Animacja szkieletowa z wykorzystaniem shaderów wierzchołków

1. Układ odniesienia, transformacja, drzewo sceny

Mając dany obiekt a w układzie odniesienia A jego stan opisujemy poprzez translację i rotację (ew. przesunięcie i obrót) określane również często mianem transformacji. Transformacja obiektu pozwala na określenie jego pozycji i obrotu w układzie odniesienia, w jakim się znajduje.

W grafice trójwymiarowej występują na ogół sceny o dużym stopniu komplikacji, gdzie wzajemne transformacje obiektów i układy odniesienia tworzą hierarchiczną strukturę zwaną drzewem sceny lub grafem sceny (ang. Scene Graph). Hierarchiczna struktura transformacji pozwala na sprawną zmianę transformacji całych grup obiektów opisanych w tym samym układzie odniesienia poprzez zmianę transformacji całego układu w jego układzie odniesienia.

Rozważmy przykładową scenę, aby nieco rozjaśnić wprowadzone pojęcia. Niech naszą sceną będzie pokój, na środku którego stoi stół, na którym siedzi kot. Widzimy tu trzy obiekty – pokój, stół i kota. Pokój będzie naszym globalnym układem odniesienia – w rozpatrywanym przypadku nie interesuje nas nic, co jest poza pokojem, rozważanie transformacji pokoju nie ma więc sensu. Niemniej stół i kot muszą być już opisane odpowiednią transformacją, żebyśmy byli w stanie przedstawić scenę na ekranie. Transformacja stołu jest podana względem pokoju, czyli względem globalnego układu odniesienia, co jest oczywiste. Nieoczywisty jest przypadek kota – jak to z kotami bywa.

Jeśli transformację kota przechowywać będziemy jako transformację względem pokoju, przesunięcie stołu nie przesunie kota, co na ogół jest zachowaniem niepożądanym. Kot śpi i nie obchodzi go, w którym miejscu pokoju stoi stół – on jest na środku stołu i koniec. Przechowując zatem transformację kota względem stołu (czyli w układzie odniesienia stołu) mamy gwarancję, że przy przesunięciu stołu kot przesunie się również pozostając na środku stołu, ale przemieszczając się wraz ze stołem w układzie odniesienia pokoju. Czy na pewno?

Aby wyświetlić model, musimy podać jego transformację globalną – w naszym przykładzie względem pokoju. Przechowując transformację kota względem stołu nie wiemy, gdzie znajduje się kot w układzie odniesienia pokoju, ponieważ do tej informacji potrzebujemy połączenia transformacji kota względem stołu oraz transformacji stołu względem pokoju. Widać tu wyraźnie, że występuje małe drzewo transformacji:

```
POKÓJ
  STÓŁ
    KOT
```

W takim przypadku transformację kota względem pokoju – a w ogólnym przypadku transformację globalną (względem globalnego układu odniesienia) – obliczamy poprzez mnożenie kolejnych transformacji względnych aż do korzenia drzewa, w naszym przykładzie będzie to pomnożenie lokalnej transformacji kota przez lokalną transformację stołu.

Zrozumienie zasad lokalnej i globalnej transformacji będzie bardzo pomocne dla dobrego zrozumienia zasad poruszania modelem podczas animacji szkieletowej.

2. Co to jest animacja?

W grafice trójwymiarowej mianem animacji określamy ruch obiektów będących na scenie, bądź względem sceny, bądź we własnym układzie odniesienia. Pierwszy przypadek – czyli ruch obiektów względem sceny – jest animacją sceny; drugi przypadek – ruch pojedynczego obiektu we własnym układzie odniesienia – jest animacją obiektu. Pod potocznym sformułowaniem "ruch" rozumiemy w tym przypadku zmianę transformacji, czyli translacji i rotacji obiektu lub jego części.

Rozważmy różnicę między wymienionymi wariantami animacji na przykładzie. Jeśli wyobrazimy sobie scenę złożoną z kilku obiektów, np. istot ludzkich, każda taka istota będzie miała określoną transformację. Transformacja ta będzie określać położenie istoty na scenie, np. to, że postać A stoi 2 metry przed postacią B. Zmiana transformacji postaci – następująca np. w wyniku przesuwania postaci – jest animacją sceny. Niemniej już ruch postaci, np. marsz, jest animacją postaci. Marsz nie zmienia bowiem transformacji postaci na scenie, a jedynie wzajemną pozycję części modelu postaci w jej lokalnym układzie odniesienia.

Aby uzyskać realistyczny efekt wizualny należy w takiej sytuacji oczywiście połączyć animację postaci z animacją sceny i podczas odgrywania animacji marszu przesuwać postać tak, aby nie było wrażenia marszu w miejscu. Nie zmienia to jednak faktu, że te dwie czynności – animacja postaci oraz jej przesuwanie po scenie, czyli animowanie sceny – są całkiem oddzielnymi czynnościami i powinny być rozważane oddzielnie.

Można zatem przyjąć definicję, że **animacja obiektu jest to zmiana kształtu obiektu w czasie**. Dzięki temu definicję tę będzie można zastosować zarówno dla sceny, jak i dla pojedynczego modelu.

3. Budowa modelu trójwymiarowego

Trójwymiarowy model składa się z jednej lub więcej siatek zbudowanych z wierzchołków. Wierzchołek jest po prostu punktem w trójwymiarowej przestrzeni. Wierzchołki modeli są połączone w trójkąty, najprostsze płaskie figury geometryczne. Chociaż trójkąty nie są jedynymi możliwymi do narysowania prymitywami, ich renderowanie jest zdecydowanie najszybsze i w aplikacjach czasu rzeczywistego bardzo rzadko używa się innych figur. Połączone ze sobą trójkąty tworzą siatkę, najważniejszą część modelu. Chociaż model może składać się z więcej niż jednej siatki, na potrzeby niniejszej instrukcji będziemy traktować model jako pojedynczą siatkę.

Siatka modelu reprezentuje jego kształ w przestrzeni, ponieważ przechowuje geometryczny układ wierzchołków, które go tworzą. Niemniej za to, jak model wygląda, odpowiada przede wszystkim materiał, jakim pokryta jest siatka. Materiał jest zestawem parametrów opisujących powierzchnię renderowanej siatki, a także sposób jej renderowania, np. często materiał przechowuje kolor siatki lub jej teksturę(y). Można przyjąć, że każda siatka musi być pokryta materiałem, a dla uproszczenia załóżmy również (co notabene jest prawdziwe w wielu bibliotekach graficznych), że cała siatka pokryta jest dokładnie jednym materiałem.

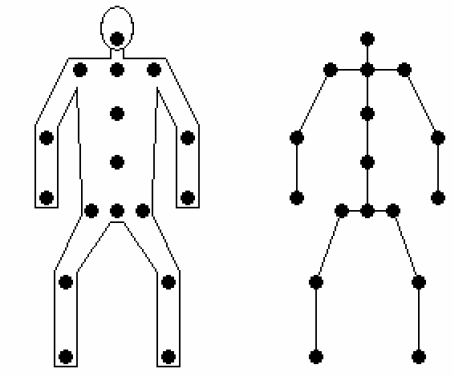
Mając tę podstawową wiedzę możemy wprowadzić dalsze terminy związane z wierzchołkami. Chociaż pozycja wierzchołka jest najważniejszą częścią danych wierzchołka, rzadko spotyka się sytuację, w której renderuje się model bez dodatkowych danych przesyłanych z każdym wierzchołkiem. Jednym z najbardziej typowych zestawów danych przesyłanych do procesora graficznego z każdym wierzchołkiem jest pozycja, normalna i współrzędne tekstur. Normalna wierzchołka jest wektorem normalnym do powierzchni siatki w miejscu, w którym jest wierzchołek. Normalna używana jest na ogół do obliczeń związanych z oświetleniem i poprawnie użyta może znacząco poprawić wygląd renderowanej sceny. Współrzędne tekstur to para liczb określających, który fragment tekstury zostanie nałożony na aktualnie renderowany trójkąt. Chociaż wraz z wierzchołkiem może zostać przesłanych dużo innych danych, nie będzie to tematem ćwiczenia i na potrzeby tej instrukcji wystarczy wiedzieć, że istnieje taka możliwość.

Wszystkie dane wierzchołków są zawsze przechowywane względem lokalnego układu odniesienia. Kiedy model jest renderowany, każdy wierzchołek jest transformowany aktualną macierzą transformacji modelu, która może przesunąć, obrócić czy przeskalować model, aby umieścić go w odpowiedniej pozycji w odpowiednim miejscu sceny. Zmieniając macierz transformacji modelu między renderowanymi klatkami możemy przesuwać czy obracać model, ale nie możemy zmienić jego kształtu. Zmiana kształtu modelu, czasem określana jakom deformacja siatki modelu, określana będzie **animacją**. Animacja **szkieletowa** jest przykładem techniki animacji dającej bardzo duże możliwości.

4. Model, szkielet modelu i animacja szkieletowa

Model trójwymiarowy jest zestawem jednej lub więcej siatek wierzchołków tworzących logiczną całość. Siatki te pokryte są materiałami, które w typowych przypadkach są po prostu teksturami, tudzież zestawami kilku tekstur o różnym znaczeniu (np. mapa nierówności, mapa połysku). Aby animować model w sposób efektywny i efektowny, potrzebna jest uniwersalna metoda deformowania jego siatek, czyli zmiany położenia wierzchołków modelu.

Szkielet jest pewnym zestawem wierzchołków nie wchodzących w skład modelu i nie mających sensu wizualnego. Wierzchołki te są uporządkowane hierarchicznie tworząc strukturę drzewiastą, a ich ułożenie w przestrzeni sceny powinno w przybliżeniu oddawać kształt modelu, jak pokazano na rysunku.



Rys. 1. Kości modelu humanoida

W odróżnieniu do rzeczywistości, gdzie kości szkieletu można uprościć odcinkami, w modelu szkieletowym za kości uznaje się wierzchołki wchodzące w skład szkieletu. Można to przedstawić jako analogię do modelu biologicznego, gdzie kość modelu wyznacza koniec kości biologicznej, natomiast początek kości biologicznej wyznaczany jest przez kość-rodzica. Korzeń drzewa jest elementem bazowym wszystkich kości, może nim zostać każda kość, jednak ze względów optymalizacyjnych wskazane jest, by długość ścieżki od korzenia do poszczególnych liści nie zmieniała się.

Celem zastosowania porządku hierarchicznego kości jest uzyskanie zależności, przy której zmiana transformacji kości powoduje tę samą zmianę u wszystkich kości-potomków danej kości. Jest to mechanizm analogiczny do omawianego w punkcie pierwszym mechanizmu drzewa transformacji obiektów, przy czym tutaj miejsce obiektów zajmują kości szkieletu.

Dla każdej kości przechowywana jest transformacja kości w układzie współrzędnych wyznaczonym przez rodzica, tzn. rotacja rodzica wyznacza układ współrzędnych, względem środka którego opisane są transformacje dzieci, natomiast translacja wyznacza przesunięcie układu odniesienia dzieci danej kości względem środka układu, w którym kość jest zdefiniowana. W ten sposób można pominąć aktualizowanie transformacji kości-dzieci podczas zmiany transformacji kości-przodka. W przypadku kości korzeniowej, czyli kości nie posiadającej rodzica, jej transformacja jest zdefiniowana w układzie odniesienia całego modelu.

Poprzez animację szkieletową rozumiemy ruch jednej lub więcej kości modelu, czyli zmianę jej transformacji. Może być to zarówno ruch oparty na wcześniej przygotowanych przez grafika-animatora ścieżkach, jak i ruch wygenerowany programistycznie za pomocą mechanizmów opartych o kinematykę prostą i odwrotną.

5. Zasady przekształceń wierzchołków przy animacji szkieletowej

Ogromną zaletą animacji szkieletowej jest fakt, że animując niewielki zestaw punktów – kości poruszamy całym, nieraz bardzo złożonym modelem. Relacje między wierzchołkami modelu a jego szkieletem są stałe i zaprojektowane przez grafika, po zaimplementowaniu ich obsługi np. w bibliotece obsługującej modele nie musimy się o nie martwić – każda zmiana kształtu szkieletu modelu będzie przełożona na kształt samego modelu i poprawnie wyświetlona na ekranie. Niemniej należy wiedzieć, jak zaimplementować animację samego modelu, ponieważ jest to element bardzo istotny i w dużym stopniu wpływający na wydajność całego silnika.

Pozycja każdego wierzchołka modelu zależy od co najmniej jednej kości szkieletu modelu. W zależności od realizacji przekształceń wierzchołków może występować górna granica liczby kości wpływających na pozycję wierzchołka, w przypadku zastosowania shaderów wierzchołków tą granicą są cztery kości. W ramach kości wpływających na pozycję wierzchołka mogą być zdefiniowane wagi wpływu, suma tych wag dla każdego wierzchołka musi być równa 1. Przykładowo na wierzchołki przedramienia może wpływać kość łokcia z wagą 0.9 i kość nadgarstka z wagą 0.1 (nie jest to może przykład dobrze oddający rzeczywistość). W znakomitej większości typowych przypadków na wierzchołek będzie wpływać tylko jedna kość z wagą 1, jednak w niektórych miejscach modelu będą to zależności bardziej skomplikowane.

Jak zatem przekształcić wierzchołki modelu aby odpowiadały aktualnej pozie szkieletu? Algorytm jest bardzo prosty:

- Aktualizujemy stan szkieletu.
- Dla każdego wierzchołka:
  - Zerujemy jego aktualną pozycję.
  - Dla każdej kości, która modyfikuje wierzchołek:
    - Transformujemy bazową pozycję wierzchołka transformacją bezwzględną kości (transformacją kości w układzie odniesienia modelu).
    - Mnożymy uzyskaną pozycję przez wagę modyfikującej kości.
    - Dodajemy uzyskaną pozycję do aktualnej pozycji wierzchołka.

Takie przekształcenia mogą być realizowane na co najmniej dwa sposoby – programowo przy wykorzystaniu standardowych mechanizmów języka C++ lub Delphi, tudzież wstawek asemblerowych wykorzystujących np. SSE czy SSE2 oraz sprzętowo przy wykorzystaniu shaderów wierzchołków. Ta druga metoda będzie przedmiotem laboratorium i jest omówiona w dalszych częściach instrukcji, niemniej na potrzeby tej instrukcji przytoczymy tutaj implementację powyższego algorytmu w pseudokodzie opartym na Delphi:

```
// dla każdego wierzchołka
for i := 0 to VertexCount - 1 do
begin
  // pobieramy wskaźnik na wierzchołek źródłowy z bufora wejściowego
  SrcVert := ...
  // pobieramy wskaźnik na wierzchołek docelowy z bufora wyjściowego
  // zakładamy, że bufor wyjściowy jest wyzerowany
  DstVert := ...

  // dla każdej kości modyfikującej dany wierzchołek
  for j := 0 to BoneCount[i] - 1 do
  begin
    // pobieramy macierz transformacji kości
    BoneTransformation := Matrices[VertexData[i].BoneIndices[j]];
    // pobieramy wagę wpływu
    Weight := VertexData[i].BoneWeights[j];

    // przekształcamy wierzchołek źródłowy
    Vec3Transform(Tmp, SrcVert^, Transformation);
    // skalujemy rezultat wagą wpływu kości
    Vec3Scale(Tmp, Tmp, Weight);
    // dodajemy otrzymany rezultat do wynikowego wierzchołka
    Vec3Add(DstVert^, DstVert^, Tmp);
  end;
end;
```

6. Shadery wierzchołków

Shadery wierzchołków (ang. vertex shaders, VS) to niewielkie programy pisane w specjalnym języku programowania wykonywane bezpośrednio na karcie graficznej podczas sprzętowego przetwarzania renderowanych danych. Można przyjąć, że VS to stan renderowania, przez który przechodzi każdy renderowany wierzchołek. Do VS można podać odpowiednie parametry, które, wraz z danymi wierzchołka, pozwolą na

zmianę efektywnej pozycji wierzchołka – tudzież innych jego elementów – tuż przed wyprowadzeniem ich na ekran. Należy tutaj zaznaczyć, że pod pojęciem wierzchołek kryć się może dużo więcej danych niż tylko pozycja, typowo jest to również normalna, współrzędne tekstur czy kolor. W naszym przypadku danych wierzchołka będzie jeszcze więcej.

Dzięki zastosowaniu VS możliwe do realizacji stają się ciekawe efekty graficzne, które zostaną pokazane na osobnych zajęciach, należy również zwrócić uwagę, że VS są znacznie wydajniejsze niż programowe obliczenia dużych ilości przekształceń macierzowych.

Można wyobrazić sobie vertex shader jako czarne pudełko, do którego podajemy pewną strukturę wejściową i które na wyjściu podaje nam strukturę wyjściową, która teoretycznie może być całkiem różna od wejściowej. Parametry vertex shadera są mapowane na stałe dostępne w kodzie programu shadera i mogą zostać użyte przy obliczeniach wewnątrz „skrzynki". Poniżej prezentujemy prostu kod shadera wierzchołków, który używa parametry „g_wind" żeby przesunąć o wektor wiatru wszystkie renderowane wierzchołki. Wszystkie macierze użyte do transformacji wierzchołków również są przesyłane do programu shadera jako parametry. Te transormacje to dość typowe operacje wykonywane na ogół w każdym shaderze i nie należy się nimi przejmować. Kilka słów odnośnie tych macierzy zostanie powiedziane podczas wprowadzenia do laboratorium.

```
// Prosty vertex shader dodający wektor wiatru g_wind do wierzchołków
void AddWind_VS(
  in  float3 i_position : POSITION,
  in  float2 i_uv       : TEXCOORD0,
  out float4 o_position : POSITION,
  out float2 o_uv       : TEXCOORD0)
{
  // transformacja wierzchołka z lokalnej przestrzeni do przestrzeni świata
  float4 world_position = mul(i_position, g_local_to_world);

  // dodanie wektora wiatru do wierzchołka
  world_position += float4(g_wind, 0);

  // transformacja z przestrzeni świata do przestrzeni widoku
  float4 view_position = mul(world_position, g_world_to_view);

  // transformacja z przestrzeni widoku na przestrzeń ekranu
  o_position = mul(view_position, g_view_to_screen);

  // kopiowanie współrzędnych tekstur
  o_uv = i_uv;
}
```

7. Zastosowanie shaderów wierzchołków do animacji szkieletowej

Wyobraźmy sobie, że algorytm opisany w punkcie 4 realizowany jest przez VS. Algorytm ten wymaga dwóch zestawów danych: transformacji kości szkieletu oraz tablicy zależności wierzchołek-kość. Można zauważyć, że transformacje kości są stałe dla całego modelu, a zależności wierzchołek-kość są charakterystyczne dla danego wierzchołka. Zgodnie z tym, co napisano wcześniej, tablica transformacji

kości będzie więc parametrem VS, a zależności wierzchołka będą dodatkowymi danymi generowanymi dla każdego wierzchołka. Dzięki temu w VS będzie można zrealizować przekształcenie każdego wierzchołka zgodnie z algorytmem podanym w punkcie 4.

8. Środowisko laboratoryjne i zadanie

Podczas laboratorium studenci będą musieli napisać vertex shader który wykona potrzebne obliczenia aby poprawnie wyrenderować animowany szkieletowo model. Shader będzie pisany w języku HLSL szeroko opisanym w materiałach na internecie. Niektóre części języka zostaną również krótko zaprezentowane na laboratorium.

Większość shadera będzie napisana wcześniej aby umożliwić skończenie zadania w czasie wyznaczonym na laboratorium. Jeśli zadanie zostanie wykonane, animowany model będzie poprawnie przechadzał się po niewielkim, trójwymiarowym terenie. Ćwiczenie jest częścią nieco większego projektu, w wyniku którego, po wykonaniu ćwiczeń 9-12, studenci będą mogli uruchomić prostą trójwymiarową grę. Cały projekt zostanie napisany wcześniej, a na ćwiczeniach będzie trzeba jedynie uzupełnić projekt o cztery brakujące fragmenty, z których pierwszym jest właśnie shader animacji szkieletowej. Poniżej prezentujemy szablon shadera z zaznaczeniem fragmentu, który będzie trzeba uzupełnić na laboratorium. Dokładniejsze instrukcję zostaną udzielone przed samym ćwiczeniem.

```
// tablica macierzy transformacji kości
float4x4 g_bones[10] < string FRS_Name = "BoneMatrices"; >;

void BoneAnimDirectionalLight_VS(
  in  float3 i_position : POSITION,
  in  float3 i_normal   : NORMAL,
  in  float2 i_uv       : TEXCOORD0,
  in  int4   i_indices  : BLENDINDICES,
  in  float4 i_weights  : BLENDWEIGHT,
  out float4 o_position : POSITION,
  out float4 o_color    : COLOR0,
  out float2 o_uv       : TEXCOORD0)
{
  // przed wszystkimi transformacjami należy przetransformować pozycję
  // i normalną wierzchołka
  float4 pos;
  float4 norm;

  // ...TUTAJ KOD STUDENTA...

  // dalsza część shadera nie jest przedmiotem ćwiczenia i nie musi być
  // analizowana

  // przestrzeń lokalna modelu -> przestrzeń świata
  float4 world_position = mul(pos, g_local_to_world);

  // obracamy normalną
  float3 normal = mul(norm, g_local_to_world);

  // przestrzeń świata -> przestrzeń widoku
  float4 view_position = mul(world_position, g_world_to_view);
```

```
  // przestrzeń widoku -> przestrzeń ekranu
  o_position = mul(view_position, g_view_to_screen);

  // kopiujemy współrzędne tekstur
  o_uv = i_uv;

  // obliczamy wyjściowy kolor wierzchołka
  o_color = saturate(dot(normal, -g_light_direction));
}
```