STUDIA INFORMATICA 2007
Volume 28 Number 1 (70)

Adrian DĘBOWSKI, Jakub GRUDZIŃSKI, Tomasz GRUDZIŃSKI
Politechnika Śląska, Instytut Informatyki

PROBLEMY I METODY WIZUALIZACJI ZJAWISK ATMOSFERYCZNYCH W CZASIE RZECZYWISTYM W ŚRODOWISKU FRS

Streszczenie. W publikacji przedstawiono problemy powstające przy próbie wizualizacji w czasie rzeczywistym nieba i atmosfery na animowanej scenie trójwymiarowej i zaproponowano metody ich rozwiązania.

Słowa kluczowe: wirtualna rzeczywistość, wizualizacja atmosfery, grafika komputerowa

THE ISSUES AND METHODS OF THE REAL-TIME ATMOSPHERIC PHENOMENA RENDERING IN FRS ENGINE

Summary. The paper presents the issues of real-time sky and atmosphere rendering on the animated 3-dimensional scene and proposes a number of solutions to this subject.

Keywords: virtual reality, atmosphere rendering, computer graphics

1. Środowisko FRS

FRS (ang. *Flexible Reality Simulation*) jest silnikiem programistycznym, pozwalającym na łatwe pisanie aplikacji opartych na wirtualnej rzeczywistości. Silnik jest obecnie intensywnie rozwijany i nie wszystkie założenia zostały spełnione, jednak osiągnięto już pewną funkcjonalność pozwalającą na zaprezentowanie prac szerszej publiczności i wyciągnięcie pewnych wniosków[1].

[1] Różne zagadnienia związane z silnikiem FRS opisano w [7, 19, 20].

Twórcy FRS postawili sobie za główny cel stworzenie silnika uniwersalnego i łatwego w użyciu, który poza standardowymi elementami podobnych projektów zawierałby komponenty bardziej wyspecjalizowane, możliwe do użycia, lecz nieobowiązkowe dla piszącego aplikację bazującą na silniku. W ten sposób powstaje silnik, który poza możliwością wyświetlania trójwymiarowej grafiki, interakcją między obiektami i sprawną obsługą dźwięku, umożliwi także symulację pogody, programowe generowanie modeli roślinności, symulację mimiki ludzkiej twarzy, użycie zaawansowanych algorytmów sztucznej inteligencji oraz sterowanie całością poprzez oferujący dużą funkcjonalność język skryptowy.

W dalszej części publikacji przedstawiono problemy wizualizacji zjawisk atmosferycznych oraz metody ich rozwiązania, które analizowane były podczas prac badawczych nad komponentami do wyświetlania nieba w silniku FRS.

2. Sklepienie niebieskie

Wizualizację zjawisk atmosferycznych przeprowadzano w trójwymiarowym środowisku. Obserwatora umieszczono w środku kopuły, która odzwierciedla sklepienie niebieskie. Przestrzeń otaczającą obserwatora wraz z powierzchnią kopuły określa się sceną symulacji.

2.1. Gwiazdy

Każda trójwymiarowa scena powinna być renderowana[2] na jakimś podkładzie ograniczającym teoretycznie nieskończony horyzont. W przypadku rysowania nieba tłem są gwiazdy oraz inne ciała niebieskie znacznie oddalone od analizowanej sceny.

Umieszczanie na scenie każdej gwiazdy jako osobnego obiektu jest praktycznie niewykonalne ze względu na ich liczbę. Nawet jeśli uwzględni się tylko najważniejsze z nich, przetwarzanie może zająć znaczną część dostępnego czasu procesora, a przecież gwiazdy mają stanowić tylko tło.

Wszystkie gwiazdy znajdują się jednak tak daleko od obserwatora, że można traktować je jako obiekty nieruchome względem siebie, niezależnie od pozycji obserwatora i czasu. Można więc rzutować je na teksturę, która zostanie odwzorowana na kopułę przed rozpoczęciem symulacji.

Obraz tworzony przez akceleratory graficzne składany jest z płaskich ścianek w kształcie trójkąta. Nie ma więc możliwości utworzenia idealnej powierzchni sferycznej. Dokładność

[2] Renderowanie (ang. *rendering*) – przeliczenie sceny trójwymiarowej i utworzenie pliku wyjściowego w formie obrazu statycznego lub animacji.

odwzorowania kopuły rośnie wraz z liczbą trójkątów, z których jest zbudowana. Niestety wraz z liczbą trójkątów rośnie obciążenie procesora. Aby całkowicie usunąć błędy rzutowania związane z niedokładnością kopuły oraz zminimalizować liczbę potrzebnych ścianek, do wyświetlenia gwiazd wykorzystuje się technikę zwaną *skybox*.

Skybox to sześcienna mapa otoczenia. Obserwator znajdujący się w środku skyboxa, który został okryty specjalnie przygotowanymi teksturami, ma wrażenie, jakby znajdował się wewnątrz kuli [2]. Zamiast jednego rzutowania sferycznego gwiazd wykonuje się sześć rzutów prostokątnych.

Takie rozwiązanie ma jeszcze jedną zaletę. Pozorny ruch sklepienia niebieskiego, spowodowany obrotem Ziemi wokół własnej osi, można zrealizować przez prostą rotację jednej bryły (sześcianu) zamiast skomplikowanego obliczeniowo korygowania mapowania tekstury na poszczególne ścianki kopuły.

Skyboxy szeroko wykorzystywane są także do tworzenia całego tła symulacji. Zamiast tekstur gwiazd stosuje się tekstury otoczenia. Takie rozwiązanie jest najprostszą i zarazem najszybszą metodą wykorzystywaną bardzo często w grach komputerowych.

2.2. Słońce

Pozorny ruch Słońca po nieboskłonie nie może już być opisany w tak prosty sposób, ponieważ zależy on również od pozycji Ziemi na orbicie. Pozycja Słońca może być wyliczona w dowolnym punkcie czasu z dużą dokładnością (poniżej dwóch minut kątowych) przy użyciu średnich składników orbitalnych [3]. Uzyskane wartości w układzie horyzontalnym, czyli azymut i wysokość obiektu nad horyzontem, są bardzo istotne w dalszych etapach wizualizacji.

Do wyświetlenia tarczy słonecznej wystarczy technika cząsteczek, czyli teksturowanych, prostokątnych obiektów, skierowanych zawsze w stronę obserwatora [4].

Ze słońcem wiążą się dodatkowo efekty flary i oślepienia. Ludzkie oko posiada zdolność adaptacji do zmiennego natężenia światła. Ten sam obiekt na niebie (np. chmura) może raz wydać się oślepiająco jasny, a innym razem znaczenie ciemniejszy. Efekt ten powstaje w momencie, gdy w polu widzenia znajduje się inny obiekt o względnie silniejszym natężeniu emitowanego światła. Tak też dzieje się w przypadku zerkania na słońce, co pokazuje rys. 1 – kontury tarczy Słońca są rozmyte, natomiast błękit nieba wydaje się być nienaturalnie ciemny. Monitory komputerowe mają ograniczoną maksymalną jasność świecenia, nie da się więc osiągnąć naturalnego efektu. Należy zatem odwrócić proces adaptacji oka przez sztuczne ściemnienie wszystkich obiektów na scenie poza słońcem, które jest tym silniejsze, im wzrok wirtualnego obserwatora będzie bardziej skupiony na tarczy słonecznej.

Rys. 1. Flara słoneczna
Fig. 1. Solar flare

Aby efekt był realistyczny, cząsteczce reprezentującej słońce musi towarzyszyć bardzo silne źródło światła, natomiast cały system renderingu powinien uwzględniać algorytm HDR (*High Dynamic Range rendering* [17, 18]). Do działania w czasie rzeczywistym algorytm wymaga procesora graficznego o dużej mocy obliczeniowej, obsługującego m.in. tekstury zapisane w formacie zmiennoprzecinkowym [16].

W przypadku braku sprzętowego wsparcia dla HDR można zastosować techniki zastępcze. Do ściemnienia sceny można wykorzystać *shadery pikseli*[3], jednak jest to metoda ingerująca bezpośrednio w każdy obiekt na scenie. Lepszym rozwiązaniem okazuje się stworzenie kolejnego obiektu, który w prosty sposób (np. ciemne okulary) przysłoni inne obiekty. Tu również idealnie sprawdza się technika cząsteczek. Stopień ściemnienia można ustawiać poprzez zmianę stopnia przezroczystości cząsteczki.

Tarcza słońca jest tak jasna, że samo ściemnienie sceny nie wystarcza. Na rys. 1 nie da się zobaczyć wyraźnych konturów tarczy. Zamiast tego można zaobserwować efekt flary słonecznej, który realizuje się podobnie jak ściemnienie, wykorzystując cząsteczki. Tutaj również flara jest tym silniejsza, im bardziej wzrok wirtualnego obserwatora jest skupiony na centrum słońca. Fragment tekstury pokrywającej cząsteczkę flary przedstawiono na rys. 2.

[3] Shader pikseli (od angielskiego słowa *shade* – cieniować) – program generujący kolor poszczególnych pikseli renderowanych prymitywów, uruchamiany w środowisku GPU (*Graphic Processing Unit*), tzn. programowalnej jednostki graficznej.

Rys. 2. Tekstura flary słonecznej
Fig. 2. The texture of the Sun's flare

2.3. Inne ciała niebieskie

Inne ciała niebieskie, w szczególności Księżyc, mogą być wizualizowane w podobny sposób jak Słońce. Jasność obiektów nie jest już jednak tak duża, więc nie ma potrzeby stosowania ściemnienia sceny dla zwiększenia kontrastu.

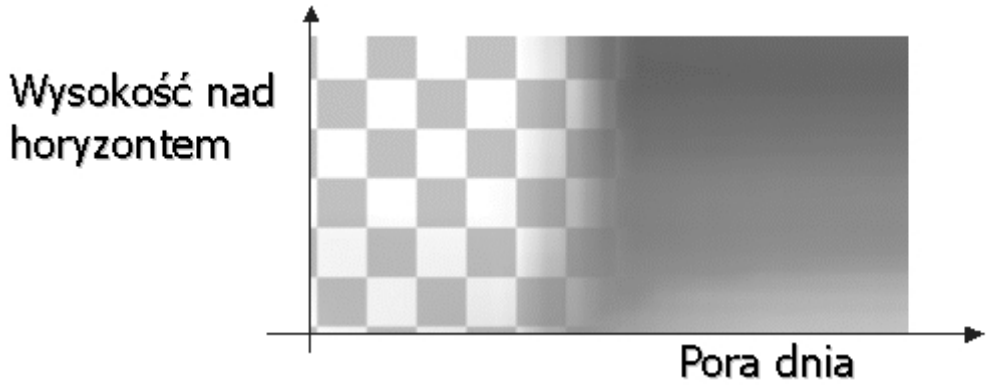
2.4. Atmosfera

Gwiazdy i ciała niebieskie stanowią pierwszą warstwę sceny. Drugą warstwę stanowi atmosfera. Czyste niebo widoczne jest jako rozproszone przez powietrze promieniowanie słoneczne. Gdyby światło słoneczne nie było rozpraszane przez atmosferę, to niebo byłoby czarne i w dzień widoczne byłyby gwiazdy. Cząsteczki gazów najlepiej rozpraszają fale krótkie, czyli niebieska część widzialnego spektrum [5]. Im więcej gazu dzieli obserwatora od czerni kosmosu, tym więcej światła może się w nim rozproszyć. Dlatego błękit nieba tuż nad horyzontem jest znacznie jaśniejszy niż widziany w zenicie.

Kolor nieba można więc wyznaczyć jako funkcję dwóch zmiennych: wysokości punktu nad horyzontem, dla którego wyznaczany jest kolor, oraz natężenia promieniowania słonecznego, czyli pory dnia lub inaczej wysokości słońca nad horyzontem. Każda zmienna ma oznaczoną dziedzinę, dzięki czemu możemy wyliczyć wartości tej funkcji dla charakterystycznych punktów jeszcze przed rozpoczęciem symulacji.

Tak wyliczone wartości należy nanieść na teksturę gradientu atmosferycznego, którą prezentuje rys. 3. Procesor graficzny zapewni wybrany rodzaj interpolacji między tekselami[4]. Całkowicie wystarczająca będzie tutaj interpolacja liniowa.

[4] Teksel (ang. *telex, texture element*) – pojedynczy element rastra tekstury.



Rys. 3. Tekstura gradientu atmosferycznego
Fig. 3. Atmospheric gradient texture

Tekstura ma kanał alfa[5]. Na rys. 3 przezroczyste teksele ukazują szachownicę znajdującą się pod spodem. Podczas wizualizacji przez przezroczyste powierzchnie atmosfery ukazywać się będzie pierwsza warstwa sceny, a więc gwiazdy. Im mniej światła rozprasza się w atmosferze, tym staje się ona bardziej przezroczysta, a więc ciemniejsza.

W praktyce teksturę gradientu atmosferycznego przygotowuje artysta-grafik, ponieważ jest to rozwiązanie szybsze niż opracowanie wzoru analitycznego funkcji.

3. Metody wyświetlania chmur

Chmury są bardzo charakterystycznym elementem nieba. Na niektórych szerokościach geograficznych bezchmurne niebo może się wydawać nienaturalne. Ze względu na swoją złożoność są one najtrudniejszym elementem do wizualizacji.

Aby symulacja mogła odbywać się w czasie rzeczywistym, trzeba godzić się na pewne uproszczenia. Celem staje się znalezienie złotego środka między wydajnością systemu renderującego a dokładnością odwzorowania rzeczywistości. Wybór metody zależy najczęściej od możliwości sprzętowych platformy, na której przeprowadzona ma być wizualizacja.

3.1. Układy proste

Najprostsza metoda nie ma wiele wspólnego z symulacją, jednak jest zdecydowanie najszybsza. Gdy z pewnych względów twórcy nie interesują dynamicznie zmieniające się

[5] Kanał alfa (ang. *alpha channel*) – czwarty składnik koloru w przestrzeni barw RGBA, określający nieprzezroczystość pikseli/tekseli.

układy chmur, można przygotować gotowe zdjęcia nieba i wyświetlać je na płaskiej powierzchni nad obserwatorem.

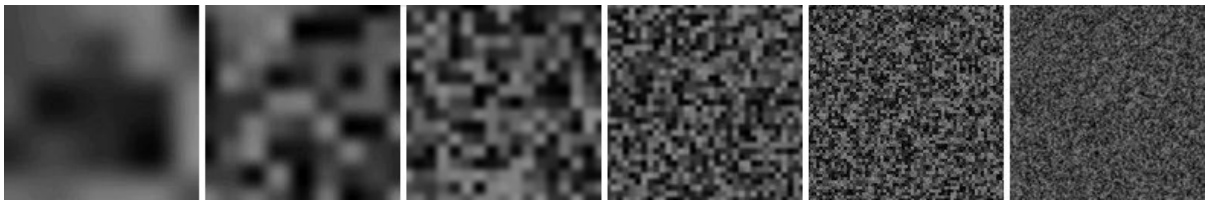
Nieco bardziej zaawansowana metoda uwzględnia również wiatr, przez proste przesuwanie tekstury po wspomnianej powierzchni. Wymaga to jednak specjalnie spreparowanego zdjęcia, w którym koniec tekstury jest jednocześnie jej początkiem (tekstura ciągła).

3.2. Tekstury proceduralne

Statyczne chmury sprawdzają się tylko wtedy, gdy niebo jest obserwowane przez krótką chwilę. Dynamika rozwoju i zaniku chmur na prawdziwym niebie jest bowiem zauważalna dla człowieka gołym okiem. Poza tym, nienaturalnie wygląda niebo, gdy patrząc na nie co jakiś czas, za każdym razem obserwuje się ten sam układ chmur. Zważywszy na charakter chmur nieuniknione wydaje się zastosowanie generatorów pseudolosowych.

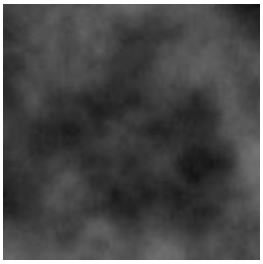
Bardzo interesującym rozwiązaniem, które daje zadawalające wyniki, są tekstury proceduralne [9, 11], czyli tekstury zmieniające się w czasie według jakiegoś programu. Ze względu na dużą złożoność obliczeniową i czasochłonne przesyłanie danych między pamięcią operacyjną a pamięcią podręczną karty graficznej, programy te wykonywane są przez procesory graficzne. Danymi programów są zwykle fragmenty innych tekstur. Samo wykonanie programu to renderowanie roboczej sceny nie na ekran, a właśnie na teksturę proceduralną.

Do przygotowania tekstury proceduralnej chmur wykorzystuje się szumy Perlina [10], ponieważ dobrze oddają one fraktalny charakter zjawisk w naturze. Najpierw obliczone wcześniej wartości funkcji generatora pseudolosowego nanosi się na teksturę szumu, a później z tak przygotowanej tekstury wycina się kilka mniejszych fragmentów różnej wielkości, które stanowić będą składowe różnej częstotliwości wynikowego szumu Perlina. Sześć przygotowanych fragmentów pokazano poniżej na rys. 4. Każdy z nich nazywa się oktawą, ponieważ stanowi on składową o częstotliwości dwa razy większej niż poprzedzający (analogicznie do oktaw w muzyce).



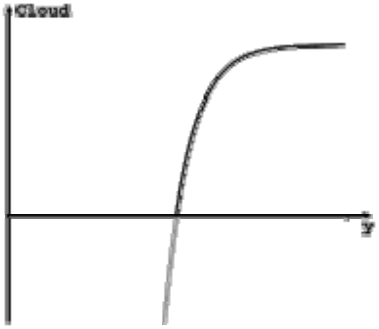
Rys. 4. Fragmenty tekstury szumu, tworzące oktawy
Fig. 4. Octaves – parts of the noise texture

Wszystkie składowe sumuje się razem z różnymi wagami, przy czym najczęściej wagi dobrane są odwrotnie proporcjonalnie do częstotliwości składowej. Wynik tej operacji jest pokazany na rys. 5.



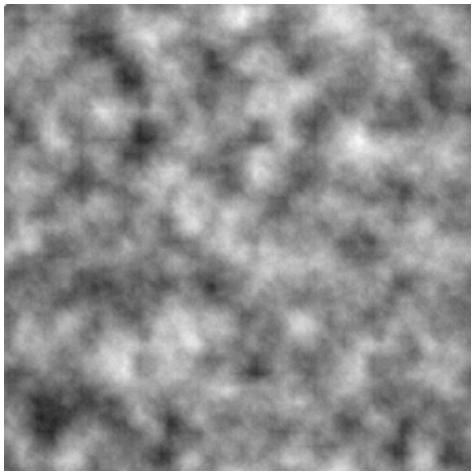
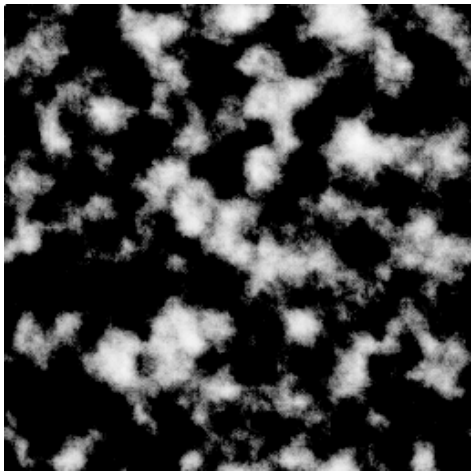
Rys. 5. Wynik dodania wszystkich oktaw szumu
Fig. 5. Result of adding all noise octaves

Do selekcji pojedynczych chmur wykorzystuje się prosty filtr, który wartości do pewnego poziomu obetnie do zera. Przykładowa funkcja filtrująca pokazana jest na rys. 6. Kształt funkcji może być różny, zależnie stopnia zachmurzenia nieba.



Rys. 6. Funkcja filtrująca
Fig. 6. Filter function

Operację filtrowania szumów Perlina zilustrowano na rys. 7. Pierwszy obraz zawiera dodane oktawy szumu, a drugi to wynik filtrowania z wykorzystaniem funkcji z rys. 6.

a)  b) 

Rys. 7. Filtrowanie szumów Perlina (a), aby wyizolować pojedyncze chmury (b)
Fig. 7. Filtering Perlin's noise (a) to isolate clouds (b)

Dynamiczne zmiany kształtów chmur, w tym ich powstawanie i zanikanie, realizowane są przez interpolację między bieżącą a poprzednią teksturą proceduralną. Przesuwanie chmur przez wiatr realizowane jest przez przesunięcie tekstury po powierzchni kopuły.

Niestety tekstury proceduralne mają jedną, podstawową wadę – są płaskie, co można zauważyć szczególnie w momencie przemieszczania się obserwatora. Nie ma możliwości symulacji lotu obserwatora przez chmury. Dwuwymiarowość powoduje też problemy związane z dodatkowym oświetleniem chmur podczas zachodów i wschodów słońca.

3.3. Układy przestrzenne

Najprostszym sposobem wyświetlenia trójwymiarowych chmur jest przygotowanie specjalnych modeli przez artystów-grafików w programach narzędziowych [1]. Obiekty składają się wtedy z wielu trójkątów i są bardzo obciążające dla systemu, jednak pozwalają na dokładne ich oświetlenie. Możliwy jest także swobodny ruch obserwatora.

Pojawianie się i zanikanie chmur realizowane jest przez stopniową zmianę stopnia przezroczystości fragmentów chmury. Elementy znajdujące się dalej od centrum obiektu są bardziej przezroczyste i stają się wyraźniejsze wraz z formowaniem się chmury.

Ze względu na dużą złożoność modeli nie jest możliwe wyświetlanie w czasie rzeczywistym wielu chmur jednocześnie. Aby pokryć chmurami całe niebo, stosuje się technikę impostorów [1]. Polega ona na tym, że w każdej klatce animacji rysowane są tylko chmury znajdujące się blisko obserwatora. Dalsze obiekty renderowane są na teksturę i wyświetlane w postaci dużej, płaskiej cząsteczki skierowanej zawsze w stronę obserwatora. Tekstura na cząsteczce zmienia się dopiero wtedy, gdy zmieni się układ chmur oraz wtedy, gdy błąd projekcji związany z przemieszczeniem obserwatora przekroczy jakąś ustaloną wartość maksymalną.

Rozwiązanie ma jednak poważne ograniczenia. Po pierwsze, nie da się narysować nieba bez znacznego wkładu artystów. Po drugie, zróżnicowanie chmur zależy od liczby przygotowanych modeli (niestety im więcej modeli, tym większe zapotrzebowanie na pamięć operacyjną). Dodatkowo tą metodą można symulować tylko niektóre typy chmur – nie nadaje się na przykład do symulacji nieba zimowego.

3.4. Algorytmy komórkowe

Wiele zalet posiada kolejna metoda tworzenia chmur, bazująca na automacie komórkowym. Wyjątkiem jest oczywiście zapotrzebowanie na pamięć operacyjną, które w tym przypadku jest nawet większe ze względu na rozmiary siatki komórek potrzebnej do pokrycia całego nieba.

Automat komórkowy to przestrzenna siatka, w której każda komórka w danej klatce symulacji znajduje się w pewnym stanie, zależnym od stanów jej sąsiadów oraz jej samej w klatce poprzedniej.

Metodę tę opisano w [8], proponując uproszczenie reguł zmiany stanu, aby można było zastąpić pojedynczą komórkę polem bitowym, a operacje przejścia przeprowadzać na wielu komórkach jednocześnie za pomocą algebry Boole'a.

Niestety nawet takie optymalizacje nie są wystarczające, aby symulacja odbywała się w czasie rzeczywistym. Metoda pozwala według autorów uzyskać realistyczne efekty przy częstotliwości kilku klatek animacji na minutę.

3.5. Cząsteczki

Do wizualizacji trójwymiarowych chmur wykorzystuje się również systemy cząsteczkowe. Realistyczne efekty daje metoda zaproponowana w [6], niemniej dotyczy ona wyłącznie statycznych chmur – aby wprowadzić dynamiczne zmiany zachmurzenia, algorytmy należy rozszerzyć o symulację pogody.

Na potrzeby platformy Flexible Reality Simulation tworzony jest specjalny komponent zajmujący się symulacją środowiska i pogody [7]. Wykorzystując dostarczane przez komponent dane, można wizualizować pogodę płynnie zmieniającą się w czasie, zgodną z rzeczywistymi zmianami zachodzącymi w atmosferze. Symulacja nie jest oczywiście dokładna i nie może mieć zastosowań prognostycznych. Jej celem nadrzędnym pozostaje realistyczny pod względem wizualnym efekt, jednak dodatkowo model uwzględnia specyficzne zjawiska występujące w skali globalnej, takie jak np. fronty atmosferyczne.

4. Wschody i zachody słońca

Najbardziej efektownym elementem symulacji są wschody i zachody słońca. Gdy słońce znajduje się nisko nad horyzontem, promienie świetlne mają do przebycia znacznie dłuższą drogę przez atmosferę. Dzięki rozpraszaniu Rayleigh,[6] paleta barw nieba i chmur jest bardzo szeroka.

Najprostszą metodą koloryzacji jest wyświetlenie na tle tarczy słońca półprzezroczystej cząsteczki, podobnie jak w przypadku flary słonecznej. Kolor cząsteczki jest funkcją wysokości słońca nad horyzontem oraz ewentualnie wilgotności powietrza, bowiem ilość pary

[6] Rozpraszanie Rayleigha to rozpraszanie światła na cząsteczkach o rozmiarach mniejszych od długości fali rozpraszanego światła.

w powietrzu bezpośrednio wpływa na stopień rozpraszania światła. Trudno jest wyznaczyć analityczny wzór tej funkcji. Często stosuje się krzywą łamaną, której punkty charakterystyczne ustalane są na podstawie obserwacji, a wartości pośrednie wyliczane są według interpolacji liniowej.

W przypadku chmur będących obiektami trójwymiarowymi, możliwe jest zastosowanie oświetlenia wspomaganego sprzętowo przez procesor graficzny. Analogicznie jest obliczany kolor światła. Dobre efekty można uzyskać uzależniając poziom oświetlenia chmury od jej pozycji na niebie – im bliżej słońca, tym mocniejsze oświetlenie.

5. Opady i efekty cząsteczkowe

Do wyrenderowania opadów atmosferycznych można wykorzystać emitery cząsteczek [15]. Każda kropla deszczu lub płatek śniegu reprezentowana jest przez pojedynczą cząsteczkę. Na ruch cząsteczki wpływ mają dwa czynniki: siła grawitacji oraz wiatr. Siłę grawitacji najczęściej zastępuje się wektorem prędkości, ponieważ ze względu na opory powietrza przyspieszenie kropli lub płatka jest pomijalne. Wiatr w najbliższym otoczeniu, które dla uproszczenia modelu rozszerza się do całej sceny trójwymiarowej, również można przedstawić jako wektor prędkości. Wektor prędkości cząsteczki jest sumą obu wspomnianych wektorów.

W przypadku płatka śniegu często zaburza się jego ruch prostoliniowy sinusoidą o losowej amplitudzie i częstotliwości, co symulować może charakterystyczne „bujanie się” płatka wywołane oporami powietrza.

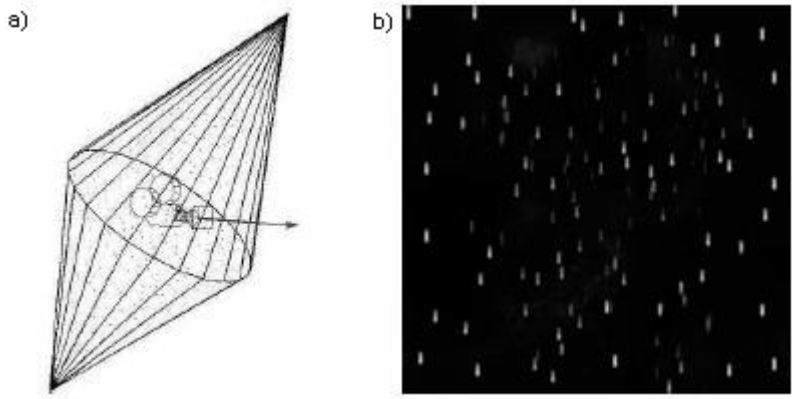
W przypadku intensywnych opadów, gdzie zachodzi potrzeba wyświetlenia wielu cząsteczek, systemy cząsteczkowe mogą wprowadzać zbyt duży narzut obciążeniowy. Świetnie sprawdza się za to metoda zaproponowana w [14]. Jedna lub więcej przezroczystych tekstur, na których umieszczone jest kilkadziesiąt kropel deszczu lub płatków śniegu, nanoszone są na dwa stożki złączone wspólnymi podstawami. Kamera zawsze znajduje się w samym środku stworzonej w ten sposób bryły.

Na rzeczywistym sprzęcie stożki aproksymuje się ostrosłupami o podstawie wielokąta foremnego, ponieważ nie można wyświetlić powierzchni krzywych (rys. 8).

Opadanie kropel lub płatków realizowane jest przez przesuwanie tekstury po bryle, czyli przez zmianę mapowania. Dodatkowo obraz przetwarzany jest przez shadery pikseli dla uzyskania efektu rozmycia ruchu (*motion blur*), czyli pozornego rozmycia konturów (deformacji kształtów) wywołanego szybkim ruchem.

Bryła przemieszcza się wraz z ruchem kamery, z którą związany jest wirtualny obserwator. Jej nachylenie zależy od wektora prędkości wiatru oraz od szybkości poruszania się

kamery. Im szybciej przemieszcza się kamera, tym bardziej górny wierzchołek bryły nachyla się w kierunku ruchu, co powoduje złudzenie, że krople deszczu opadają w stronę kamery.



Rys. 8. Podwójny ostrosłup (a), na który naniesiono teksturę deszczu(b)
Fig. 8. Double pyramid (a) with rain texture on it (b)

Trzeba jednak przyznać, że zaproponowana metoda nie daje najlepszych rezultatów w przypadku mało intensywnych opadów atmosferycznych. Konieczność stosowania programów pikseli uniemożliwia też renderowanie efektów na słabszych kartach graficznych.

Dobrym rozwiązaniem może być połączenie obu zaproponowanych metod. Efekt rozmycia *motion blur* można zastąpić rozciąganiem pojedynczych cząsteczek, gdzie stopień rozciągnięcia zależałby od prędkości przemieszczania się cząsteczki i intensywności opadów. Ruch cząsteczki odbywa się po trajektorii tworzących[7] dwóch stożków złączonych podstawami (należy zrezygnować z aproksymacji ostrosłupami). Wraz ze wzrostem intensywności opadów jedna cząsteczka reprezentowałaby kilka, a nawet kilkanaście kropel, co wyeliminowałoby problem nadmiernego obciążenia obliczeniowego przez redukcję liczby wyświetlanych cząsteczek.

6. Uwagi końcowe

Chociaż niebo stanowi tylko tło dla pozostałych elementów symulacji wirtualnej rzeczywistości, to jest bardzo istotnym elementem, dodającym renderowanej scenie realizmu. Ze względu na ogromną złożoność obliczeniową algorytmów realizujących naturalne procesy rządzące atmosferą, na potrzeby symulacji wirtualnej rzeczywistości korzysta się ze znacznych uproszczeń. Często wyświetlany efekt opiera się jedynie na złudzeniu optycznym, aby symulacja mogła odbywać się w czasie rzeczywistym. Szeroko stosowaną techniką są

[7] Tworząca – przeciwprostokątna trójkąta prostokątnego, którego obrót wokół jednej z przyprostokątnych spowodował zakreślenie bryły stożka.

emitery cząsteczek, głównie ze względu na ich elastyczność i niewielkie obciążenie obliczeniowe.

LITERATURA

1. Wang N.: Realistic and Fast Cloud Rendering. Journal of Graphics Tools, 2004.
2. Bell G.: Creating Backgrounds for 3D Games. Gamasutra, 1998.
3. Schlyter P.: How to compute planetary positions. www.stjarnhimlen.se/comp/ppcomp.html.
4. Erez E.: Interactive 3D Lighting in Sprites Rendering. Gamasutra, 2005.
5. Ditchburn R. W.: Light (2nd ed.), s. 582÷585. Blackie & Sons, London 1963.
6. Harris M., Lastra A.: Real-Time Cloud Rendering. The Eurographics Association and Blackwell Publishers, Vol. 20, No 3, 2001.
7. Grudziński J., Dębowski A.: Symulacja pogody w czasie rzeczywistym w środowisku FRS. Studia Informatica, Vol. 27, No. 1(66), Wyd. Pol. Śl., Gliwice 2006.
8. Dobashi Y., Kaneda K., Yamashita H., Okita T., Nishita T.: A Simple, Efficient Method for Realistic Animation of Clouds. Hiroshima City University, Hiroshima University, University of Tokio, 2000.
9. DeLoury M. -red: Perełki programowania gier – vademecum profesjonalisty. Helion, Gliwice 2000.
10. Elias H.: Perlin Noise. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm.
11. Elias H.: Cloud Cover. http://freespace.virgin.net/hugo.elias/models/m_clouds.htm.
12. Heinzlreiter P., Kurka G., Volkert J.: Real-time Visualization of Clouds. V. Skala -red, Journal of WSCG 2002, Vol. 10(3), 2002.
13. Trembilski A., BroßlerA.: Surface-Based Efficient Cloud Visualisation for Animation Applications. V. Skala -red, Journal of WSCG, Vol. 10, 2002.
14. Wang N., Wade B.: Rendering Falling Rain and Snow. SIGGRAPH, 2004.
15. Reeves W.: Particle systems – a technique for modeling a class of fuzzy objects. SIGGRAPH ’83, Detroit, Michigan. ACM, 1983.
16. Greek S., Cebenoyan C.: High Dynamic Range Rendering on the GeForce6800. NVIDIA's HDRR technical summary, 2004.
17. Unger J., Wrenninge M., Wanstrom F., Ollila M.: Real-Time Image Based Lighting in Software Using HDR. Computer graphics and interactive techniques in Australasia and South East Asia, 2003.
18. DiCarlo J. M., Wandell B. A.: Rendering high dynamic range images. SPIE: Image Sensors, 2000 (ftp://white.stanford.edu/users/brian/pdc/spiehdr.pdf).

19. Grudziński T., Mysłek T., Ross J.: Wykrywanie kolizji obiektów trójwymiarowych w środowisku FRS. Studia Informatica, Vol. 26, No. 4(65), Wyd. Pol. Śl., Gliwice 2006.
20. Grudziński T., Mysłek T., Ross J.: Zaawansowane techniki animacji trójwymiarowych modeli szkieletowych w środowisku FRS. Studia Informatica, Vol. 27, No. 1(66), Wyd. Pol. Śl., Gliwice 2006.

Recenzent: Dr hab. inż. Maria Pietruszka Prof. Pol. Łódzkiej

Wpłynęło do Redakcji 16 listopada 2006 r.

Abstract

The paper introduces the issues of real-time sky and atmosphere rendering on an animated 3-dimensional scene. Several sky dome rendering methods are proposed in section 2. The stars, nearby heavenly bodies and atmosphere rendering is discussed. Several techniques of clouds rendering are proposed in section 3. Section 4 describes the influence of sunrise and sunset on the entire scene. In addition, a few methods of rainfall rendering are discussed in section 5. Further conclusions are contained in the final section (section 6).

Adres

Adrian DĘBOWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, adrian.debowski@polsl.pl.
Jakub GRUDZIŃSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, jakub.grudzinski@polsl.pl.
Tomasz GRUDZIŃSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, tomasz.grudzinski@polsl.pl.