněna)
- skutečná barva povrchu (vypočítaná shadery)
- informace o průhlednosti

V případě průhledných objektů se nejdříve seřadí trojúhelníky, které mají na barvu pixelu vliv a poté jsou vykresleny postupně odpředu dozadu - skončí se posledním průhledným trojúhelníkem nebo pozadím.

Každý objekt může mít přiřazen jeden nebo více materiálů - pro různé polygony. Tyto můžou být buďto základní nebo „sestavené“ pomocí Nodů. Ve vlastnostech materiálu jsou určeny jeho základní parametry jako je barva, průhlednost, odrazivost apod.. Texturování je v Blenderu řešeno pomocí texturovacích slotů, kterých je k dispozici 10 pro každý materiál. Do každého slotu lze nastavit samostatnou texturu a nezávisle nastavit její mapování a parametry materiálu které ovlivňuje (barvu, průhlednost, normálu, odrazivost, lesklost apod.). Textury se přes sebe překrývají podobně jako vrstvy v 2D editoru – výsledná barva vznikne překrytím s využitím alfa kanálu nebo různými operacemi přičítání nebo odečítání barev. Pokud je potřeba použít více než 10 textur, je třeba je „namixovat“ předem nebo využít materiálový Node Editor.

Pro mapování textur lze využít:

- koordináty globální a lokální nebo koordináty jiného objektu
- koordináty UV, okna nebo kamery (sticky - druh automaticky generovaných UV koordinátů)
- reflexní mapování a mapování podle normál nebo pnutí v materiálu (při animaci meshe)
- tzv. tangent mapování (využívá se např. pro zobrazování broušených kovů)

Pro účely této práce je důležitá zejména podpora lokálních a globálních koordinátů objektů, jelikož jsou definovány obecně v 3D prostoru - na rozdíl od ostatních, kde jsou k dispozici pouze 2 souřadnice.

6 Implementace algoritmu

6.1 Propojení algoritmu s Blenderem

Jako první bylo třeba vyřešit vytváření mraků a kouře v 3D editoru Blenderu. Z výše popsaných metod (kapitola 3.3) byla vybrána reprezentace pomocí obyčejného objektu, který bude mít přiřazen speciální materiál. Pro tento účel byl do materiálu přidán další parametr – *typ materiálu*, který ve fázi vykreslování rozlišuje mezi obyčejným (*Normal*) materiálem, *Halo* materiálem (používaným pro vykreslování částic) a speciálním materiálem pro mraky a kouř nazvaným *Cloud*. Bylo také potřeba zpřístupnit tento parametr v GUI Blenderu – editoru materiálu a také přizpůsobit tento editor tak, aby bylo možné v něm nastavit další vlastnosti mraků a kouře. Ukázka upraveného GUI je zobrazena v příloze 3.

Hlavní nevýhodou tohoto řešení je nekorektní chování algoritmu v situacích, kdy objekt reprezentující mrak nebo kouř není uzavřen a nemá souvislý povrch. Tento nedostatek však není příliš významný, jelikož při chybě lze tento problém poměrně snadno najít a odstranit. Takové objekty jsou navíc pro modelování obecně nevhodné, protože přinášejí problémy také v jiných oblastech (např. vyhlazovacích algoritmech) a zkušení 3D grafici se jim vyhýbají. Původně měla být hustota média v meshi plynule měněna od středu k okrajům (jak ukazuje obrázek 17 v kap. 3.3), ovšem toto řešení se ukázalo z hlediska realizace jako poměrně problematické a proto od něj bylo upuštěno. U jednoduchých objektů lze však podobného výsledku docílit použitím vhodné kombinace přechodových (*Blend*) textur.

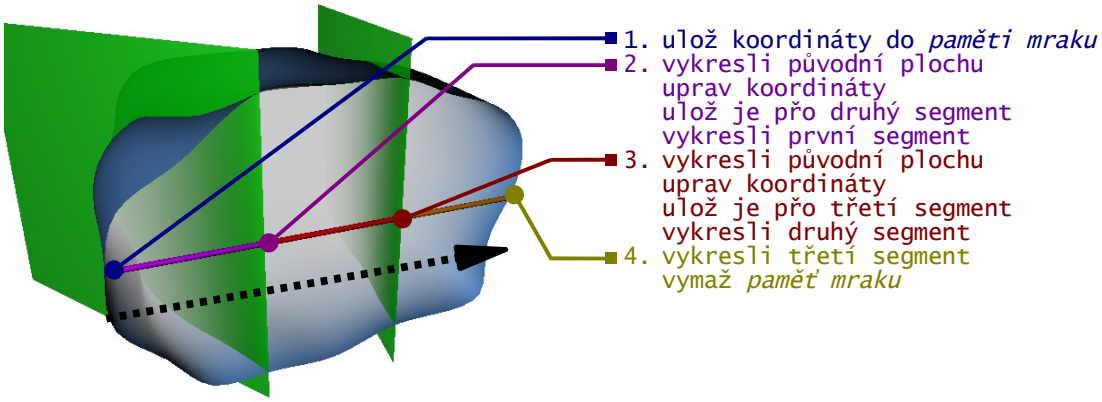
V interním renderovacím engine Blenderu jsou jeho jednotlivé části navzájem úzce propojeny (osvětlení, stíny, práce se z-bufferem apod.), a proto byl algoritmus implementován přímo do něj. Zobrazování mraků totiž ovlivňuje výpočet vzorku v několika úsecích jeho zpracování a tak není možné jej jednoduše přidat na začátek nebo na konec zobrazovacího procesu. Zároveň jsou při vykreslování přímo použity některé již implementované funkce, které se používají při vykreslování povrchů obyčejných objektů (zejména texturování).

Při začleňování algoritmu do procesu vykreslování bylo třeba vzít v potaz tyto skutečnosti:

- Interní renderer Blenderu je výrazně orientován na vykreslování ploch – trojúhelníků a quadů
- Je třeba zajistit, aby se mraky, jakožto objemová data, správně vykreslily také pokud se protínají s nějakým dalším objektem (obyčejným nebo jiným objektem mraků).
- Oblast kde se mrak nebo kouř nachází je definována pomocí meshe.

Tyto podmínky a požadavky vedly k tomu, že jsou mraky vykreslovány spolu s plochami. Každá plocha která má na vykreslování vliv je buď jeho ohraničením nebo jej rozděluje na části. Pro všechny tyto části je pak potřeba počítat úsek mraku zvlášť (viz Obrázek 20). Pokud je daná plocha rozdělující mrak částečně průhledná, je potřeba zajistit její správně prolnutí s mraky.

Vykreslování je tedy realizováno tak, že při výpočtu každé plošky v rámci vzorku je proveden test na typ materiálu. Pokud se jedná o mrak, je to jeho hranice – tedy počátek nebo konec. Jestli-že žádná plocha stejného typu ještě v aktuálním vzorku nebyla vykreslována (není ve speciální proměnné), jedná se zřejmě o začátek – v tom případě je třeba uložit údaje o pozici a texturovacích koordinátech, aby mohly být použity při následném vykreslování přilehlého úseku. Tyto informace jsou proto uchovány právě ve speciální proměnné (lze ji nazvat např. *paměť mraku*). Pokud je tato *paměť* naplněna, je při vykreslování dalších plošek vzorku zřejmé, že aktuální ploška se nachází uvnitř mraku nebo je jeho zakončením.



Obrázek 20: Rozdělení objektu na segmenty

Po vykreslení této plošky je proto spuštěn samotný algoritmus zobrazování mraku a kouře, zároveň je aktualizována *paměť mraku*, tak aby při vykreslování dalšího úseku obsahovala potřebné hodnoty. Barva mraku je potom podle jeho průhlednosti připočtena k původní barvě materiálu plochy a tím se vyvolá dojem, že je plocha za aktuálním segmentem mraku, pak je upravena také celková průhlednost plochy (jelikož ji mrak ovlivnil). Při ukončení vykreslování mraku jeho druhou plochou je *paměť mraku* resetována a tím se ukončí jeho vykreslování – u dalších ploch se již žádné segmenty nepočítají. Ve skutečnosti je speciálních proměnných (*pamětí mraků*) více, aby bylo umožněno zobrazit mraky které se prolínají – pro každý mrak musejí být tyto informace uloženy zvlášť.

6.2 Zpracování struktury mraku a kouře

Jak bylo v kapitole 3.3 uvedeno, je pro reprezentaci struktury kouře použita procedurální textura. Blender má naštěstí již k dispozici poměrně širokou nabídku procedurálních textur, které jsou vytvořeny pro obecné použití. Pro generování mraků a kouře je nevhodnější textura „Clouds“ která, jak již název napovídá, je přímo určena pro zobrazování struktury mraků – původně však pouze na plochách. Výhodou je také možnost ovlivnit druh šumu který se má použit pro generování této textury – k dispozici je:

- Blender Original
- Perlin a Improved Perlin

- Voronoi – několik druhů
- Cell noise

Kromě druhu šumu lze také definovat jeho měřítko, hloubku rekurze (členitost) a barevný přechod (včetně průhlednosti) do kterého budou hodnoty šumu promítnuty.

Jak již bylo výše uvedeno, je pro zobrazování mraků použita stochastická metoda – tedy vzorkování. Proto je potřeba pro každý vykreslovaný úsek mraků získat určitý počet vzorků barvy a průhlednosti z textury. Z tohoto pohledu bylo hlavním problémem při implementaci algoritmu zajištění správných parametrů pro funkci počítající barvu povrchu v daném bodě. Vzhledem k povaze mraků a kouře (jedná se o prostorová data) má smysl využívat při jejich vykreslování pouze některé druhy mapování textury:

- Glob – Globální koordináty odpovídající pozici bodu v prostoru scény.
- Orco – Lokální koordináty objektu na který je materiál aplikován – jsou ovlivněny pozicí jeho počátku, velikostí a natočením.
- Object – obdobné jako u Orco, ale pro výpočet koordinátu je použit jiný objekt scény. Toto mapování je vhodné např. pokud je potřeba zajistit plynulé navazování procedurální textury mezi různými objekty nebo v případě animace (původní objekt zůstává na místě a hýbe se pouze objekt na který je mapována textura).

Při vykreslování úseku mraků mezi dvěma plochami je pro účely vzorkování nutné znát koordináty v místě každého vzorku. V Blenderu jsou však mapovány pouze plochy, ale vzorky které jsou použity pro výpočet barvy a průhlednosti úseku mraku se nacházejí mezi nimi a proto je potřeba tyto hodnoty interpolovat. To ovšem vede k potřebě mít k dispozici příslušné informace o mapování textury nejen na hranicích mraků, ale také u všech ploch, které se nacházejí uvnitř a rozdělují mrak na úseky – pro každý úsek je potřeba interpolaci provádět zvlášť z koordinátů ploch které jej ohraničují. Renderovací engine Blenderu pro každou plochu počítá pouze ty koordináty, které jsou potřeba pro aplikaci textur použitých v aplikovaném materiálu. Navíc každá textura (slot) může být v případě mapování typu *Object* mapována podle jiného objektu. Proto je nutné nechat pro každou plochu, která se nachází uvnitř mraku nebo kouře, spočítat koordináty dvakrát – jednou pro původní materiál – aby mohl být správně zobrazen a podruhé pro mraky. Mapování plochy rozdělující mrak je ale potřeba upravit následujícím způsobem:

- Jestliže je použito mapování *Glob*, není třeba nic upravovat – koordináty jsou pro všechny objekty stejné.
- V případě, že je textura mraku mapována jako *Orco*, musí být příslušná plocha mapována jako *Object* podle objektu mraku.
- Pokud je textura mraku mapována jako *Object*, musí být plocha mapována také režimem *Object* podle stejného objektu.

Výpočet konečné barvy a průhlednosti segmentu probíhá obdobně jako vrstvení průhledných ploch. Vzorky jsou tedy počítány postupně a připočteny k průběžné hodnotě segmentu.

6.3 Výpočet stínů a osvětlení

Při výpočtu stínů a osvětlení mraků a kouře je potřeba opět vycházet z prostředků které Blender nabízí, aby byl výsledný algoritmus kompatibilní s jeho rendererem a stínem byly ovlivněny také ostatní objekty. Výpočet stínu a osvětlení je v případě mraků velmi podobný. Pokud je potřeba vypočítat nasvětlení u objektu s povrchem, stačí k tomu normála povrchu a pozice světla (v případě difúzního shaderu). Objemová data ovšem žádnou normálu nemají, šlo by ji sice nějakým způsobem získat – např. výpočtem rozdílu průhlednosti, nemusela by však korektně vystihovat světelné podmínky v daném bodě. Proto je potřeba vypočítat přímo útlum světla způsobený průchodem v mraku nebo kouři.

Blender již disponuje prostředky pro simulaci osvětlování průsvitných těles, kterými mraky a kouř jsou – od nedávné verze podporuje tzv. subsurface scattering (SSS). Bohužel je zvolená implementace určena pouze pro plochy. Její princip spočívá ve výpočtu světelných map a hloubky těles z viditelné (natočené ke kameře) a neviditelné (odvrácené) strany objektu [11]. Hodnota z-bufferů a dopadajícího světla potom slouží pro výpočet světla pronikajícího z pod povrchu tělesa. Z toho vyplývá, že pro výpočet osvětlení v mracích a kouři tuto metodu nelze použít, jelikož nejsou definovány svým povrchem, ale procedurální texturou a řešení je potřeba hledat jinde.

Osvětlení je nakonec třeba přece jen počítat stejně jako stíny. Renderovací engine Blenderu nabízí několik způsobů jejich vykreslování – výběr se liší podle typu použitého světla. K dispozici jsou *Ray Shadow* a *Buffer Shadow* (pouze pro typ světla Spot – kuželové světlo). Podstatou Buffer shadow je vytvoření tzv. „shadow buferu“ a jeho následné promítnutí na plochu, kde se stín promítá. V případě mraků by tato metoda sice šla použít pro výpočet stínů, ale pouze těch, které jsou mrakem vrhány – v objemových datech nejsou žádné plochy na které by šel stín promítnout a tak jsou pro výpočet osvětlení nepoužitelné.

Proto je pozornost zaměřena především na *Ray Shadow*, který je počítán pomocí raytraycingu. Ve vývojové verzi Blenderu, do které je algoritmus implementován, jsou kromě klasického výpočtu stínu raytraycingem nově k dispozici také Adaptive QMC (Quasi-Monte Carlo) a Constant QMC algoritmy, které umožňují stín vzorkovat s „téměř“ náhodným rozmístěním vzorků [9]. Díky tomu lze v Blenderu pomocí raytraycingu vykreslovat stíny nebodových světel s rozmazaným okrajem (tzv. Soft shadows). Tato metoda spojuje výhody náhodného a pravidelného rozmístění vzorků – bez šumu (který vzniká u čistě náhodného rozmístění) a bez artefaktů (vznikajících při pravidelném rozmístění) a tak je možné tyto stíny vykreslit poměrně rychle s nízkým počtem vzorků.

Osvětlení je počítáno současně s výpočtem barvy vzorků – přímo ji ovlivňuje. Pro každý zdroj světla ve scéně je potřeba vypočítat jeho útlum v daném bodě – vzorku, jsou jím pak vynásobeny všechny barevné složky (RGB) a tím dojde k celkovému ztmavnutí. Tento útlum určují objekty a úsek mraku které se nacházejí mezi daným bodem a světlem. Pro jeho zjištění světla v objemu média lze použít stejný algoritmus jako při výpočtu barvy mraku – v tomto případě ovšem stačí znát pouze

průhlednost, barva materiálu na množství ani barvu světla nemá vliv. Pro zjištění radiance v daném bodě jsou využity již existující funkce pro výpočet stínu při raytraycingu. Bylo ovšem potřeba je mírně upravit, jelikož původní verze byla schopná počítat stín pouze z hran objektů a jejich textury, nikoliv z jejich objemu.

6.4 Paměťová a časová náročnost algoritmu

Při finálním vykreslování scény renderovacím engine Blenderu je většina spotřebované paměti využita pro pomocné buffery, které se při výpočtu používají. Převážně jde o data která jsou vyžadována v celém průběhu vykreslování scény – paměť snímku, radiosita, některé druhy stínů apod.. Výpočet barvy i osvětlení výsledného vzorku algoritmem však probíhá vždy nezávisle a proto je potřeba uchovávat pouze aktuální hodnotu vzorku, která je postupně aktualizována. Díky tomu došlo integrací algoritmu pouze k zanedbatelnému nárůstu množství paměti potřebné pro vykreslení scény.

Naopak v případě času, potřebného k renderování snímku došlo k poměrně výraznému nárůstu. Důvodem je především velké množství vzorků potřebných k dosažení uspokojivého výsledku – jedná se o trojrozměrná data a proto je jich potřeba několikrát více, než u povrchů. Na rozdíl od běžných shaderů však stačí pro každý vzorek vypočítat pouze barvu textury a průhlednost. Časově náročný je také výpočet stínů a osvětlení – zejména kvůli potřebě ještě většího množství vzorků u nebodových světel (viz předešlá podkapitola). V tomto ohledu je algoritmus částečně optimalizován – výpočet osvětlení probíhá pouze u vzorků s nenulovou průhledností. Také počet vzorků stínu nemusí být tak vysoký jako u povrchů, jelikož je to kompenzováno vysokým počtem vzorků objemu.

Celkově vzato jsou paměťové i časové nároky přijatelné a odpovídají získanému výsledku. Pokud se grafik s tímto algoritmem naučí dobře pracovat, je často možné vhodným modelováním dat čas vykreslování zkrátit – to lze například použitím menšího množství textur, vhodného rozčlenění objektů a dobrým nastavením světel. Čas potřebný k vykreslení objemového tělesa je závislý především na následujících parametrech:

- Plocha, kterou zabírají objekty reprezentující mraky nebo kouř na výsledném renderu. Ne však množství pixelů na kterých je nějaký mrak skutečně vykreslen. Pokud bude zcela průhledný (nepůjde vidět – např. kvůli posunutí textury), bude vykreslovat stejně dlouho, jako normální mrak se stejnými parametry, který vidět půjde. Protože algoritmus „neví“ kde má čekat neprůhledné pixely, bude počítat stejné množství vzorků. Rozdíl nastane pouze pokud budou povoleny stíny, jelikož pro tento případ je algoritmus optimalizován a stíny pro průhledné vzorky již počítat nebude.
- Hloubka těchto objektů z pohledu kamery – ta ovlivňuje množství vzorků ze kterých bude vypočítána barva pixelu.

- Nastavená hustota vzorků. Tou je mimo jiné ovlivněna kvalita výsledku. Dobu vykreslování lze tedy zkrátit snížením detailu.

První dva parametry v podstatě určují velikost objemu který se bude vykreslovat, třetí pak rozlišení, na kterém závisí počet vzorků.

6.5 Známá omezení

Jelikož je zvolený renderer velmi komplexní, bylo by v některých případech velmi problematické algoritmus zcela dokonale integrovat, proto má výsledná implementace některá omezení. Zřejmě by tato omezení šla obejít, znamenalo by to však velmi citelný zásah do rendereru, který by ovlivnil také jeho ostatní součásti a proto od toho bylo upuštěno.

Prvním, poměrně ne příliš významným omezením, je nutnost použití průhledných materiálů pro objekty které zasahují do objemu mraku nebo kouře. Je to způsobeno tím, že renderer vykresluje průhledné a neprůhledné povrchy odděleně a po vykreslení průhledných povrchů jsou všechna související data uvolněna z paměti. Při vykreslování neprůhledného povrchu, který se nachází uvnitř mraku tedy již nejsou k dispozici a proto není možné úplně definovat segment pro vykreslení. Řešením je tedy používat uvnitř mraku pouze průhledné materiály, pokud zvolený objekt nemá být průhledný, je možné nastavit jeho průhlednost na minimum, případně zprůhlednit některé jeho, z pohledu kamery neviditelné, části – tím se bude jevit jako neprůhledný, jeho povrch bude však při vykreslování zařazen mezi průhledné a tak dojde ke správnému vykreslení.

Druhým, poměrně závažnějším omezením, je chyba při vykreslování stínů, která nastává v některých speciálních případech. Bohužel se nepodařilo přesně zjistit co chybu způsobuje, zřejmě se jedná o chybu mapování textury (a tedy získávání průhlednosti objektu) při výpočtu stínů. K jejímu výskytu dochází především při použití vyššího množství textur. Projevuje se tak, že stín se promítne pouze v určité části objektu, zbytek jakoby nevrhal stín. Řešením této situace je zmenšení počtu textur v objektu nebo jeho rozdělení na více částí.

7 Závěr

Cílem této práce byla implementace algoritmu do existující aplikace pro zobrazování scén (rendereru). Nejdříve byly nastudovány a popsány existující algoritmy pro realistické zobrazování objemových dat, především mraků a kouře (kapitola 3). Poté byly vybrány nástroje pro realistické zobrazování scén, které by byly vhodné pro implementaci zobrazovacího algoritmu. Doposud uvedená část byla součástí semestrálního projektu. Na základě poznatků získaných studiem metod pro zobrazování mraků a kouře byl navržen algoritmus který zobrazování implementuje – pro vykreslení objemových dat využívá stochastickou metodu (vzorkování). Jako aplikace, do které je algoritmus implementován, byl zvolen Blender a jeho interní renderovací engine. Následně byl navržen způsob, jakým bude algoritmus do aplikace zakomponován, byl navržen postup modelování dat v editoru a započalo se implementací. S ohledem na komplexnost a provázanost jednotlivých částí vybraného rendereru byl algoritmus integrován přímo do něj. Nakonec byly vytvořeny scény demonstrující schopnosti implementovaného algoritmu a plakátek prezentující výsledky projektu.

Jednou z nevýhod je poměrně velká náročnost použité metody na výpočetní výkon, která počítá osvětlení pro každý vzorek zvlášť. Rychlejší bylo vypočítat osvětlení pro celý objem najednou a následně pouze číst hodnoty. Pro vzorky v jednotlivých směrech paprsků by tak byl výpočet rychlejší, jelikož by bylo možné využít již vypočítané hodnoty z předešlých vzorků, navíc by tato metoda umožnila komplexnější simulaci chování světla (tzv. Multiple scattering – viz podkapitola 3.2). Takovým řešením by však výrazně vzrostly požadavky na paměť a tak by se při výpočtu velkých objemů pravděpodobně stala „úzkým hrdlem“ celého algoritmu a následné swapování systému by vedlo ke konečnému zpomalení.

Přestože vytváření mraků a kouře pomocí tohoto algoritmu grafikovi značně ulehčí práci, pro dosažení kvalitního výsledku je třeba, aby se s ním důkladně naučil pracovat a dokázal získat plnou kontrolu nad výsledkem. Při vytváření mraků je zřejmě nejpodstatnějším parametrem volba vhodných textur a jejich vzájemná kombinace. Pro dosažení uspokojivého výsledku je většinou třeba použít minimálně dvě různé textury – záleží především na typu mraku a jejich struktuře. Většinou je první textura použita pro definování samotných mraků, druhá, překrytá např. režimem násobení, slouží pro nadefinování struktury. V případě kouře je vhodné modelovat buď přímo mesh, který kouř reprezentuje nebo oblast kouře určit pomocí několika textur.

Zřejmě nejnáročnější částí projektu bylo důkladné nastudování a pochopení zdrojových kódů Blender Internal Rendereru, do kterých byl posléze algoritmus implementován. Hlavní překážkou byla absence kvalitní a aktuální dokumentace, způsobená rychlým vývojem a rozsáhlostí aplikace. Původně měla být implementace algoritmu a jeho integrace do aplikace oddělena. S ohledem na způsob zobrazování scény renderovacím engine Blenderu se však autor rozhodl implementovat

algoritmus přímo do něj, jelikož oddělení implementace a integrace by bylo komplikované a neefektivní.

Implementovaný algoritmus poskytuje poměrně kvalitní výsledky, které by bez něj v dané aplikaci nebyly jinými metodami dosažitelné, stále je však prostor pro vylepšení. Kromě algoritmu a jeho optimalizace může být předmětem dalšího vývoje především hlubší integrace a propojení s dalšími součástmi Blenderu. Z uživatelského hlediska by například byla vhodná lepší podpora pro modelování mraků ve scéně s použitím speciálních objektů a interaktivním náhledem. Pro animaci mraků by bylo možné implementovat algoritmy schopné realisticky zachytit jejich chování, například simulovat tepelné proudění apod. – současné řešení umožňuje v čase pouze transformovat texturovací koordináty. Užitečný by byl také nástroj umožňující snadno modelovat prostorová data – použití textury se hodí spíše u mraků, naopak u kouře je často požadován specifický a složitější tvar a pro tento účel není reprezentace objemových dat texturou příliš vhodná.

Literatura

[1] Eva Cerezo, Frederic Pérez, Xavier Pueyo, Francisco J. Seron, Francois X. Sillion: A Survey on Participating Media Rendering Techniques [online], <http://artis.imag.fr/Publications/2005/CPPSS05/>, 30. 12. 2007.

[2] skupina autorů: Beer-Lambert law, Wikipedia, The Free Encyclopedia [online], <http://en.wikipedia.org/wiki/Beer-Lambert_law>, 30. 12. 2007.

[3] Pat Hanrahan: Participating Media and Volume Rendering, Image Synthesis Techniques [online], <http://graphics.stanford.edu/courses/cs348b-02/lectures/lecture13/>, 30. 12. 2007.

[4] Nave, R.: HyperPhysics [online], <http://hyperphysics.phy-astr.gsu.edu/hbase/atmos/blusky.html#c3>, 30. 12. 2007.

[5] Ryo Miyazaki, Yoshinori Dobashi, Tomoyuki Nishita: A Fast Rendering Method of Clouds Using Shadow-View Slices, University of Tokyo, Hokkaido University

[6] Sundstedt, V., Gutierrez, D., Anson, O., Banterle, F., Chalmers, A.: Perceptual Rendering of Participating Media, University of Zaragoza & University of Bristol

[7] Pauly, M., Kollig, T., Keller, A.: Metropolis Light Transport for Participating Media, ETH Zurich University of Kaiserslautern [online], <http://graphics.uni-ulm.de/MediaMLT.pdf>, 30. 12. 2007.

[8] Skupina autorů: Blender – Developement [online], Blender.org, <http://www.blender.org/development/>, 10. 5. 2008

[9] Skupina autorů: QMC Sampling Types [online], Blender.org <http://www.blender.org/development/current-projects/changes-since-245/qmc-sampling/>, 10. 5. 2008

[10] Skupina autorů: Blender – Features, Blender.org [online], <http://www.blender.org/features-gallery/features/>, 10. 5. 2008

[11] Skupina autorů: Blender – Subsurface scattering [online], Blender.org, <http://wiki.blender.org/index.php/Manual/Subsurface_Scattering>, 10. 5. 2008

Seznam příloh

Příloha 1: Slovníček pojmů

pojem	význam
akcelerátor, grafický	rozšiřující prvek grafické karty umožňující vykreslování 3D objektů
ambient occlusion	metoda pro výpočet osvětlení povrchů – výpočet probíhá v rámci celé scény s ohledem na ostatní objekty
billboard	polygon, který je natočen vždy ke kameře
Blender	nástroj pro 3D editaci, animaci, vykreslování a tvorbu interaktivních animací
blob	prostorový objekt definován polohou a intenzitou svého „pole“
CAD	Computer Aided Design – počítačem podporované modelování (nachází uplatnění především ve výrobním průmyslu)
CPU	procesor určený pro univerzální výpočty
computer graphics, CG	počítačová grafika
částicový systém	skupina částic(bodů), tyto body bývají vykreslovány jako polygony s texturou natočené vždy ke kameře – využívá se pro zobrazování objemových dat
global illumination	metoda osvětlování scény jejím okolím (např. oblohou v podobě textury texturou)
GPU	grafický procesor - specializovaný na práci s vektory
kompozitor	grafikcký nástroj sloužící ke zpracování scény po jejím vykreslení
koordináty, texturovací	souřadnice v n-rozměrném prostoru určující polohu bodu v textuře
materiál (CG)	skupina parametrů ovlivňující vlastnosti povrchu nebo objemu grafického objektu
mesh	trojrozměrné těleso složené z polygonů (označován také jako model)
node editor	nástroj umožňující interaktivní kombinování několika vstupů (např. materiálů, vrstev apod.) používaný pro vytváření materiálu nebo postprocessing scény
objekt (CG)	prvek popisující objekt reálného světa ve virtuální scéně – určuje jeho polohu, použitý mesh (tvar), materiál apod.
objemová data	množina hodnot popisující vlastnosti nějaké veličiny v prostoru (např. barvu, průhlednost apod.)
OSA, oversampling	nadvzorkování pixelů při jejich vykreslování – zabrání nežádoucím „zubatým“ hranám
polygon	plocha v prostoru definována několika body (nejčasteji třemi – trojúhelník nebo čtyřmi - quad)

pojem	význam
postprocessing	skupina úprav po vyrenderování scény, která vede k finální podobě výsledku
quad	plocha v prostoru určená čtyřmi body (často dochází při vykreslování k převodu na trojúhelníky)
render	výstup rendereru
renderer, renderovací engine	nástroj sloužící ke konečnému vykreslení scény nebo animace, umožňuje výpočet stínů, osvětlení, odrazů apod.
scéna (CG)	skupina virtuálních objektů a dalších grafických prvků (např. světel) modelující část reálného nebo smyšleného světa
shader	program definující výpočet barvy povrchu
sprite	polygon natočený vždy ke kameře
SSS, subsurface scattering	modelování chování světla uvnitř objektu – používá se při zobrazování mraků, kouře, ale také vosku a organických materiálů (kůže, ovoce)
textura (CG)	grafická data sloužící k definování barev a dalších vlastností povrchu nebo objemu objektu (např. obrázek)
textura, procedurální	textura definovaná pomocí algoritmu (např. šum, šachovnice, barevný přechod apod.)
volumetrická data	viz objemová data

Příloha 2: Obrázky demonstračních scén a porovnání časové náročnosti

Všechny obrázky v této příloze byly vyrenderovány v Blender Internal Rednereru a za pomoci implementovaného algoritmu. Uvedené časy byly naměřeny na stolním počítači s dvoujádrovým CPU (frekvence 1.86 GHz), 2GB RAM a operačním systémem Linux. Obrázky byly vykresleny v rozlišení 640×480 bez použití OSA. Demonstrační scény jsou uloženy na přiloženém CD v adresáři demos.

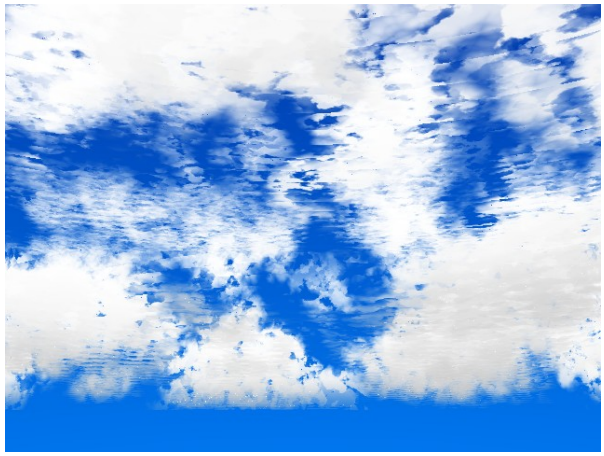
Poznámka: RT (Render Time) udává dobu vykreslování, čas je udáván ve formátu MM:SS.

Jednoduché mraky – *demo_clouds.blend* **ukázka vlivu parametru hustoty vzorkování**

Struktura mraků je definována pomocí tří textur:

- rozložení mraků – definuje jednotlivé mraky
- struktura mraků – zajišťuje „roztřepené“ okraje
- tvar mraků – dole jsou mraky nejhustší, směrem nahoru řidší

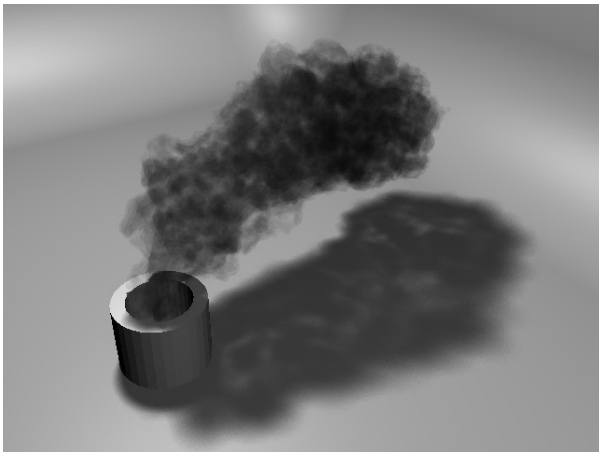
Parametry: SampleStep 1.0 (maximum), RT 14:11
Komentář: Vysoký počet vzorků sice poskytl pěkný výsledek, ale za cenu nepřijatelné doby vykreslování



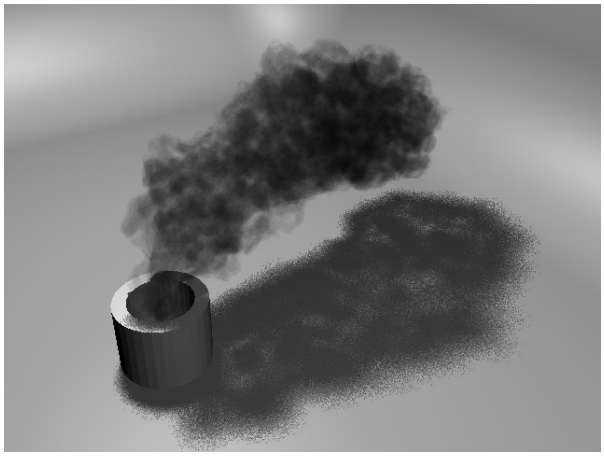
Parametry: SampleStep 0.01, RT 0:09
Komentář: Malý počet vzorků způsobil na první pohled viditelné artefakty (pruhy)

Parametry: SampleStep 0.1, RT 1:24
Komentář: Vhodným nastavením počtu vzorků se podařilo dosáhnout dobrý výsledek v přijatelném čase

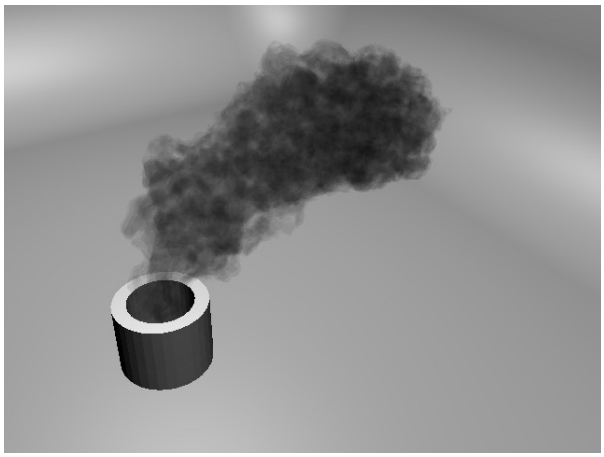
Ukázky kouře – *demo_smoke.blend*



Parametry: stín (8 vzorků), RT 5:58
Komentář: Ukázka stínu vrhaného kouřem.

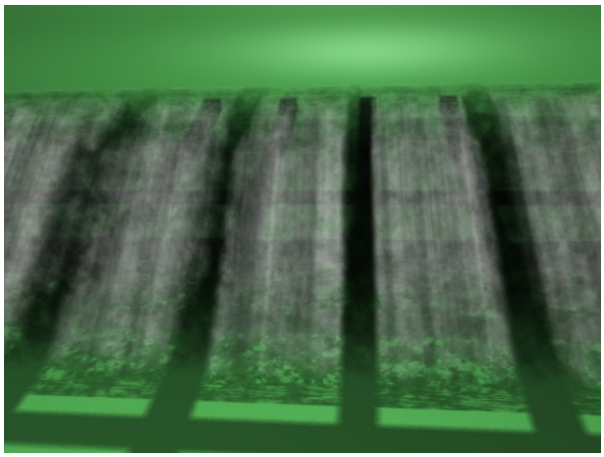


Parametry: stín (2 vzorky), RT 0:51
Komentář: Při použití menšího počtu vzorků je stín zašumělý



Parametry: bez stínu, RT 0:03
Komentář: Bez vykreslování stínu je výsledný render podstatně rychlejší

Ukázka zobrazení stínu dopadajícího na mrak nebo kouř



Parametry: stín o osmi vzorcích, RT 23:41
Komentář: Takto lze snadno zobrazit například také rozvířený prach.

Příloha 3: Postup sestavení a ovládání

Sestavení

Před tím než je možné s upraveným Blenderem pracovat, je nutné jej sestavit. Jak bylo řečeno, je Blender multiplatformní aplikace a lze jej sestavit na všech nejrozšířenějších platformách. Blender primárně využívá sestavovací nástroj SCons (http://www.scons.org/). Po instalaci tohoto nástroje a závislostí Blenderu lze sestavení provést zadáním příkazu `scons` v adresáři `blender`, ve kterém se nacházejí zdrojové kódy Blenderu:

```
cd blender
scons
```

Blender má řadu závislostí a proto někdy bývá sestavení problematické – především na platformě Windows, jelikož na POSIXových jsou již knihovny nainstalovány. Některé funkce Blenderu jdou při jeho kompilaci vypnout a tím tak snížit počet závislostí. Podrobný seznam těchto možností se vypíše po zadání

```
scons --help
```

Po sestavení se Blender nainstaluje do instalačního adresáře, implicitně `../install` – relativně ke kořenovému adresáři `blender` se zdrojovými kódy. Nyní je ho možné spustit:

```
cd ..
cd install/linux2/
./blender
```

Samotný algoritmus je implementován v jednom souboru:

```
source/blender/render/intern/source/clouds.c
```

Ovšem pro jeho integraci byly upraveny také některé další soubory.

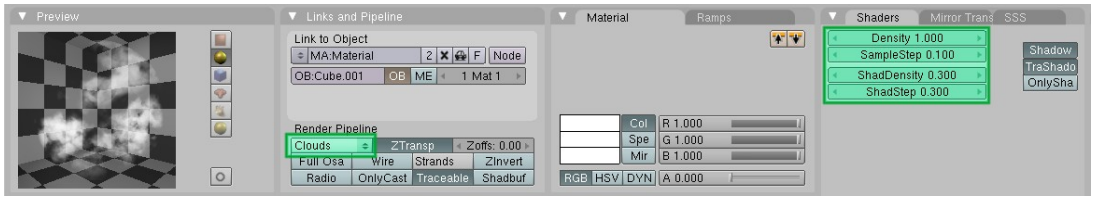
Ovládání algoritmu z prostředí Blenderu

Pro vyrenderování jednoduché scény s mraky je potřeba uskutečnit několik kroků:

1. Spustit Blender, ve kterém je implementován příslušný algoritmus.
2. Vložit do scény:
   - kameru
   - objekt který bude definovat prostor kde se mraky nacházejí
   - světlo
3. (volitelně) Nastavit pozadí scény (např. barevný přechod).
4. Přiřadit objektu mraků nový materiál s typem *Clouds* a alfa-průhledností (*ZTransp*) a nastavit požadované parametry:

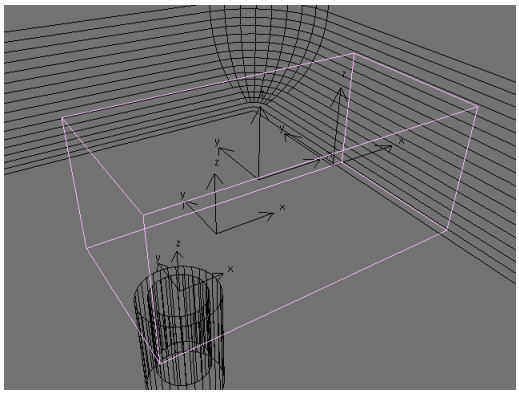
   Density – hustota kouře nebo mraku
   SampStep – vzorkovací krok (v Blender jednotkách)
   ShadDensity – intenzita stínu
   ShadStep – vzorkovací krok pro stín (v Blender jednotkách)

   Ostatní nastavení mají stejný význam jako u běžného materiálu

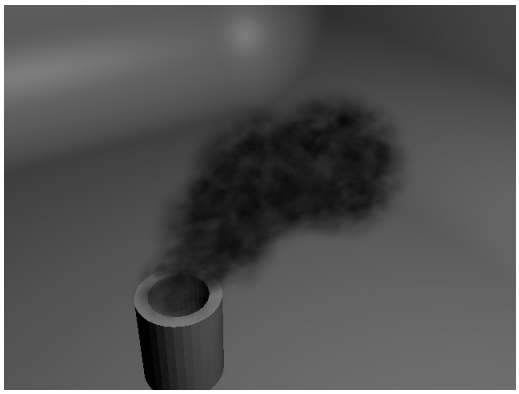


5. Pomocí procedurálních textur nadefinovat strukturu mraku (důležité je především namapovat alfa-kanál). Práce s texturami je stejná jako v případě těles s povrchem, ovšem textury budou vykresleny trojrozměrně. Mapování jiná než *Orco*, *Glob* a *Object* jsou ignorována.
6. Vyrendrovat.

Pro definování komplexnějších struktur lze použít mapování vhodných textur (např. Blend) na jiné objekty (např. typu Empty). Příklad – scéna *demo_smoke-controll* – pro dosažení požadovaného tvaru jsou použity 4 textury typu *Blend (Sphere)* mapované podle 4 různých objektů, struktura je určená pátou texturou.



Ukázka návrhu tvaru v editoru



Tvar kouře po vyrenderování