UNIVERZITA KOMENSKÉHO, BRATISLAVA

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Generovanie a Animovanie Oblohy

2011

Jakub Dobšovič

UNIVERZITA KOMENSKÉHO, BRATISLAVA

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Generovanie a Animovanie Oblohy

Bakalárska práca

Študijný program : Informatika

Študijný odbor: 9.2.1 Informatika

Školiace pracovisko: Katedra Aplikovanej Informatiky

Školiteľ: Martin, Samuelčík, RNDr.

Konzultant: Martin, Samuelčík, RNDr.

Evidenčné číslo: 36e8a16b-f40c-415b-b276-4c8cfeec1eb6

Bratislava, 2011

Jakub Dobšovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jakub Dobšovič
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Generovanie a zobrazenie animovanej oblohy

Cieľ: Študent preskúma rôzne spôsoby zobrazenie animovanej pomocou grafickej knižnice OpenGL a vytvorí aplikáciu, ktorá implementuje dané postupy. Súčasťou práce bude aj generovanie animovaných textúr obsahujúcich pohyb oblakov na oblohe.

Vedúci: Mgr. Martin Samuelčík, PhD.

Dátum zadania: 04.10.2010

Dátum schválenia: 22.10.2010

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci

Vyhlasujem, že som prácu vypracoval samostatne, s použitím uvedenej literatúry a pod odborným vedením RNDr. Martina Samuelčíka.

Chcel by som poďakovať svojmu vedúcemu RNDr. Martinovi Samuelčíkovi za jeho pomoc, rady a usmerňovanie.

Abstract

The work discusses computer methods of generating and displaying sky and it's animation. This work consists of two parts: theoretical and practical. Theoretical discusses selected methods of generating sky for 3D graphic environment, analyzes their advantages and disadvantages and means of animating results. Practical part implements one of the methods with use of OpenGL graphic library. Result of practical part is transferable, reusable, animated texture.

KEYWORDS: animated sky, skybox, generating clouds

Abstrakt

Práca sa venuje počítačovým metódam generovania a zobrazovania oblohy a jej animácie. Táto práca pozostáva z dvoch častí, praktickej a písomnej. Písomná časť pojednáva o vybraných metódach generovania oblohy pre 3D grafické prostredie, rozoberá ich výhody a nevýhody a spôsoby, ako výslednú oblohu animovať. Praktická časť implementuje jednu z metód pomocou grafickej knižnice OpenGL. Výsledkom praktickej časti je prenositeľná, znovupoužiteľná animovaná textúra oblohy.

KĽÚČOVÉ SLOVÁ: animovaná obloha, skybox, generovanie oblakov

Obsah

Úvod

Obloha je nezastúpiteľným prvkom exteriérových scén. Zobrazovanie realistickej oblohy je jedným z kľúčových prvkov virtuálnych prostredí a bez nej tieto prostredia strácajú na uveriteľnosti. Preto už istú dobu pozorujeme v počítačovej grafike snahu verne zobrazovať oblačnosť. Získavanie dát zo skutočnej oblohy je časovo aj technologicky náročné. S technologickým pokrokom sa však ponúka reálna možnosť oblaky počítačovo generovať.

Hlavným problémom tejto témy je tvar. Napriek tomu, že tvar oblakov sa nedá matematicky definovať, existuje niekoľko spôsobov ako ho odhadnúť. V tejto práci predvedieme, že aj keď nie sú fyzikálne korektné, ich výsledky sú uveriteľné. Druhým problémom je osvetlenie oblakov. Svetlo, prechádzajúce oblakom sa rozptyľuje. Vodné častice, z ktorých je oblak zložený časť svetla pohltia a zvyšok vyžiaria. Toto vyžarovanie nie je lineárne a navyše ovplyvňuje ostatné častice, preto sa pre potreby počítačovej grafiky nedá fyzikálne simulovať. Vizuálne napodobnenie tohto efektu je jednou z tém tejto práce.

V teoretickej časti načrtávame niekoľko metód generovania oblohy, ktoré boli úspešne použité, a rozoberáme v nich použité postupy. Ide najmä o metódy založené na perlinovom šume. Elias[1] a Quílez[2] používajú šum priamo, ich metódy spočívajú vo vygenerovaní "cloud map" - mapy oblakov a jej následnom nasvietení. Ďalšia metóda tiež využíva perlinov šum, no len ako mapu hustoty, podľa ktorej vygeneruje 3D oblačnosť. Túto metódu opísal Man[3].

V praktickej časti navrhujeme vlastnú metódu, založenú na fraktáloch a sebepodobnej štruktúre oblakov. Táto metóda na základe parametrov oblohy v danom čase pseudonáhodne vytvorí základy oblakov. Na tieto základy možno nahliadať ako na semená, z ktorých potom na základe fraktálov vznikne celý oblak. Táto metóda kladie dôraz na determiničnosť a spojitosť. Pre dané parametre vznikne daná obloha a pre podobné parametre vznikne podobná obloha. Animácia sa potom zjednoduší na lineárnu interpoláciu medzi parametrami v dvoch rôznych časoch.

1. Počítačová obloha

1.1 Obloha

Obloha je časť atmosféry viditeľná voľným okom zo zeme. Ako taká je významnou súčasťou exteriéru a teda aj jeho počítačovej vizualizácie. Skladá sa najmä z molekúl dusíka, kyslíka a z vody vo forme pary a ľadových kryštálov v útvaroch oblačnosti. Farba oblohy je spôsobená tým, ako slnečné lúče prechádzajú týmito zložkami. Počas dňa svetlo prechádza tenkou vrstvou atmosféry. Plyny v atmosfére pohltia časť svetla s malou vlnovou dĺžkou - najmä modré svetlo. Zvyšok svetla sa dostane na povrch, preto sa slnko javí bielo-žlté. Pohltené svetlo je následne vyžiarené všetkými smermi a preto sa obloha javí modrá. Pri západe slnka svetlo dopadá šikmo a prechádza väčšou vrstvou atmosféry. Plyny pohltia oveľa viac svetla dlhších vlnových dĺžok a menej sa dostane na povrch zeme. Preto sa slnko pri západe javí tmavšie, oranžové až červené a obloha naberá odtiene od červenej tesne nad horizontom až po fialovo-modrú na východe, ako to vidno na Obrázku 1.

Obrázok 1. Západ slnka.

Viac informácií o atmosfére môžno nájsť na [Sms97].

Voda v oblakoch pohltí a následne vyžiari oveľa viac svetla ako vzduch, preto sú oblaky biele. Pri prechode je svetlo pohltené a vyžiarené mnoho krát, zakaždým sa časť rozptýli do okolia častice. Takto vzniká napríklad takzvaný strieborný okraj oblakov, viditeľný na Obrázku 2; jeden z efektov, ktorý niektoré počítačové oblohy napodobňujú.

Obrázok 2. Strieborný oblak.

Oblaky sa vzhľadom na typ a miesto výskytu väčšinou nachádzajú vo výškach od 2 000 až do 18 000 metrov nad povrchom. [wik11] Tento fakt sa využíva pri renderovaní. Pokiaľ je pozorovateľ na zemi, možno zanedbať perspektívu, prípadne generovať najvyššiu vrstvu len v 2D.

V tejto práci pojednávame o troch základných pojmoch: generovanie, zobrazovanie a animácia oblohy. Pri generovaní, alebo vytváraní, sa zaoberáme rozmiestnením a štruktúrou oblakov, vstupom, použitými dátovými štruktúrami a podobne. Pri renderovaní, alebo zobrazovaní, hovoríme o spôsobe ako zobraziť vygenerované dáta, spôsobe ukladania výsledkov, či o osvetlení oblakov. Animácia zahrňuje premenlivosť oblakov, ich pohyb po oblohe, dynamiku atmosféry, pohyb slnka a podobne.

1.2 OpenGL

"OpenGL, teda Open Graphics Library, je interface ku grafickému hardvéru. Pozostáva z asi 120 príkazov, ktoré sa používajú na vytvorenie interaktívnych, trojrozmerných aplikácií." [WNDS99]. Vytvorila ho spoločnosť Silicon Graphics Inc. v roku 1992. V súčasnosti ho ako open- source projekt vyvíja nezisková organizácia Khronos Group. OpenGL je nezávislé od operačného systému, aj od platformy. Definuje štandard, ktorý implementuje až výrobca grafických kariet. OpenGL sa používa pri CAD, virtuálnej realite, vizualizácií informácií, v hrách a letových simulátoroch. Je populárne aj vďaka vysokej kvalite oficiálnej dokumentácie. Skvelým zdrojom informácií a učebnou pomôckou je napríklad OpenGL Programming Guide, alebo takzvaná Red Book ('červená kniha' - podľa červeného obalu na všetkých

vydaniach) [WNDS99].

OpenGL je virtuálny stroj riadený stavmi. Tieto stavy sa dajú meniť za behu aplikácie a zostávajú v platnosti až do ďalšej zmeny.

V našej implementácii používame štandardnú knižnicu GLUT [Gro11], ktorá zabezpečuje inicializáciu OpenGL a okna a takisto vykreslenie renderovacieho buffera. GLUT je closed-source knižnica, ktorá implementuje jednoduché oknové API pre OpenGL. GLUT je nezávislá od operačného systému a je použiteľná s takmer každou implementáciou OpenGL. Ďalej využívame GLU [Gro11] - knižnicu zabezpečujúcu zložitejšie funkcie tvorené viacerými jednoduchými volaniami základného OpenGL. Využívame napríklad volanie gluPerspective, ktoré nastaví projekčnú maticu, či gluLookAt, ktoré nastaví pozíciu, smer a otočenie kamery. GLU sa zvyčajne dodáva so základným OpenGL balíkom. Obe tieto knižnice slúžia hlavne na uľahčenie a zrýchlenie práce s OpenGL. Ďalej z OpenGL využívame priehľadnosť, konkrétne priehľadné textúry. OpenGL implementuje alfa zmiešavanie pomocou rôznych zmiešavacích funkcií. My používame zmiešavaciu funkciu glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA). To znamená, že farba, ktorá sa vykreslí na frame buffer sa vypočíta pomocou vzorca 1:

$$cOut = cSrc*aSrc + (1-aSrc)*cDst$$

Vzorec 1. [Gro99].

cSrc a aSrc sú farba respektíve alfa fragmentu, ktorý je treba zapísať, cDst je farba destinácie, teda fragmentu, ktorý je aktuálne zapísaný v frame buffery.

1.3 Off Screen renderovanie

OpenGL sa zvyčajne používa na renderovanie do okna, ktoré sa zobrazí na obrazovke, no niekedy môže byť výhodné renderovať do buffra, ktorý sa nezobrazí. To sa používa napríklad pri generovaní dočasných obrázkov - napríklad textúr, hromadnom renderovaní neinteraktívnych animácií - ako v našom prípade, alebo pri generovaní obrázkov s vysokým rozlíšením. Dobrým zdrojom informácií o off screen

renderingu je [Bri97]. Off screen renderovanie nie je základnou súčasťou OpenGL, zabezpečuje ho interface operačného systému, pod ktorým je OpenGL spustené, prípadne externé knižnice. Jedným zo spôsobov je využiť renderovanie do MS Windows device-independent bitmap. Treba vytvoriť bitmapu pomocou CreateDIBSection. Je potrebné zvoliť formát s parametrami PFD_DRAW_TO_BITMAP, PFD_SUPPORT_OPENGL a PFD_SUPPORT_GDI. Vytvorí sa WGL contex a pripojí sa k OpenGL, potom už môže nasledovať samotné renderovanie.

Výhody:

* Štandardná funkcia WGL

Nevýhody:

* Použiteľné len s Windows 95/NT OpenGL

Ďalšou možnosťou sú GLX Pixmapy. GLX pixmapa je X Pixmapa obohatená o niekoľko prídavných bufferov. Na vytvorenie GLX pixmapy treba otvoriť 'X display' spojenie, vybrať 'X visual', vytvoriť X pixmapu a z nej GLX pixmapu. OpenGL renderovací context sa pripojí k GLX pixmape a môže sa renderovať.

Výhody:

* Obsahy bufferov sú zachované
* GLX Pixmapy sú štandardnou súčasťou GLX
* GLX pixmapy môžu byť väčšie ako okno na obrazovke

Nevýhody:

* Renderovanie do GLX pixmáp nemusí byť akcelerované hardware-om a teda môže byť pomalé
* Veľkosť ja obmedzená veľkosťou obrazovky

* Pre UNIX systémy, vyžaduje spojenie s X serverom

Pbuffre: Pixel buffre, alebo pbuffre, sú rozšírením OpenGL. Sú to neviditeľné, prídavné renderovacie buffre, do ktorých môže OpenGL renderovať. Ich cieľom je hardware-ovo akcelerované off screen renderovanie, s možnosťou formátov zvyčajne nedostupných cez X display.

Výhody:

* Hardwareová akcelerácia

* Ponúka formáty, ktoré iné metódy neponúkajú

Nevýhody:

* Alokácia pbuffra môže zlyhať pre nedostatok zdrojov

* Obsah pbuffera sa niekedy môže vymazať, ak sa zmení zobrazovací mód

* Obtiažnejšie použitie než pri GLX pixmapách

* Veľkosť ja obmedzená veľkosťou obrazovky

FBO (framebuffer objekt). GL_EXT_framebuffer_object je ďalším rozšírením OpenGL, od verzie OpenGL 3.0 sa stalo štandardnou súčasťou. "Poskytuje interface pre vytváranie nezobrazovaných framebuffer objektov. Tento framebuffer sa označuje ako aplikáciou vytvorený framebuffer, aby sa odlíšil od štandardného framebuffra, ktorý poskytne oknový systém." [Ahn08]. Vďaka tomuto rozšíreniu OpenGL môže presmerovať renderovanie do framebuffer objektu, miesto do štandardného framebuffera. Framebuffer objekt obsahuje niekoľko logických bufferov, nazývaných 'framebuffer-attachable images'. Sú to 2D polia pixelov, ktoré môžu byť pripojené k tomuto framebuffer objektu.

Výhody:

* Rýchle prepínanie medzi jednotlivými logickými buffermi

* Flexibilné a pomerne jednoduché použitie

Nevýhody:

* Nie každý hardware podporuje FBO, software-ová implementácia môže byť pomalá

glReadPixels vráti dáta z frame buffera. Táto funkcia má niekoľko argumentov - x, y je počiatočná pozícia prečítaného obdĺžnika. Šírka a výška určujú veľkosť tohto obdĺžnika. Ďalej sa zadáva formát pixelov a dátový typ výsledku a nakoniec odkaz na štruktúru, do ktorej sa dáta uložia. Výhody:

* Obsah buffera ostáva nezmenený

Nevýhody:

* Môže byť pomalé

1.4 Python a PyOpenGL

Python je interpretačný, objektovo orientovaný, high-level programovací jazyk. Bol vytvorený dánskym programátorom Guidom van Rossumom, jeho prvá verzia vyšla v roku 1991. Projekt je - a vždy bol - open-source a je distribuovaný pod veľmi benevolentnou licenciou. Python je jednoduchý na používanie aj na učenie. Má jednu z najjednoduchších a najprehľadnejších syntaxí; obsahuje veľa vyšších dátových štruktúr a metódy na prácu s nimi. Preto je Python pokladaný za silný a jednoduchý jazyk a programovanie v ňom je pomerne rýchle. Informácie o projekte poskytuje stránka [Fou11]. To je hlavný dôvod prečo sme sa ho rozhodli použiť. Zaujímavosťou je, že typický kód v jazyku Python je až niekoľko násobne kratší, než kód rovnakého programu v C++.

"PyOpenGL je platformovo nezávislé prepojenie Pythonu s OpenGL a niektorými príbuznými API." [pro11] Prepojenie je zabezpečené pomocou ctypes knižnice

(štandard od verzie Python 2.5). PyOpenGL podporuje OpenGL od verzie 1.1 po 3.2, GLU, GLUT a GLE 3. Takisto podporuje stovky rozšírení pre OpenGL. Spolupracuje s mnohými knižnicami pre Python, ktoré zabezpečujú grafické rozhranie, no tiež sa dá použiť GLUT. PyOpenGL dodržiava konvencie pre pomenovanie funkcií OpenGL, čiže obsahuje rovnaké funkcie a volania. Argumentami sú zvyčajne Pythonovske ekvivalenty typov jazyka C.

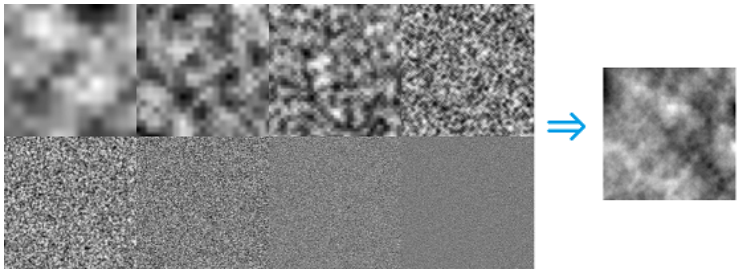
1.5 Vytváranie video súborov

Video súbory pozostávajú zo sady obrázkov, prípadne audia a pomocných údajov (metadata). Tieto dáta sú v analógovej, alebo surovej digitálnej podobe príliš veľké na šírenie. Tvorba videa teda spočíva v kompresii údajov, a ich uložení do jednotného súboru. Komprimuje sa zvlášť video a zvlášť audio, táto kompresia sa deje pomocou takzvaných kodekov. Kodek, alebo compressor/decompressor, je algoritmus ktorý komprimuje dáta na uloženie, a dekomprimuje na prehrávanie. Komprimácia môže byť stratová alebo bezstratová, môže mať rôznu kvalitu. Jej úspešnosť sa väčšinou vyjadruje ako bit/sec, teda objem dát ktorý je potrebný na prehranie sekundy obsahu. Zvyčajne platí, že väčšia kvalita znamená viac dát za sekundu a teda aj veľkosť výsledného súboru. Príklad video kodekov: H.264, MPEG-4, MPEG-2; audio: AAC alebo MP3. Nasleduje uloženie všetkých typov informácií do jedného súboru, akéhosi kontajnera. Tento proces sa nazýva aj enkapsulácia alebo zapúzdrenie a výsledkom je kompozitný súbor v niektorom z video formátov, napríklad .mov, .mpeg alebo .avi. Úvod do tejto problematiky poskytuje napríklad [Rob11]

2. Generovanie a renderovanie oblohy

2.1 Ken Perlin

Dôležitým prvkom nielen počítačovej grafiky je pseudo náhodnosť, a jej pravdepodobne najznámejší zdroj je perlinov šum. Preto by bolo dobré spomenúť na tomto mieste jeho tvorcu Kena Perlina. "Ken Perlin, Ph.D. je profesorom Oddelenia Počítačových vied Univerzity New Yorku, riadi Games For Learning Institute Univerzity New Yorku. Získal ocenenie 'Academy Award for Technical Achievement' za svoju prácu na šumových a turbulentných procedurálnych textúrach." [Per11] Šum je textúrovacia primitíva, kombinovaním šumu do rôznych matematických výrazov vznikajú procedurálne textúry. Nevyžadujú žiadny zdrojový obrázok a teda ani takmer žiadnu komunikáciu pri ukladaní, alebo prenášaní. Takéto textúry dosahujú sebe podobnosť zložením niekoľkých oktáv šumu a spojitosť vhodne zvolenou funkciou generujúcou samotný šum, ako je to znázornené na Obrázku 3.



Obrázok 3. Osem oktáv perlinovho šumu a výsledné zloženie [Cop08].

Dobrá implementácia dokonca zabezpečí, že výsledná textúra sa dá ukladať do mriežky - ako kachličky - bez viditeľného spoja. Ken Perlin sa začal vážne zaoberať procedurálnymi textúrami, keď spolupracoval na animovanom filme Tron, v roku 1981 - [Per99]. Pôvodným zámerom bolo oživenie sterilných textúr, pridanie prirodzeného, náhodne vyzerajúceho faktoru. Dnes sa však perlinov šum používa v mnohých aplikáciach 2D aj 3D grafiky, od textúr, cez modelovanie, generovanie terénov až po tvorbu efektov ako oheň, dym, či oblaky.

2.2 Obloha ako kulisa alebo prostredie

Obloha vo virtuálnych prostrediach má dva hlavné spôsoby použitia: ako vzdialená statická kulisa a ako dynamické prostredie. Od toho sa odvíja aj renderovanie. Vzdialená kulisa je vhodná najmä pre virtuálne prechádzky mestom, virtuálne múzeá a počítačové hry, kde je hráč neustále na zemi. Je to preto, že takýto spôsob renderovania oblohy ťaží zo vzdialenosti pozorovateľa k oblohe. Vzdialenosť je taká veľká, že aj pri veľkom posunutí pozorovateľa sa vzájomná pozícia jednotlivých oblakov takmer nezmení. Potom možno zanedbať perspektívu a oblaky zobraziť premietnuté do jednej roviny. Tento prístup sa väčšinou realizuje takzvaným skyboxom alebo skydomom. Skybox je útvar v tvare kocky, s piatimi alebo šiestimi stenami, otočenými "dovnútra". V strede tejto kocky sa nachádza pozorovateľ a na stenách sa zobrazuje obloha ako textúra. Výhodou je jednoduché zobrazenie, keďže stačí zobrazovať päť otextúrovaných plôch a takisto jednoduchá animácia. Nevýhodou je, že vzdialenosť k stredu steny kocky je menšia ako vzdialenosť k jej rohu. Vzdialenosť k ideálne premietnutým oblakom by mala byť všade rovnaká a preto, v závislosti od spôsobu generovania, je treba tento rozdiel kompenzovať. Skydome je polguľa alebo guľa v ktorej strede je pozorovateľ. Podobne ako pri skyboxe sa na jej vnútorné steny premieta textúra oblohy. Vzhľadom na tvar skydomu, je vzdialenosť pozorovateľa k oblohe všade rovnaká, no skybox je náročnejší na vyhotovenie aj zobrazenie. Oba spôsoby umožňujú zobraziť viacero nezávislých vrstiev oblakov, v rôznych výškach.

Pri leteckých simuláciách a ďalších hrách s lietaním je situácia zložitejšia. Očakáva sa od nich, že objekty budú prelietavať pred aj za oblakmi, prípadne skrz ne. Oblaky musia byť trojrozmerné a priesvitné, aby sa dosiahla realistickosť.

2.3 2D obloha zo šumu

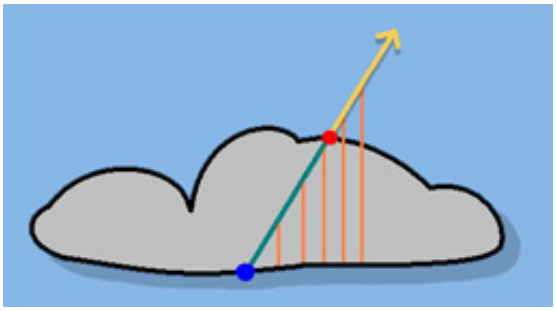
Jednoduchou a rýchlou možnosťou generovania oblohy je vytvoriť ju z nejakého pseudonáhodného šumu. Dobrým príkladom je Perlinov šum, keďže sám osebe pripomína prirodzenú oblačnosť. Aplikovaním jednoduchých úprav na textúru takéhoto šumu sa

zvýši jej reálnosť. Takisto sa získajú určité možnosti ako nastaviť vzhľad výsledných oblakov.

Príkladom použitia šumu je implementácia Huga Eliasa. Elias miesto Perlinovho šumu použil zloženie 4 oktáv jednoduchšieho šedo-tónového šumu, pre dosiahnutie väčšej rýchlosti. Na hodnoty výslednej textúry aplikoval exponenciálnu funkciu s prahom. Okrem zlepšenia vernosti oblohy tak vytvoril 2 premenné na ovplyvnenie oblakov: 'CloudCover' a 'CloudSharpness'. "Hodnoty CloudCover medzi 0 a 255 udávajú úplné pokrytie oblakmi, respektíve čistú oblohu. CloudSharpness kontroluje ako rozmazané alebo ostré sú oblaky." [Eli11]. Takto upravenú textúru aplikoval na skydome a vyrenderoval mapu oblačnosti, ktorú ešte treba zafarbiť. Pre každý pixel renderovaného obrazu sa vypočíta jeho hodnota na základe mapy oblačnosti, modrej farby oblohy, slnečnej žiary, šedej hmly v diaľke a osvetlenej bump mapy oblakov. Nakoniec sa na vypočítanú hodnotu aplikuje úprava expozície. Elias implementoval aj animáciu takto generovanej oblohy, znázornené na Obrázku 4. "Na animovanie oblohy stačí animovať pôvodné mapy šumu. ... Ich animovanie jednoducho spočíva v interpolácii od jednej mapy šumu k druhej." [Eli11].

Obrázok 4. Prechod na polceste od prvej mapy k druhej [Eli11].

Doplníme, že ďalšou možnosťou animácie je použitie 4 rozmerného perlinovho šumu, kde štvrtý rozmer slúži ako čas animácie. Podobne postupoval aj Quílez, no s iným riešením osvetlenia. Implementoval raycasting algoritmus - lúče prechádzajúce akoby 3D oblakom, reprezentovaným 2D mapou - textúrou šumu. "Myšlienka je pokladať oblak za hustejší tam, kde je hodnota mapy vyššia. Ak predpokladáme viac menej plochú základňu oblaku, hustejší pre nás znamená vyšší." [Qui05]. Keďže sa na oblohu pozerá zospodu, počítal Quílez farbu pre základňu oblaku. Čím dlhšie prechádzal slnečný lúč k pixelu základne, tým tmavšiu farbu pixel mal (znázornené na Obrázku 5)



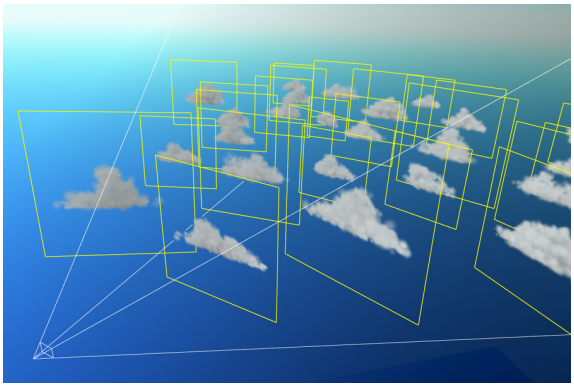
Obrázok 5. Prechod lúča oblakom [Qui05].

Pri správnej implementácii sa dajú obe metódy použiť na renderovanie oblohy real-time.

2.4 3D obloha

Vytvoriť oblohu použiteľnú napríklad pre letecký simulátor je oveľa zložitejšie, keďže skybox alebo skydome očividne nestačia. Lietadlo sa môže pohybovať medzi oblakmi, či priamo v nich a teda tieto musia byť plne trojrozmerné. To nemusí platiť pre renderovaciu fázu, ako ukázal Harris, no v niektorom štádiu tvorby je potrebné vygenerovať celý objem oblaku.

Harris používa na reprezentovanie oblakov časticový systém, kde častica reprezentuje guľový objem, ktorého hustota klesá smerom od stredu k povrchu. Každá častica má definovaný stred, polomer, hustotu a farbu. Samotná oblačnosť je buď náhodne vygenerovaná, alebo vytvorená používateľom. Vykresľovací algoritmus pozostáva z dvoch fáz. Prvá je osvetľovacia fáza, prebieha len raz pre danú scénu, z čoho vyplýva, že oblaky aj osvetlenie je statické. Druhá fáza je samotné renderovanie a beží real-time. Prvá fáza používa osvetľovací model s viacnásobným rozptýlením svetla. V dvoch prechodoch vypočíta osvetlenie na základe fyzikálneho modelu. Renderovanie je riešené dynamicky generovanými impostormi, alebo podvodníkmi. Impostor je 2D textúra s obrazom vyrenderovaného oblaku, orientovaná smerom k pozorovateľovi, ako vidno na Obrázku 6.



Obrázok 6. Impostory orientované na kameru [Har02].

Výraz 'podvodník' si vyslúžila tým, že nie je renderovaná pre každý frame. To znamená, že ak sa pozorovateľ plynulo pohybuje, daný impostor je väčšinu času približným odhadom, skôr než presným renderom. Keď sa uhol, pod ktorým je oblak videný výrazne zmení, vyrenderuje sa nový impostor, pod adekvátnym uhlom. Týmto spôsobom dochádza k renderovaniu len vtedy, keď je nevyhnutné, a zároveň sa zabezpečí úroveň detailu podľa vzdialenosti, keďže pre vzdialené oblaky sa uhol pohľadu mení pomalšie ako pre blízke oblaky.

Man generuje oblaky priamo z perlinovho šumu. Svoju metódu popísal v článku [Man06]. Oblak tvorí 3D šumová funkcia, prirodzený tvar je dosiahnutý namapovaním na elipsoid, ktorého transparentnosť postupne klesá smerom k povrchu. Perlinov šum však vygeneruje priveľa informácie, na to aby sa dal zobraziť, preto treba Man aproximuje takýto oblak štruktúrou metagulí. Je to vlastne reprezentácia, kde jedna jedna metaguľa približne odhadne celý zhluk voxelov šumu. Metaguľa je guľový objem, ktorého hustota je vyjadrená funkciou vzdialenosti od stredu, a klesá k nule na hranici gule, podobne ako častice v predchádzajúcej metóde. Reprezentácia je vypočítaná RBF (radial basis function) neurónovou sieťou. Počet neurónov tejto siete vyjadruje počet výsledných metagúľ a dá sa nastaviť, čím prakticky vytvára účinnú úroveň detailu. Vytvorenie oblaku a aproximácia prebieha raz pre danú scénu. Osvetlenie prebieha v renderovacej časti, ktorá beží real-time. To znamená, že hoci sú oblaky statické, zdroj svetla sa môže real-time meniť. Renderovací algoritmus je založený na fyzikálnom modeli osvetlenia. Počíta sa viacnásobné rozptýlenie svetla; v dvoch prechodoch.

3. Implementácia

Naša metóda je založená na oddelení jednotlivých fáz tvorenia oblohy. Hlavným prostriedkom je to, že oblaky sú diskrétne objekty. Môžu tiež byť navzájom nezávislé. Na rozdiel od metód, ktoré priamo zo vstupu odvodia vzhľad oblakov - obzvlášť metódy používajúce šum - možno každému oblaku určiť vlastnosti, umiestnenie a podobne. Zdrojové kódy programov sa nachádzajú na priloženom CD.

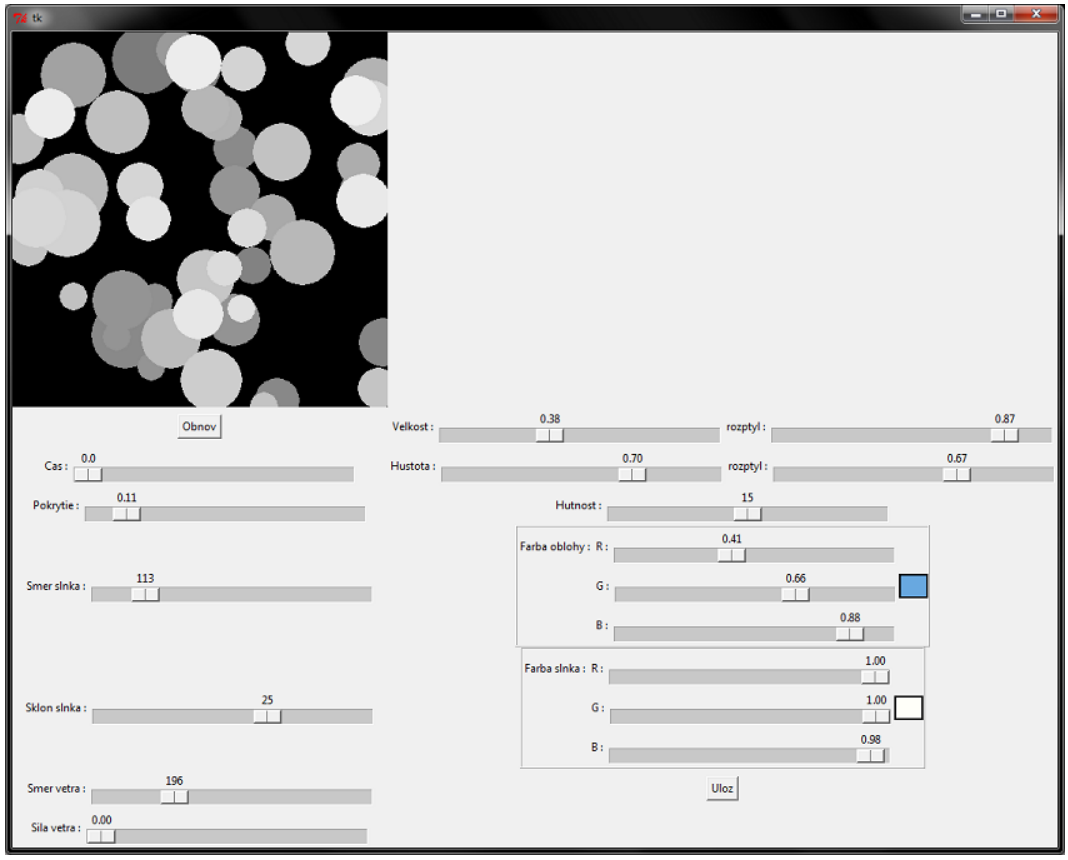
3.1 Štruktúra

Obloha je reprezentovaná objektom triedy 'Skymap' - mriežkou 40x40 objektov triedy 'Cloud'. 'Cloud' je oblak a má jeden dcérsky objekt triedy 'Gula' a vlastnosti pre oblak ako celok - napríklad absolútnu pozíciu, veľkosť, šírku, dĺžku, hustotu a tak ďalej. Objekt 'Gula' je stavebným kameňom oblaku. Má určitú veľkosť a niekoľko menších, dcérskych gúľ. Tie majú svoje dcérske gule a tak ďalej. Rekurzia sa zastaví, keď je veľkosť bilboardu, ktorý bude guľu reprezentovať menšia ako konštanta - v našom programe nazvaná 'rozlisenie'.

Guľa má relatívnu pozíciu, závislú od jej otcovského objektu 'Cloud'. Obsahuje zoznam uhlov a ich modifikátorov. Dcérske gule vytvára v určitej vzdialenosti od svojho stredu, pod uhlami (zvislý a vodorovný rovinný uhol) z tohto zoznamu. Vzdialenosť a presná pozícia záležia na vlastnostiach oblaku ako napríklad dĺžka, či hustota.

3.2 Generovanie

Prvá fáza generovania je definovanie stavov oblohy pre rôzne časy. Stavy sa vytvárajú v pomocnom programe - screenshot na obrázku 7. Program je napísaný v jazyku Python, pomocou štandardnej GUI knižnice Tkinter a knižnice Pygame, použitej na vykresľovanie náhľadu. Obsahuje nastavenia oblakov a oblohy. Približný efekt týchto nastavení sa zobrazuje v okne s náhľadom, akoby pohľadom na oblaky zhora. Tlačidlo 'Ulož' pridá výpis pre nastavený čas na koniec súboru skyfile.sf.



Obrázok 7. Pomocný program na prípravu skyfile.sf

Každý stav má unikátne číslo od 0.0 po 23.9, ktoré reprezentuje jeho čas. Ďalej pre stav nastavíme rôzne vlastnosti oblohy, či jednotlivých oblakov. Napríklad poloha a farba slnka, priemerná veľkosť oblaku a náhodný rozptyl a iné. Z týchto údajov potom program obsadí 'oblakmi' mriežku 40x40 políčok (pričom niektoré políčka môžu obsahovať 'nulové oblaky'). Pre oblohu aj pre oblaky sa dá vytvoriť ľubovoľne veľa vlastností, ktoré potom interpretuje hlavný program. Všetky údaje sú následne uložené v súbore s príponou '.sf' ako zoznam stavov a zodpovedajúcich nenulových oblakov. Ukážka formátu je vo výpise 1.

```
#0.0 0 90 270 2 1 1 1 0.41 0.66 0.88 14
115 55 35 0.850213080645
251 51 35 0.808865665019
338 55 35 0.819501750461
49 62 35 0.733085065692
```

```
348 133 35 0.737893225729
159 142 35 0.818509072202
177 149 35 0.750334660344
#0.11 0 90 270 2 1 1 1 0.41 0.66 0.88 14
20 16 35 0.773500087373
272 11 35 0.700546580951
...
```

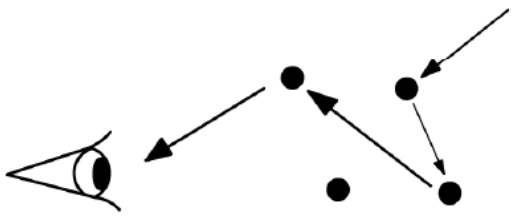
Výpis 1. Ukážka formátu skyfile.sf. Riadok začínajúci znakom # obsahuje všeobecné vlastnosti oblohy, nasledujú jednotlivé oblaky.

Druhá fáza prebieha v hlavnom programe. Program načíta 'skyfile.sf' a animuje od prvého stavu k poslednému určitou rýchlosťou. Animácia pre daný bod v čase vznikne lineárnou interpoláciou najbližších stavov pred a po tomto bode. Určia sa vlastnosti všetkých oblakov a pokračuje sa úpravou objektov.

Úprava spočíva v prepočítaní pozícií oblakov, tak aby sa zohľadnil vietor a zakrivenie vrstvy oblakov podľa vzdialenosti od stredu priestoru. Pokračuje sa prepočítaním uhlov gulí, ich pozícií, veľkostí a tak podobne. Každá upravená guľa pridá svoju pozíciu a veľkosť do zoznamu používaného pri renderovaní. Taktiež guľa vytvorí, prípadne zničí časť potomkov, podľa nových vlastností oblaku a novej veľkosti. V tejto fáze sa tiež počíta vzdialenostKSlnku, spomínaná v časti 3.3.

3.3 Nasvietenie

"'Multiple scattering' - viacnásobné rozptýlenie - je rozptýlenie svetla viacerými časticami po sebe. " [Har02].



Obrázok 8. Viacnásobné rozptýlenie svetelného lúča časticami oblaku [Har02]

Priesvitné objekty s viditeľnou hustotou, ako napríklad oblaky, rozptyľujú svetlo na tomto princípe. "'Modely osvetlenia simulujúce viacnásobné rozptýlenie svetla sú fyzikálne presnejšie, ale musia počítať rozptýlenie všetkými smermi a preto sú omnoho komplikovanejšie a výpočtovo zložitejšie." [Har02]. Tento efekt náš program iba napodobňuje.

Každý oblak má základnú farbu. V procese nasvietenia si ju každá guľa upraví. Jednak podľa toho, či sa nachádza na povrchu oblaku, alebo v jeho vnútri. Druhým faktorom ja poloha vzhľadom na osvetlenie. V rámci úpravy gule sa pre ňu vypočíta takzvaná 'vzdialenostKSlnku'. Je to z-ová súradnica v pomocnej súradnicovej sústave. Z-ový vektor tejto sústavy zároveň určuje smer z ktorého momentálne svieti slnko. Následne sa táto vzdialenosť normalizuje. Z nej a z čísla vyjadrujúceho ako blízko pri povrchu sa nachádza guľa sa vypočíta konečné osvetlenie gule. Toto osvetlenie určí aký zlomok farby slnka sa pripočíta k základnej farbe oblakov.

3.4 Renderovanie

Pri samotnom renderovaní je každá guľa reprezentovaná billboardom - 2D textúrou v priestore. Jeho stred určuje pozícia gule. Jeho rohy sa dopočítajú na základe smeru pohľadu kamery, tak aby veľkosťou odrážal veľkosť gule a aby bol otočený oproti smeru kamery. Ideálne by bolo, keby bol billboard otočený priamo na kameru a nielen jej smerom. Vyriešilo by to chybu perspektívy, ktorá takto vzniká v rohoch výsledného skyboxu. Vzhľadom na vzdialenosť billboardov je však táto chyba zanedbateľná a kvôli zjednodušeniu renderovania sme sa rozhodli pre otočenie smerom opačným než určuje vektor pohľadu predávaný funkcii glLookAt. Po vyrenderovaní všetkých oblakov sa pre štyri pohľady do strán vyrenderuje hmla, ktorá akoby zakrývala oblaky v diaľke. To je v skutočnosti textúra v popredí - plynulý prechod od šedej farby k priehľadnosti, zdola nahor.

3.5 Ukladanie

Ako už bolo spomenuté, existuje niekoľko možností off screen renderovania. My

sme sa rozhodli pre použitie štandardného vykresľovacieho buffera a volania glReadPixels, ktoré z tohto buffera získa dáta v podobe "surového" reťazca. Následne tento reťazec uložíme pomocou voľnej knižnice PIL - Python Imaging Library. Ukážka kódu je vo výpise 2. Výhodou je, že vykresľovací buffer ostane naplnený a dá sa pre OpenGL prirodzeným spôsobom zobraziť do okna - napríklad pre jednoduchšie ladenie programu.

```
def screenshot( l ):
    global counter
    data = glReadPixels( 0, 0, 500, 500, GL_RGB, GL_UNSIGNED_BYTE )
    im = Image.frombuffer( "RGB", (500, 500), data, "raw", "RGB", 0, 0 )
    im = im.transpose( Image.FLIP_TOP_BOTTOM )
    counter += 1

    #vytvor meno pre obrazok
    num = str( counter/5 )
    num = '0'*(3-len( num )) + num
    num = path+'screenshot_'+ str( l ) + num +'.png'
    im.save( num )
```

Výpis 2. Zdrojový kód funkcie, ktorá ukladá vyrenderovane obrázky.

Vznikne sada obrázkov, ktoré reprezentujú jednotlivé steny skyboxu. Tieto obrázky je možné spojiť do videí, pre uľahčenie manipulácie. My sme sa rozhodli pre použitie programu voľne šíriteľného programu FFmpeg, videá sme ukladali do formátu .avi, pod kodekom mpeg4. FFmpeg je cross-platformové riešenie na nahrávanie, konverziu a šírenie audia a videa. Jedným z možných použití je uloženie sady obrázkov pod niektorým z mpeg kodekov do rôznych video formátov. Dá sa takisto použiť na opačnú konverziu - vytvorenie obrázkov z videa, alebo na otvorenie videa a načítanie frameov v inom programe, pomocou jeho knižníc.

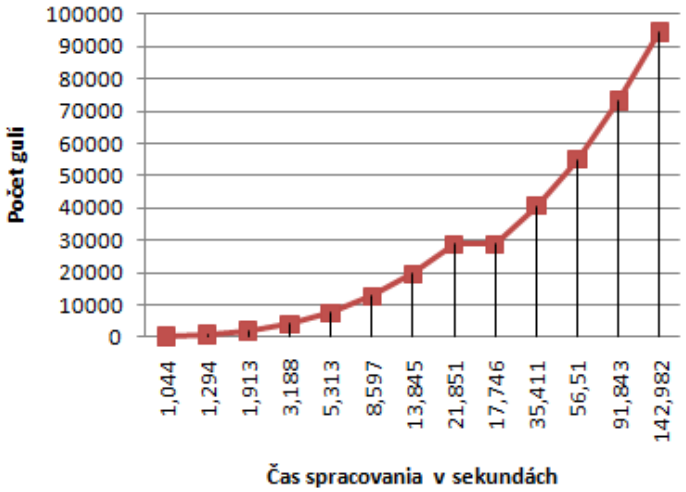
3.6 Použitie

Použitie spočíva v postupnom premietaní jednotlivých frameov na skybox v hre alebo virtuálnom prostredí. To sa dá spraviť buď vytvorením obrázkov z videí pred renderovaním skyboxu a ich postupným načítaním, alebo otvorením video streamov a načítaním frameov priamo z nich. Obrázky, ktoré sú výstupom programu, sú označené písmenom a indexom, napríklad _b003. Písmeno označuje stenu skyboxu a môže byť f-front (predná, smerom po kladnej x-ovej osi), b-back (zadná), l-left (ľavá), r-right (pravá) alebo t-top (vrchná). Podobné označenie sa predpokladá aj pri videách.

Pre potreby ladenia sme implementovali prehliadač animovaných skyboxov (nie je súčasťou prílohy). Screenshot z tohto programu je na obrázku 15. Náš prehliadač načíta všetky obrázky dopredu, ako pole textúr, pomocou glGenTextures. To preto, lebo textúry prehráva veľmi rýchlo. Pri aspoň trochu realistickej rýchlosti prehrávania vôbec nie je vylúčené načítavanie za behu programu. Vykresľovací cyklus pripojí správnu textúru a pomocou priameho vykresľovacieho módu GL_QUADS vykreslí stenu, toto opakuje pre zvyšné steny. Súradnice rohov stien sa dajú zadať priamo, alebo sa na ich umiestnenie môžu využiť transformácie, ktoré ponúka OpenGL. Tiež sa môže použiť iný vykresľovací mód (display lists, vertex arrays) aby sa zrýchlil rendering alebo zmenšil dátový prenos. Nezávisle od rýchlosti renderingu sa vymieňajú textúry skyboxu. To sa dá buď vo vykresľovacom cykle, pomocou sledovania času; alebo pomocou časovaných eventov. Z hľadiska rýchlosti by bolo ideálne mať obrázky jedného framu spojené do jednej textúry. Potom by sa nemusela pripájať textúra zvlášť pre každú stenu, v každom prechode cyklu. Textúra by sa tak pripájala len v rámci animácie, raz za nejaký časový úsek.

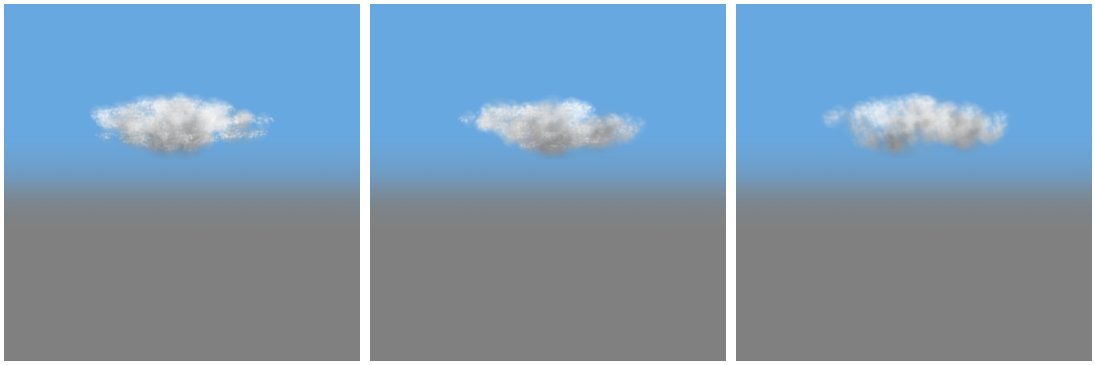
4. Výsledky

Generovanie bolo testované na počítači s 4GB RAM, s procesorom AMD Athlon s frekvenciou 1600 MHz. Čas sa meral v sekundách a množstvo práce je vyjadrené počtom gúľ, ktoré treba spracovať a vyrenderovať. Test prebiehal pri hodnote rozlíšenia 10. Jedno meranie zahŕňa jeden frame oblohy, teda úpravu oblakov a vyrenderovanie a uloženie 5 stien skyboxu (spodnú stenu nerenderujeme). Steny majú veľkosť 500x500 pixelov. Výsledok ukazuje následujúci Graf 1.



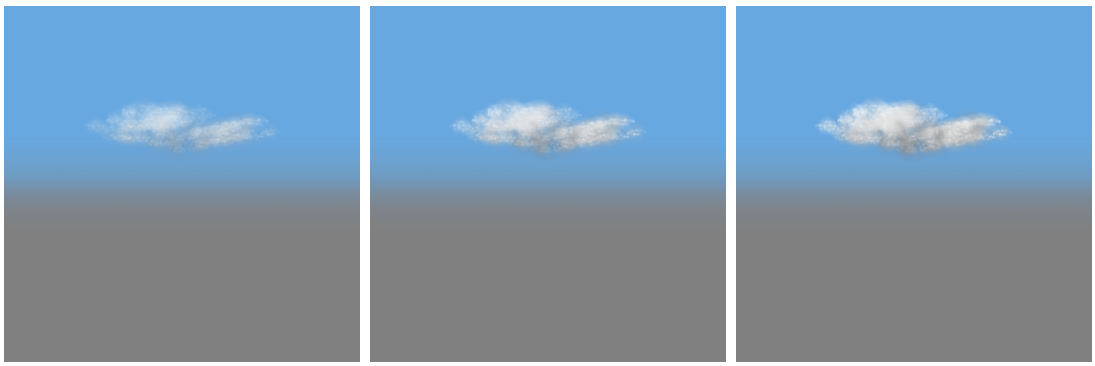
Graf 1. Závislosť času spracovania frameu od počtu gulí oblohy.

Kvalitu približne určuje parameter rozlisenie. Udáva, pri akej veľkosti zobrazenia bilboardu guľa ukončí rekurziu. Následujúce obrázky ukazujú ten istý vstup pri rozlíšení 1, 30 respektíve 60:



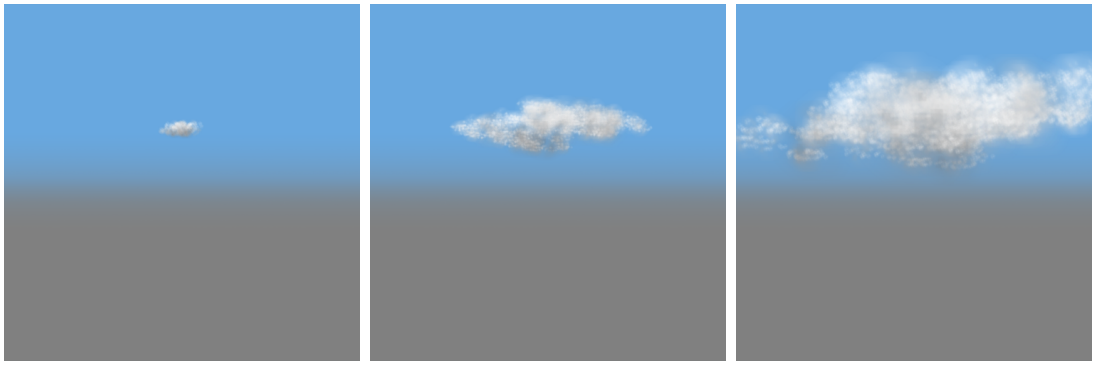
Obrázok 9. Rozlíšenie 1, 30 a 60.

Nižšia hodnota parametra rozlisenie teda zlepší kvalitu detailov, ale povážlivo zvýši čas generovania. Použitie výsledku však nijako neovplyvňuje a nemá vplyv na rýchlosť zobrazenia. Vzhľad oblakov ďalej závisí od hustoty oblaku - jeho priehľadnosti. Tá je vyjadrená číslom od 0 do 1. Následujúce obrázky ukazujú ten istý oblak s hustotou 0,3 0,6 respektíve 0,9:



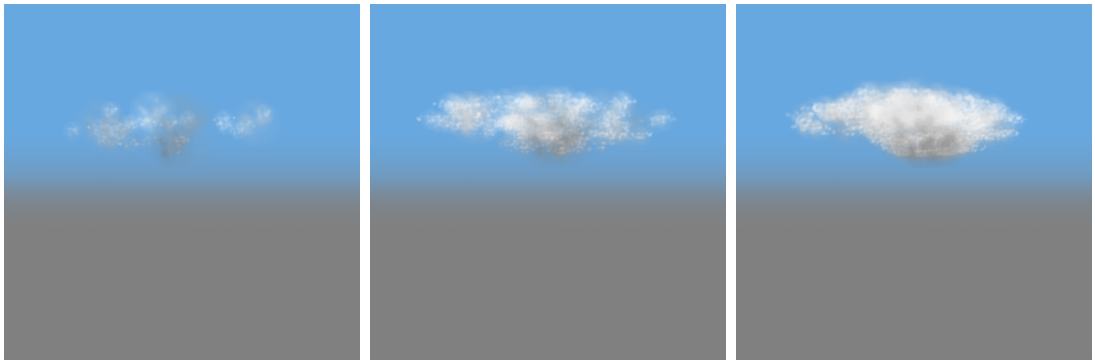
Obrázok 10. Hustota 0,3 0,6 a 0,9.

Ďalším parametrom je veľkosť - kladné číslo, rozumné výsledky sú pre hodnoty približne od 3 do 30. Následujúce obrázky ukazujú ten istý oblak s veľkosťou 3 10 respektíve 20:



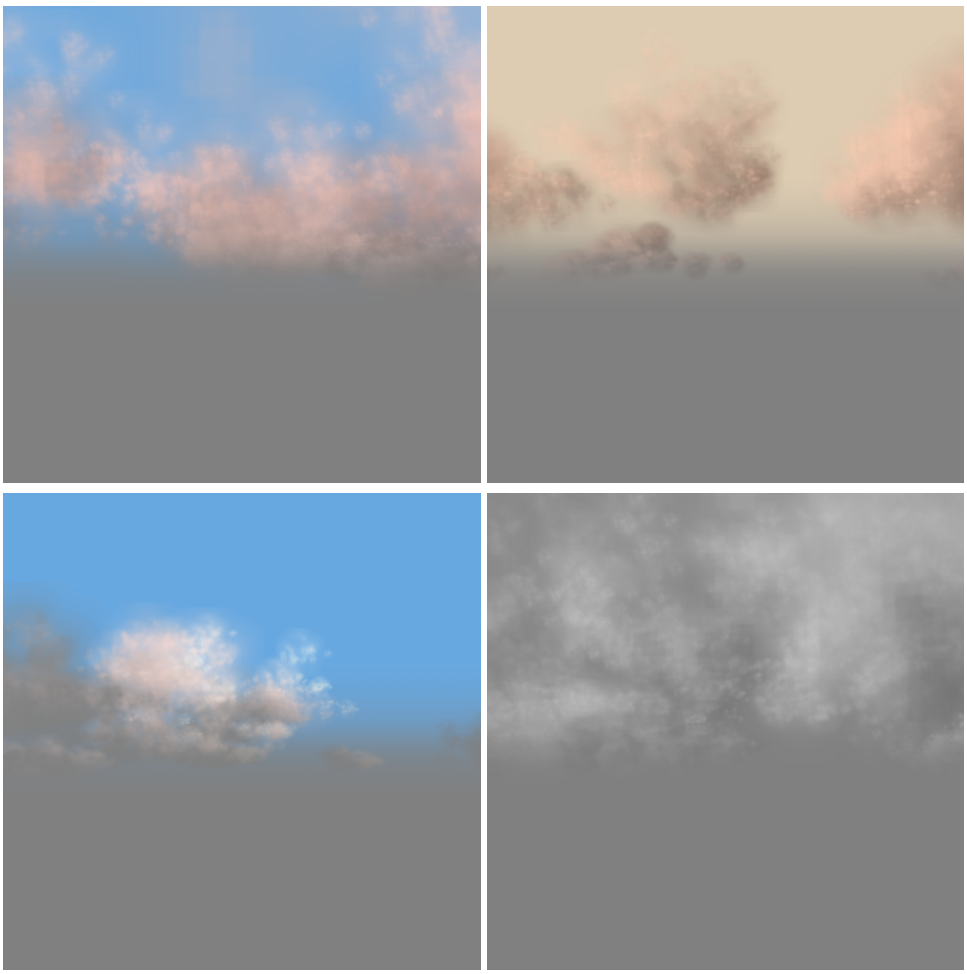
Obrázok 11. Veľkosť 3, 10 a 20.

Parameter pocet určuje počet detí každej gule a vyjadruje akúsi hutnosť oblaku. Opäť ukážka, ten istý oblak s hutnosťou 5 12 respektíve 20:

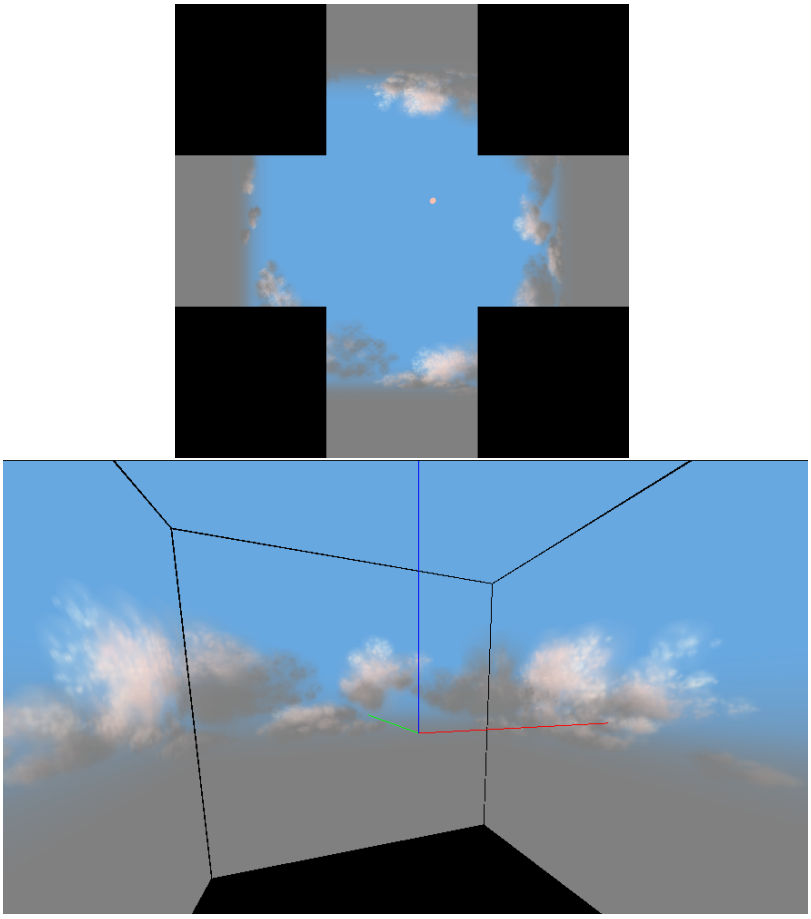


Obrázok 12. Počet potomkov 5, 12 a 20.

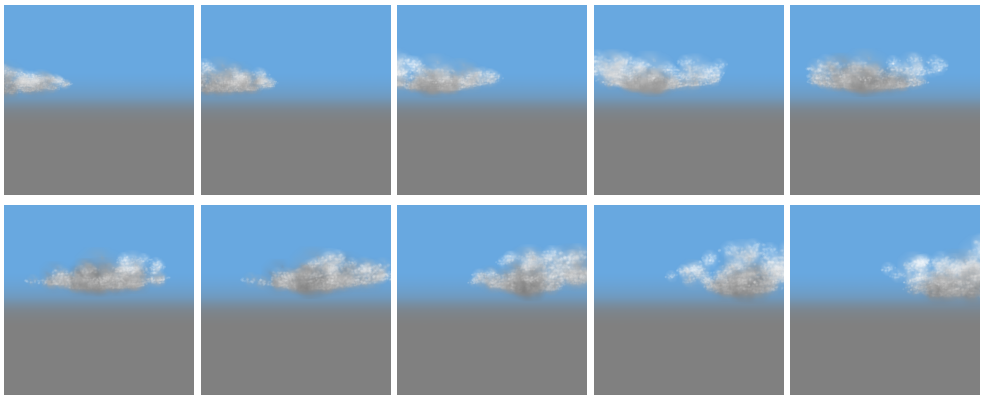
Ďalšie parametre určujú smer a farbu slnečných lúčov, farbu oblohy, a posun pri animácii, teda smer a silu vetra. Nasleduje pár ďalších ukážok. Videá s animáciou sa nachádzajú na priloženom CD.



Obrázok 13. Rôzne nastavenia.



Obrázok 14. Skybox rozložený hore a namapovaný na kocku dolu. Pre lepšiu ilustráciu sú steny skyboxu schválne oddelené a tiež kamera nie je v strede kocky.



Obrázok 15. Animácia oblaku.

Záver

V práci podávame základné informácie o oblohe a tiež o grafickej knižnici OpenGL. Nasleduje všeobecný úvod do generovania počítačovej oblohy. Tento by mal čitateľovi pomôcť zorientovať sa v problematike a oboznámiť ho so základnými pojmami a špecifikami počítačovej oblohy. Takisto je rozobraných niekoľko konkrétnych riešení tvorby oblačnosti, ktoré slúžia ako prehľad základných metód. Na záver prezentujeme metódu na tvorbu 3D oblačnosti. Podarilo sa nám dosiahnuť dostatočne dôveryhodnú oblohu. Táto je plynulo animovaná, pričom sa dá zvoliť rýchlosť animácie. Takisto funguje dynamické osvetlenie a animácia vzniku a zanikania oblakov. Prínos našej metódy vidíme hlavne v jej flexibilite - oddelení parametrov oblohy od samotného spôsobu jej generovania. To znamená aj jednoduchšie a prehľadnejšie úpravy kódu v budúcnosti. Prípadná ďalšia práca by sa mohla týkať optimalizácie, a prepísania programu do rýchlejšieho programovacieho jazyka, či grafických shaderov. Takisto sa dá pracovať na generovaní, či osvetľovacom modeli. Dobrým nápadom by mohlo byť integrovanie systému do hry alebo simulátoru pre real time generovanie, či možnosť generovať spojitú slučku animácie oblohy.

Zoznam Literatúry

[Ahn08] Song Ho Ahn. Opengl frame buffer object (fbo). http://www.songho.ca/opengl/gl_fbo.html, 2008.

[Bel11] Fabrice Bellard. Ffmpeg. http://www.ffmpeg.org/, 2011.

[Bri97] Paul Brien. Course notes on: Opengl and window system intergration, course 24 at siggraph '97. http://www.mesa3d.org/brianp/sig97/offscrn.htm, 1997.

[Cop08] Davide Coppola. Algorithms: Perlin noise. http://www.m3xbox.com/GPU_blog/?p=28, 2008.

[Eli11] Hugo Elias. Cloud cover. http://freespace.virgin.net/hugo.elias/models/m_clouds.htm, 2011.

[Fou11] Python Software Foundation. About python. http://python.org/about/, 2011.

[Gro99] Khronos Group. Opengl notes on alpha blending. http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node203.html, 1999.

[Gro10] Khronos Group. Opengl homepage. http://www.opengl.org/about/overview/, 2010.

[Gro11] Khronos Group. Glut and opengl utility libraries. http://www.opengl.org/resources/libraries, 2011.

[Har02] M. J. Harris. Real-time cloud rendering for games. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.9149\&rep=rep1\&type=pdf, 2002.

[Man06] Petr Man. Generating and real-time rendering of clouds. http://www.cg.tuwien.ac.at/hostings/cescg/CESCG-2006/papers/Prague-Man-Petr.pdf, 2006.

[Per99] Ken Perlin. Making noise. http://www.noisemachine.com/talk1/index.html, 1999.

[Per11] Ken Perlin. Ken perlin bio. http://cs.nyu.edu/~perlin/doc/bio.html, 2011.

[pro11] SourceForge PyOpenGL project. Pyopengl 3.x. http://pyopengl.sourceforge.net/, 2011.

[Qui05] Inigo Quilez. Dynamic clouds. http://www.iquilezles.org/www/articles/dynclouds/dynclouds.htm, 2005.

[Rob11] Mark R Robertson. Video file container formats, compression and codecs. http://www.reelseo.com/file-formats-containers-compression, 2011.

[Sms97] Inc Science made simple. Why is the sky blue? http://www.sciencemadesimple.com/sky_blue.html, 1997.

[wik11] wiki. Cloud. http://en.wikipedia.org/wiki/Cloud, 2011.

[WNDS99] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Pub. Co., USA, 1999.