Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Inżynieria Systemów Informacyjnych

Rok Akademicki 2007/2008

PRACA MAGISTERSKA

Imię Studenta: *Dariusz Maciejewski*

Tytuł Pracy: *Gra komputerowa "Camera Fighter"*

Promotor:

prof. nzw. dr hab. Przemysław Rokita

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminacyjnej

Inżynieria Systemów Informacyjnych

Data Urodzenia: 3 grudnia 1983

Data Rozpoczęcia Studiów: 1 października 2006

Curriculum Vitae:

Urodziłem się 3 grudnia 1983 roku w Warszawie.

W latach 1990-1998 byłem uczniem Szkoły Podstawowej im. Wiktora Gomulickiego w Warszawie. W tym czasie zacząłem się interesować informatyką. Od roku 1998 do roku 2002 uczęszczałem do Liceum Ogólnokształcącego im. Adama Mickiewicza w Warszawie. W okresie tym stworzyłem swoje pierwsze gry i programy komputerowe. Po ukończeniu liceum, rozpocząłem studia w języku angielskim na kierunku Electrical and Computer Engineering na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Od końca roku 2005 pracuję jako programista ASP.NET i MS-SQL w firmie programistycznej Eracent. W roku 2006 obroniłem swoją pracę inżynierską pt. “Komponentowy System Tworzenia Grafów – AutoGrapher”. W roku tym rozpocząłem również uzupełniające studia magisterskie na kierunku Inżynierii Systemów Informatycznych na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej.

.....

Podpis Studenta

Egzamin Magisterski

Egzamin odbył się .....

Z wynikiem: .....

Ostateczny wynik studiów: .....

Sugestie i wnioski Magisterskiej Komisji Egzaminacyjnej.....

.....

.....

STRESZCZENIE

Słowa Kluczowe:

ABSTRACT

Title: “Camera Fighter” - a computer game.

Keywords:

Spis Treści:

1. WSTĘP

1.1. CEL PRACY

1.2. ROZWAŻANE METODY IMPLEMENTACJI

1.3. ZAWARTOŚĆ PRACY

2. NOTACJA I OKREŚLENIA UŻYTE W PRACY

3. ARCHITEKTURA APLIKACJI

3.1. WINDOWS I LINUX

3.2. STRUKTURA KATALOGÓW

3.3. BIBLIOTEKA MATEMATYCZNA

3.4. BIBLIOTEKA POMOCNICZA

3.4.1. Logowanie zdarzeń

3.4.2. Dostęp do dysku

3.4.3. Menadżer uchwytów

3.4.4. Singletony

3.5. BIBLIOTEKI MODELI

3.5.1. Biblioteka Lib3ds

3.5.2. Biblioteka Lib3dx

3.6. SZKIELET APLIKACJI

3.6.1. Aplikacja

3.6.2. Okno

3.6.3. Sceny

3.6.4. Obsługa zdarzeń klawiatury i myszki

3.7. SCENY

3.7.1. Menu

3.7.2. Rozgrywka

Wstęp

Świat

Logika

3.7.3. Konsola

3.7.4. Edytor modeli szkieletowych

Wstęp

Tworzenie szkieletu

Skórowanie modelu

Animacja szkieletowa

4. SILNIK GRAFICZNY

4.1. WSTĘP

4.2. MENADŻER TEKSTUR

4.3. MENADŻER CZCIONEK

4.4. ROZSZERZENIA OPENGL

4.5. RENDEROWANIE MODELI 3DX

4.5.1. Wstęp

4.5.2. Renderowanie jednoprzejściowe

4.5.3. Renderowanie do bufora głębi

4.5.4. Renderowanie oświetlenia otoczenia

4.5.5. Renderowanie oświetlenia rozproszonego i odblaskowego

4.6. RENDEROWANIE SCENY 3D

4.6.1. Renderowanie jednoprzejściowe

4.6.2. Renderowanie wieloprzejściowe

4.6.3. Renderowanie odroczone

4.7. PROGRAMOWANIE POD GPU

4.7.1. GLSlang

4.7.2. HLSL

5. SILNIK FIZYCZNY

5.1. DETEKCJA KOLIZJI

5.1.1. Wstęp

Detekcja kolizji jest niezbędnym elementem każdego symulatora fizyki. Nie mając informacji o zachodzących zderzeniach, symulator nie mógłby na nie zareagować. Obiekty poruszałyby się w 'świecie duchów', przenikając się nawzajem. Ruch byłby jedynie skutkiem zaszytych w silniku sił, takich jak np. siła ciążenia. Obserwowanie takiego świata nie było by ani pouczającym, ani zajmującym widowiskiem.

W wyniku detekcji kolizji silnik fizyczny otrzymuje informacje które umożliwiają symulację oddziaływań między ciałami. Dzięki tej wiedzy możemy przenieść się ze 'świata duchów' do świata materialnego. Detektory najczęściej wykrywają kolizje na jeden z dwóch sposobów: metodą a priori bądź a posteriori [WIKI_CD].

W metodzie a posteriori silnik porusza symulację o pewien mały krok do przodu, a następnie sprawdza, czy istnieją obiekty które przecinają się. W każdym kroku symulacji tworzona jest lista przecinających się obiektów. Na bazie tej listy poprawiane są pozycje i trajektorie ruchu kolidujących obiektów. Nazwa metody bierze się stąd, iż najczęściej symulator przeskoczy moment zaistnienia kolizji i dopiero po jej zaistnieniu podejmuje akcję.

Odmiennym podejściem cechuje się metoda a priori. W metodzie tej algorytm detekcji kolizji musi potrafić przewidzieć dokładne trajektorie obiektów. Wykorzystując tę wiedzę, może precyzyjnie określić moment zderzenia, dzięki czemu ciała nigdy się nie przetną. Momenty kolizji znane są jeszcze przed przemieszczeniem ciał. Niestety kosztem tej wiedzy jest duże skomplikowanie numeryczne algorytmu.

Przewagą metody a posteriori jest możliwość logicznego rozdzielenia algorytmu fizycznego od detektora kolizji. Detektor nie musi mieć wiedzy o zmiennych fizycznych opisujących ruch obiektów oraz o rodzajach materiałów z których są wykonane. Problemem tej metody jest jednak etap poprawiania niezgodnych z fizyką pozycji

obiektów, przez co metoda a priori cechuje dużo większą dokładnością i stabilnością symulacji.

Specjalnym przypadkiem do rozpatrzenia jest stan spoczynku. Jeżeli odległość oraz ruch dwóch obiektów względem siebie jest poniżej pewnego progu, ciała powinny przejść w stan spoczynku, z którego zostaną wyrwane dopiero pod działaniem nowej siły.

5.1.2. Algorytm detekcji kolizji

Oczywistym podejściem do detekcji kolizji jest sprawdzenie kolizji między wszystkim parami symulowanych obiektów. Ponieważ modele 3d składają się ze zbioru trójkątów, detekcja kolizji sprowadza się do sprawdzenia czy nie doszło do przecięcia trójkątów jednego z obiektów z trójkątami drugiego. Powstało wiele różnych algorytmów badających czy doszło do przecięcia dwóch trójkątów.

Jako bazę detektora wybrałem algorytm 'Fast Triangle-Triangle Intersection Test' autorstwa Oliviera Devillers oraz Philippe'a Guigue z INRIA [RR4488]. Algorytm ten w pierwszym kroku sprawdza czy trójkąty przecinają wzajemnie płaszczyzny na których leżą. Jest to niezbędny warunek by mogło dojść do kolizji. Jeżeli istnieje potencjalna kolizja, wystarczy przeprowadzić cztery kolejne testy by uzyskać ostateczne rozstrzygnięcie.

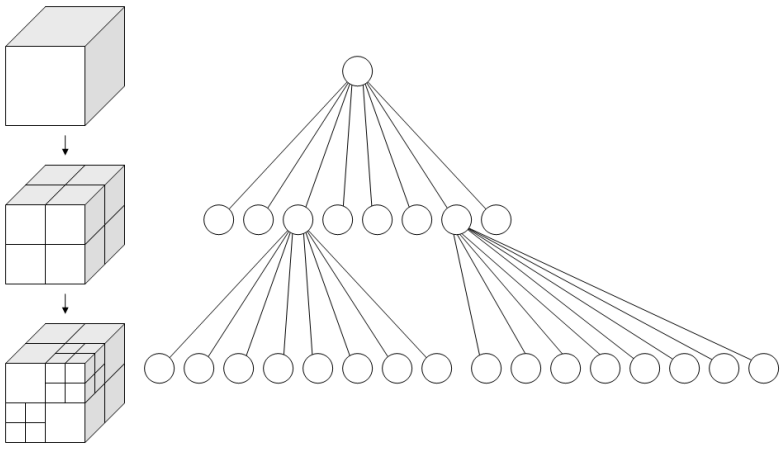
Algorytmy wykrywania przecinających się trójkątów są obecnie uproszczone do granic możliwości. Modele obiektów składają się najczęściej z co najmniej kilkuset trójkątów, a modele postaci mogą być zbudowane nawet z kilku milionów trójkątów. Jak łatwo wyliczyć, sprawdzenie kolizji zachodzącej między dwoma modelami może wymagać przeprowadzenia od dziesiątek tysięcy do milionów testów kolizji trójkątów. Grafika komputerowa umożliwia wykorzystanie sztuczek związanych z mapowaniem modelu, które umożliwiają ograniczenie ilości wierzchołków bez wyraźnej straty na jakości wizualnej. Kolejną techniką obniżającą ilość wierzchołków jest przygotowanie dla detektora kolizji modelu fizycznego, będącego uproszczoną wersją modelu pokazywanego na ekranie. Pozwala to na opisanie modelu postaci za pomocą jedynie

kilkudziesięciu do kilkuset wierzchołków. Nadal daje to jednak kilka tysięcy testów dla tylko jednej pary obiektów.

W celu ograniczenia ilości wymaganych testów kolizji trójkątów, stosuje się kilka technik umożliwiających przycinanie zbioru potencjalnych trójkątów mogących wejść w kolizję z innym obiektem, a także przycinanie zbioru obiektów które mogą się ze sobą przeciąć.

5.1.3. Hierarchie brył otaczających

Technika ta polega na podziale obiektu na hierarchiczne drzewo mniejszych elementów. Każdy węzeł drzewa zawiera informację o otaczającej go bryle. Bryłą tą najczęściej są sfery, sześciany bądź walce.



Rysunek 1: Hierarchiczny podział przestrzeni (na przykładzie Drzewa Ósemkowego).

Jeżeli bryła otaczająca dany węzeł przecina się z bryłą otaczającą węzeł należący do innego obiektu, to oznacza, że występuje potencjalna kolizja obiektów. Aby ją rozstrzygnąć, należy sprawdzić czy zachodzą kolizje między bryłami otaczającymi węzły niższego rzędu. Jeśli algorytm dojdzie do najniższego rzędu hierarchii brył otaczających, będzie musiał dokonać testu kolizji między trójkątami należącymi do tych najmniejszych węzłów w hierarchii. Dzięki zastosowaniu tej techniki już pierwszy test może wykluczyć kolizję między obiektami. Co więcej, wszystkie testy przeprowadzane w węzłach drzewa operują na prostych bryłach geometrycznych, pozwalających na

efektywne sprawdzenie czy zachodzi kolizja. Dopiero test najniższego rzędu wymaga badania kolizji między trójkątami, jednakże jest ich na tym poziomie tylko od kilku do kilkunastu dla każdego z węzłów.

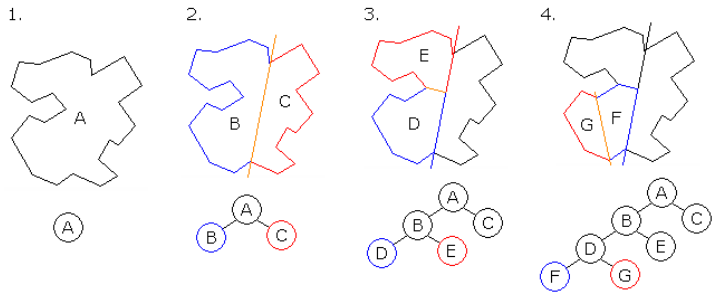
Projektując detektor, warto się również zastanowić, jak wielką precyzję wykrywania kolizji wymaga nasza symulacja. Prawdopodobnie możemy całkowicie zrezygnować z testowania kolizji między trójkątami, gdyż wystarczy nam przybliżenie zapewnione przez hierarchię brył otaczających.

[dopisać doświadczenia z implementacji]

5.1.4. Wybór obiektów mogących kolidować

Złożona scena 3d może składać się z bardzo dużej ilości obiektów. Obiekt poruszający się np. na scenie rozgrywającej się w mieszkaniu może wejść w kolizję z dużą ilością obiektów. Mogą to być zarówno drobne modele przedstawiające np. książki, ale też obiekty znaczniejszych rozmiarów przedstawiające meble i ściany. Sprawdzanie czy poruszający się obiekt wchodzi w kolizję z wszystkim modelami w scenie niepotrzebnie mnoży ilość przeprowadzanych testów, wpływając na znaczne obniżenie szybkości detekcji kolizji. Jeśli postać znajduje się w kuchni, to niema najmniejszego sensu sprawdzanie, czy przypadkiem nie zderzyła się z wanną mieszczącą się na drugim końcu mieszkania. W celu wyeliminowania takich zbędnych testów stosuje się techniki podziału przestrzeni.

Najbardziej uniwersalnym sposobem podziału przestrzeni jest zastosowanie drzewa BSP (Binary Space Partitioning - Binarny Podział Przestrzeni). Każdy węzeł takiego drzewa dzieli opisywaną przez siebie przestrzeń na dwie podprzestrzenie leżące po przeciwnych stronach pewnej hiperpłaszczyzny. Drzewo takie można zastosować do podziału przestrzeni o dowolnej wymiarowości. Pozwala ono na bardzo szybkie odrzucenie obiektów znajdujących się poza rozpatrywaną podprzestrzenią.



Rysunek 2: Drzewo Binarnego Podziału Przestrzeni.

Drzewo BSP dobrze nadaje się do opisu zamkniętych pomieszczeń, jednakże otwarte przestrzenie lepiej opisują drzewa ósemkowe (lub czwórkowe dla przestrzeni dwuwymiarowych). Drzewa takie polegają na otoczeniu sceny sześcianem, który jest następnie dzielony na osiem mniejszych sześcianów. Podział taki zagłębia się rekurencyjnie, aż do osiągnięcia ustalonego kryterium – zadanej głębokości lub minimalnej ilości obiektów w minimalnym sześcianie. Obiekt przypisywany jest do minimalnego sześcianu w którym może się pomieścić oraz wewnątrz którego znajduje się jego środek ciężkości. Wybierając obiekty które mogą wejść w kolizję z danych modelem, należy sprawdzić wszystkie obiekty znajdujące się w jego sześcianie (a więc i w jego sześcianach niższych rzędów) oraz sąsiednie sześciany. W praktyce sprawdzając sześciany sąsiadujące, wystarczy przejrzeć te leżące po stronie rosnących współrzędnych – kolizje z pozostałymi sąsiadami zostały sprawdzone podczas badania kandydatów dla obiektów należących do tamtych podprzestrzeni.

5.2. SYMULACJA FIZYKI

5.2.1. Opis dynamiczny

Posiadając wiedzę o siłach oddziałujących na obiekty oraz kolizjach do których doszło, można przejść do symulacji ruchu. Przykładowym podejściem może być otwarcie podręcznika do fizyki i zastosowanie przedstawionych w nim wzorów opisujących ruch.

Stosując opis dynamiczny na poziomie modelu jako całości, należałoby uwzględnić w algorytmie takie wielkości jak pozycja, prędkość, masa obiektu, a także

działające nań siły. Model taki jest w najprostszym przypadku ciałem sztywnym, nie można więc również zapomnieć o opisujących ruch obrotowy momencie bezwładności oraz momencie siły. Gdy zechcemy symulować ciała odkształcalne, takie jak materiały, powierzchnie elastyczne, bądź rośliny, nasz algorytm rozrośnie się jeszcze bardziej.

[dopisać doświadczenia z implementacji]

5.2.2. Całkowanie Verlet'a

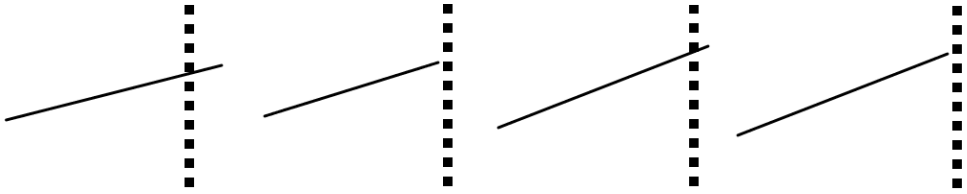
Alternatywne podejście do symulacji zachowania się ciał fizycznych zastosowano po raz pierwszy w grze *Hitman: Codename 47*. Rozwiązanie to przedstawił w swym artykule dyrektor do spraw badań i rozwoju w firmie IO Interactive, Thomas Jakobsen [JAKOB]. Polega ono na rozbiciu obiektu na zbiór punktów materialnych i rozpatrywaniu ruchu każdego punktu osobno. Ponadto, w celu poprawy stabilności algorytmu, zrezygnowano z przechowywania prędkości cząsteczki. W efekcie każda cząsteczka opisywana jest przez cztery wielkości fizyczne: pozycję, pozycję w poprzedniej klatce (wymagana do wyliczenia faktycznej prędkości), masę oraz wypadkowe przyśpieszenie cząsteczki. Cały opis ruchu pojedynczego punktu materialnego sprowadza się do następującego wzoru:

$$x_{t+1} = 2 \cdot x_t - x_{t-1} + a \cdot \Delta t^2$$

Podejście to nazywa się całkowaniem Verlet'a i jest szeroko wykorzystywane w symulacji dynamiki molekularnej. Prędkość ciała zaszyta jest w tym wzorze pod postacią różnicy położenia w w stałej jednostce czasu: $x_t - x_{t-1}$.

5.2.3. Nakładanie ograniczeń na punkty materialne

Za pomocą chmury niezależnych punktów materialnych, możemy opisać co najwyżej silnik cząsteczkowy. Jeśli jednak chcielibyśmy by nasze cząsteczki opisywały bardziej złożone ciała, musimy nałożyć na nie ograniczenia.



Rysunek 3: Kolejne kroki procesu spełniania ograniczeń.

W podejściu przedstawionym przez Jakobsena zbiór ograniczeń nałożonych na system spełniany jest poprzez relaksację. Proces ten postaram się opisać na przykładzie dwóch cząsteczek opisujących sztywny pręt. Odległość między tymi cząsteczkami musi być stała. Dodatkowo pręt nie powinien przebić żadnego innego obiektu. Na rysunku 3a widzimy sytuację w której to drugie ograniczenie zostało naruszone. Pręt przechodzi przez ścianę reprezentowaną przez przerywaną linię. W pojedynczym kroku symulacji, przez zadaną ilość iteracji, silnik będzie po kolei spełniał każde z ograniczeń. W ramach spełniania tego ograniczenia, punkt który przebił ścianę został rzutowany na poprawną pozycję. Spełnienie tego ograniczenia naruszyło jednak ograniczenie drugie – cząsteczki znalazły się zbyt blisko siebie (rys. 3b). Aby spełnić ten warunek, punkty muszą zostać rozsunięte – spowoduje to jednak ponowne przebicie ściany (rys. 3c). W kolejnych iteracjach relaksacji ograniczenia będą naruszane w coraz mniejszym stopniu, aż w końcu wszystkie zostaną spełnione.

Co interesujące, w pojedynczej klatce symulacji nie musimy doprowadzać do spełnienia wszystkich ograniczeń. Korygowanie pozycji cząsteczek może być kontynuowane w kolejnych klatkach.

5.2.4. Symulacja ciał odkształcalnych

Aby opisać zachowanie ciał odkształcalnych, takich jak ubrania bądź rośliny, wystarczy każdy wierzchołek siatki modelu symulować jako pojedynczy punkt materialny. Na punkty te powinny zostać nałożone warunki zachowania stałej odległości pokrywające się z krawędziami trójkątów siatki. Dla uzyskania ładnej animacji takich ciał wystarczy tylko pojedyncza iteracja relaksacji na klatkę symulacji.

5.2.5. Symulacja ciał sztywnych

Najprostszym sposobem na przedstawienie ciała sztywnego, jest zastosowanie takiego samego podejścia jak dla ciał odkształcalnych, lecz z nałożeniem ograniczeń stałej odległości na wszystkie pary wierzchołków. Wydajniejszym podejściem jest jednak opisanie całego ciała sztywnego z wykorzystaniem jedynie czterech punktów materialnych połączonych ograniczeniami stałej odległości. Ustawienie tych cząstek w konfiguracji np. czworościanu foremnego zapewni 6 stopni swobody, a więc dokładnie

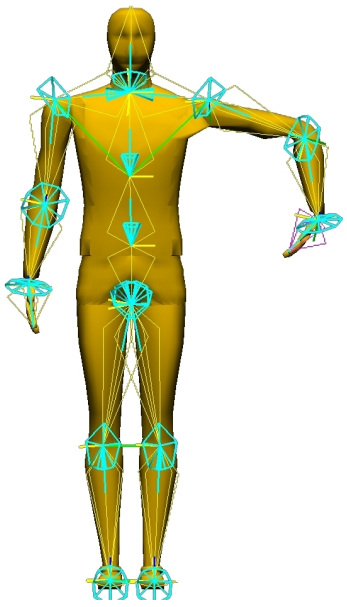
tyle, ile posiada ciało sztywne. W zależności od sposobu ustawienia cząsteczek, ciało będzie miało różny moment bezwładności.

Jedynym problemem jaki pozostaje do rozwiązania jest rozłożenie siły przyłożonej do pewnego punktu (np. punktu kolizji) na siły składowe działające na poszczególne cząsteczki. Dowolny punkt w przestrzeni może być przedstawiony jako liniowa interpolacja punktów materialnych które symulujemy. Korzystając z parametry tej interpolacji możemy dokonać operacji odwrotnej by proporcjonalnie rozłożyć siłę pomiędzy wszystkie cząsteczki.

[dopisać doświadczenia z implementacji]

5.2.6. Fizyka szkieletu

Silnik oparty na całkowaniu Verlet'a bardzo dobrze nadaje się również do symulowania fizyki ciał posiadających szkielet. Wystarczy stawy przedstawić jako punkty materialne, a kości jako warunki zachowania stałej odległości. Dodatkowo należy dodać ograniczenia kątowe, które uniemożliwią 'wykręcanie stawów'. Dobrym rozwiązaniem ograniczającym przecinanie się kości postaci (silnik fizyczny rozpatruje tylko kolizje z innymi modelami), jest dodanie warunków minimalnej odległości pomiędzy takimi stawami jak kolana i kostki stóp.



Rysunek 4: Szkielet z nałożonymi ograniczeniami kątowymi.

W przeciwieństwie do innych typów obiektów, cząsteczki reprezentujące stawy muszą być napędzane nie tylko samym procesem całkowania Verlet'a, ale również przez zewnętrzne źródła ruchu, takie jak animacje przygotowane przez artystów. Rozwiązaniem problemu może być algorytm, który będzie mieszał zewnętrzną animację z wynikiem działania całkowania. Waga każdego ze źródeł powinna być zmienna. Jeśli np. model zostanie uderzony z dużą siłą, to kontrolę powinien przejąć silnik Verlet'a. W pozostałych sytuacjach większą wagę powinno się przykładać do 'woli' postaci, czyli animacji narzuconych przez logikę gry.

[dopisać doświadczenia z implementacji]

6. PODSUMOWANIE

6.1. UWAGI I WNIOSKI

6.2. MOŻLIWOŚCI DALSZEGO ROZWOJU

7. BIBLIOGRAFIA

8. DODATEK – ZAWARTOŚĆ PŁYTY KOMPAKTOWEJ