Politechnika Łódzka

Instytut Informatyki

Instrukcja do laboratorium

Fotorealistyczna Grafika Komputerowa

Laboratorium III

Przecięcie z trójkątem, parsowanie pliku OBJ

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

dr inż. Piotr Napieralski
mgr inż. Krzysztof Guzek

Łódź, 28.02.2012

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: napieral@ics.p.lodz.pl

1. Wstęp

Podstawowe, zaimplementowane dotychczas prymitywy, takie jak sfera i płaszczyzna pozwoliły na przetestowanie algorytmów raycastingu i antyaliasingu. W przypadku bardziej złożonych geometrycznie scen zachodzi jednak konieczność reprezentacji obiektów za pomocą siatek składających się z trójkątów. W tym celu konieczne jest rozbudowanie silnika renderującego o nową klasę zawierającą definicję trójkąta. W grafice komputerowej istnieje wiele algorytmów umożliwiających obliczenie przecięcia promienia z trójkątem. W niniejszej instrukcji przedstawione zostały jedynie dwie najpopularniejsze techniki. Aby umożliwić swobodny import geometrii do sceny, najwygodniej zaimplementować prosty parser jednego z formatów opisu scen 3D jakim jest OBJ.

2. Przecięcie promienia z trójkątem

Trójkąt możemy zdefiniować przez trzy wierzchołki A, B, C oraz wektor normalny określający jego zwrot. Wektor normalny trójkąta jest równoważny wektorowi normalnemu płaszczyzny w której zawiera się trójkąt.

Szukanie punktu przecięcia promienia z trójkątem jest bardzo często spotykanym problemem w grafice 3D, bowiem jest to zadanie tożsame do operacji przy precyzyjnej kolizji.

2.1 Metoda bazująca na kątach

Wpierw sprawdzane jest przecięcie promienia z płaszczyzną zawierającą trójkąt za pomocą algorytmu zaprezentowanego w pierwszej instrukcji. W wypadku znalezienia punktu przecięcia należy następnie sprawdzić czy ów punkt płaszczyzny leży wewnątrz trójkąta. Najczęściej proponowanym rozwiązaniem jest testowanie dla każdej z prostych AB BC CA czy punkt przecięcia P znajduje się po tej samej stronie prostej, co odpowiednio punkt C A B. Alternatywnym sposobem jest algorytm polegający na:

- znalezieniu wektorów od punktu przecięcia do każdego z wierzchołków, czyli PA PB PC
- sprawdzeniu czy kąt miedzy wektorami PA-PB, PB-PC, PC-PA jest mniejszy od Π
- jeśli wszystkie kąty są mniejsze to punkt P leży wewnątrz trójkąta

Aby zoptymalizować obliczenia zamiast sprawdzania kąta pomiędzy poszczególnymi parami wektorów wyznaczamy ich wektor z iloczynu wektorowego zawierający sinus kąta i zamiast porównania z kątem Π sprawdzamy jego iloczyn skalarny z normalną trójkąta. Wektor normalny do trójkąta jest niczym innym jak iloczynem wektorowym dwóch nie równoległych wektorów z płaszczyzny. Używając zapisu punktów z trójkąta, używając ich zgodnie kierunkiem wskazówek zegara, wektor normalny będzie miał następującą postać:

$$n = (b - a) \times (c - a).$$

Wadą algorytmu jest dokładność, ponieważ testowanie iloczynu skalarnego z liczbą 0 może dawać błędy na krawędziach trójkąta. Dlatego rozwiązaniem jest testowanie z definicją MINUS_ZERO w celu zaokrąglenia krawędzi trójkąta:

```
#define PLUS_ZERO 0.00001
#define MINUS_ZERO -0.0001
```

Przykładowy kod w C++ obliczający przecięcie promienia z trójkątem na podstawie powyższego algorytmu:

```
CGeometry* CTriangle::CountCrossPoint(CRay* ray,CVector3& crosspoint,double
&odleglosc)
{

    // sprawdzam czy promien w ogole przecina plaszczyzne na ktorej lezy trojkat

    // funkcja MyPlane.CountCrossPoint powinna wyznaczac, o ile istnieje, punkt
    // przeciecia promienia z plaszczyzna

    if(!this->MyPlane.CountCrossPoint(ray,crosspoint,odleglosc))
    {
            return 0;
    }

    // jesli przecina plaszczyzne, sprawdzam czy obliczony punkt przeciecia lezy
    // wewnatrz czy na zewnatrz trojkata

    else
    {
            CVector3 fa,fb,fc;
            CVector3 x;

            // obliczam wektory od punktu przeciecia do wierzcholkow trojkata
            fa=this->a-crosspoint;
            fb=this->b-crosspoint;
            fc=this->c-crosspoint;

            // obliczam iloczyn wektorowy pierwszej pary
            x.Cross(&fa,&fb);

            // jesli kat (fa,fb) jest wiekszy niz 180, czyli crosspoint lezy na
            // zewnatrz trojkata, wowczas uzyskany z iloczynu skalarnego wektor
            // ma zwrot przeciwny do wektora normalnego plaszczyzny trojkata
            // (kat miedzy nimi jest 180) czyli ich iloczyn skalarny bedzie
            // ujemny

            if(x.Dot(&this->MyPlane.ABC)<MINUS_ZERO)
            {
                  return 0;
            }
            else
            {
```

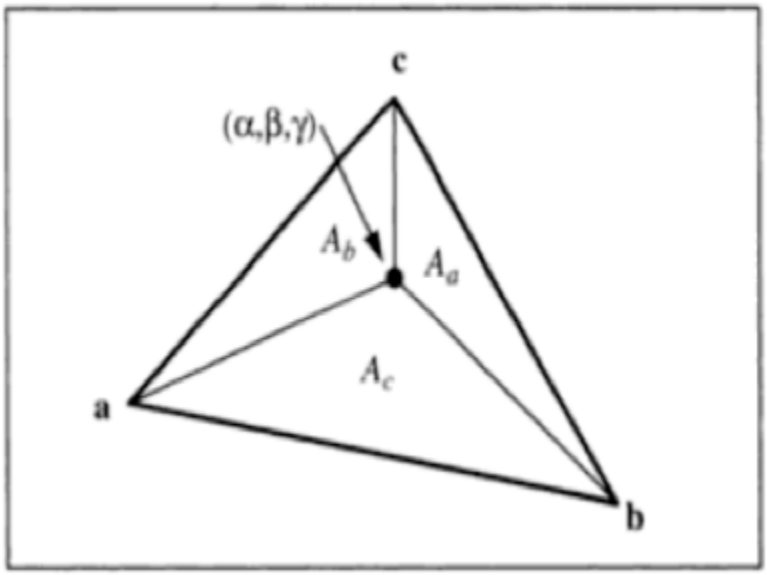
```
                        // sprawdzam dla nastepnej pary wektorow
                        x.Cross(&fb,&fc);
                        if(x.Dot(&this->MyPlane.ABC)<MINUS_ZERO)
                        {
                              return 0;
                        }
                        else
                        {

                              // sprawdzam dla trzeciej pary wektorow
                              x.Cross(&fc,&fa);
                              if(x.Dot(&this->MyPlane.ABC)<MINUS_ZERO)
                              {
                                    return 0;
                              }
                              // jesli wszystkie trzy wieksze od zera
                              // punkt przeciecia lezy wewnatrz trojkata
                              else
                              {
                                    return this;
                              }
                        }
                  }
            }
};
```

2.2 Metoda bazująca na współrzędnych barycentrycznych

Trójkąt zdefiniowany jest za pomocą trzech wierzchołków a, b oraz c. Jeśli nie są one współliniowe, to powinny leżeć w jednej płaszczyźnie. Do opisania tego rodzaju płaszczyzny można posłużyć się współrzędnymi barycentrycznymi:

$$p(\alpha,\beta,\gamma) = \alpha a + \beta b + \gamma c$$



Rys.1 Współrzędne barycentryczne trójkąta.

Jednym ze sposobów obliczania współrzędnych barycentrycznych jest znalezienie trzech obszarów A_a, A_b, A_c trójkątów z podobszarów (Rys.1). Współrzędne posiadają wówczas następujące właściwości:

$$\alpha = A_a / A,$$
$$\beta = A_b / A,$$
$$\gamma = A_c / A,$$

gdzie A jest powierzchnią trójkąta. Zasady te będą ważne również dla punktów znajdujących się poza obszarem trójkąta. Zauważmy, że:

$$\alpha + \beta + \gamma = 1.$$

Punkt p leży wewnątrz trójkąta wtedy i tylko wtedy gdy:

$$0 < \alpha < 1,$$
$$0 < \beta < 1,$$
$$0 < \gamma < 1.$$

Jeśli któraś ze współrzędnych wynosi zero oznacza to że znajdujemy się na krawędzi. Jeśli dwie współrzędne wynoszą zero oznacza to że znajdujemy się na wierzchołku.

Możemy dokonać podstawienia:

$$\alpha = 1 - \beta - \gamma$$

do równania:

$$p(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c$$

otrzymując:

$$p(\beta, \gamma) = a + \beta (b - a) + \gamma (c - a).$$

Reasumując α i β parametryzują nieortogonalny układ współrzędnych na płaszczyźnie. Promień p(t) = o + td przecina płaszczyznę tam gdzie:

$$o + td = a + \beta (b - a) + \gamma (c - a)$$

Punkt ten będzie wewnątrz trójkąta wtedy i tylko wtedy gdy:

$$\beta > 0, \gamma > 0 \quad \text{oraz} \quad \beta + \gamma < 1$$

Przykład widoczny na Rys.2 ilustruje sytuację, gdzie promień przecina płaszczyznę w punkcie:

$$(\beta, \gamma) = (1.2, 0.8)$$

Widzimy, że punkt ten nie znajduje się wewnątrz trójkąta, wskazuje na to suma β i γ która wynosi 2.

By rozwiązać równanie ze względu na parametr t, β iγ , użyjemy równania wektorowego trzech współrzędnych:

$$o_x + td_x = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x),$$
$$o_y + td_y = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y),$$
$$o_z + td_z = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z).$$

Rys.2 Współrzędne barycentryczne na przeciętej płaszczyźnie

Możemy to zapisać jako standardowe równanie liniowe:

$$\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - o_x \\ a_y - o_y \\ a_z - o_z \end{bmatrix}$$

Najszybszym sposobem rozwiązania równania liniowego 3 na 3 jest metoda Cramera. Dostaniemy następujące rozwiązanie:

$$\beta = \frac{\begin{bmatrix} a_x - o_x & a_x - c_x & d_x \\ a_y - o_y & a_y - c_y & d_y \\ a_z - o_z & a_z - c_z & d_z \end{bmatrix}}{|A|},$$

$$\gamma = \frac{\begin{bmatrix} a_x - b_x & a_x - o_x & d_x \\ a_y - b_y & a_y - o_y & d_y \\ a_z - b_z & a_z - o_z & d_z \end{bmatrix}}{|A|},$$

$$t = \frac{\begin{bmatrix} a_x - b_x & a_x - c_x & a_x - o_x \\ a_y - b_y & a_y - c_y & a_y - o_y \\ a_z - b_z & a_z - c_z & a_z - o_z \end{bmatrix}}{|A|},$$

gdzie A jest macierzą następującej postaci:

$$A = \begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix},$$

oraz |A| oznacza wyznacznik A. Wyznacznik 3x3 ma rozwinięcie które może zostać odpowiednio rozbite. Równanie nasze przyjmie postać:

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix},$$

Zasada Cramera daje nam:

$$\beta = \frac{j(ei - hf) + k(gf - di) + l(dh - eg)}{M},$$

$$\gamma = \frac{i(ak - jb) + h(jc - al) + g(bl - kc)}{M},$$

$$t = \frac{f(ak - jb) + e(jc - al) + d(bl - kc)}{M},$$

gdzie:

$$M = a(ei - hf) + b(gf - di) + c(dh - eg)$$

Tego rodzaju równanie może zostać zaimplementowane przez stworzenie zmiennych, takich jak a oraz ei_minus_hf , kompilator będzie za nas wykonywał niezbędne działania.

3. Format OBJ

Plik OBJ jest popularnym formatem zapisu geometrii. Format opracowany został jako domyślny dla programu Advanced Visualizer animation package przez Wavefront Technologies. Format pliku jest otwarty i obsługiwany przez większość programów 3D. Plik może zostać zapisany w formacie ASCII (.obj) lub też w formacie binarnym (.mod). Na laboratorium wystarczy zaimplementować parser dla plików w formacie ASCII. Pliki muszą mieć rozszerzenie .obj.

Plik OBJ zawiera dane reprezentujące geometrię 3D za pomocą prostych zbiorów danych zdefiniowanych za pomocą wektorów, dodatkowo format ten ma możliwość definiowania materiałów oraz świateł w scenie. Wielokrotnie rozszerzany jest o dodatkowe funkcjonalności w zależności od potrzeb opisu sceny. Dokładny opis i specyfikację formatu OBJ można znaleźć pod adresem [9].

3.1 Struktura pliku

W pliku mogą być zdefiniowane następujące dane opisu geometrii:

- Współrzędne
  - **Wierzchołki geometrii (oznaczone jako v)**
  - **Współrzędne teksturowania (oznaczone jako vt)**
  - **Współrzędne normalnych (oznaczone jako vn)**
  - Parametry wierzchołków przestrzeni (oznaczone jako vp)

Dodatkowo mogą być opisane współrzędne innych elementów, zainteresowanych odsyłam do dokumentacji.

- Elementy
  - Punkty (oznaczone jako p)
  - Linie (oznaczone jako l)
  - **Ścianki (oznaczone jako f)**
  - Krzywe (oznaczone jako curv)
  - Krzywe 2D (oznaczone jako curv2)
  - Powierzchnie (oznaczone jako surf)

- Grupowanie
  - **Nazwa grupy (oznaczona jako g)**
  - Wygładzona grupa (oznaczona jako s)
  - Połączona grupa (oznaczone jako mg)
  - Nazwa obiektu – geometrii (oznaczone jako o)

- Współczynniki wyświetlania i renderingu (wymienię tylko istotne z punktu widzenia ćwiczenia, zainteresowanych odsyłam do dokumentacji)
  - **Nazwa materiału (usemtl)**
  - **Biblioteka materiału (mtllib)**

Najczęściej plik zaczyna się od definicji wierzchołków opisywanej geometrii. Lista wierzchołków określa kolejne wierzchołki ze współrzędnymi x,y,z i opcjonalnie z w. Jeśli wymienione wierzchołki składają się na grupę wcześniej występuje słowo g. Chcąc zdefiniować współrzędne teksturowania należy wprowadzić listę współrzędnych poprzedzoną słowem kluczowym vt. Normalne zdefiniowane są za pomocą współrzędnych x,y,z należy pamiętać że mogą one być nieznormalizowane, poprzedzone są słowem kluczowym vn. Ściany definiowane są na podstawie wskaźników do listy wierzchołków, tekstur i normalnych. Wielokąty mogą być zdefiniowane przez trzy wierzchołki/tekstury/ normalne. Istnieje kilka sposobów zdefiniowania ściany, każda linijka powinna zaczynać się od literału "f". Poniżej znajduje się przykład opisu sześcianu bez współrzędnych teksturowania.

Przykładowa budowa pliku OBJ

```
# cube.obj
#

g cube

v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0

vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0

f  1//2  7//2  5//2
f  1//2  3//2  7//2
f  1//6  4//6  3//6
f  1//6  2//6  4//6
f  3//3  8//3  7//3
f  3//3  4//3  8//3
f  5//5  7//5  8//5
f  5//5  8//5  6//5
f  1//4  5//4  6//4
f  1//4  6//4  2//4
f  2//1  6//1  8//1
f  2//1  8//1  4//1
```

3.2 Materiał

Materiał opisuje wizualny aspekt wielokąta i zdefiniowany jest w osobnym pliku z rozszerzeniem .mtl. Plik ten może zawierać kilka definicji różnych materiałów. Chcąc prawidłowo nadać materiał powinniśmy nadać nazwy obiektom i grupom:

- o [nazwa obiektu]
- ...
- g [nazwa grupy]
- ...
- usemtl [nazwa materiału]

Standardowo biblioteka wykorzystuje cieniowanie Phonga (podobnie jak nasz program) lecz sposób jej zapisu pozwala na jej modyfikację w celu implementacji innego modelu cieniowania. Pliki MTL są ściśle powiązane z plikami OBJ, które jak wiemy określają geometrię/siatkę w scenie, na nią właśnie będzie mapowany materiał zdefiniowany w plikach MTL. Poniżej znajduje się opis materiałów podstawowych:

- Materiał definiujemy za pomocą komendy newmtl :

  # define a material named 'Colored,
  newmtl Colored

- Składową ambient materiału deklarujemy za pomocą Ka, wartości składowych koloru zmieniają się od 0 do1.

  Ka 1.000 1.000 1.000 # white

- Podobnie wygląda deklaracja składowej matowej - Kd.

  Kd 1.000 1.000 1.000 # white

- Składową lustrzaną definiujemy za pomocą Ks, a rozbłysk z użyciem słowa kluczowego Ns.

  Ks 0.000 0.000 0.000 # black (off)

  Ns 10.000 # ranges between 0 and 1000

- Materiał może być przezroczysty..

  d 0.9 # some implementations use 'd,

  Tr 0.9 # others use 'Tr'

Poniżej znajduje się przykład pliku mtl:

newmtl Textured

Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
Ks 0.000 0.000 0.000
d 1.0
illum 2
map_Ka lenna.tga # the ambient texture map
map_Kd lenna.tga # the diffuse texture map (most of the
time, it will be the same as the ambient texture map)

```
map_Ks lenna.tga             # the specular texture map

map_d lenna_alpha.tga        # the alpha texture map
map_bump lenna_bump.tga      # the bump map
bump lenna_bump.tga          # some implementations use 'bump'
                             #instead of 'map_Bump'
```

W przypadku niejasności pomocna może się okazać biblioteka dla C++ pod adresem:

http://people.sc.fsu.edu/~jburkardt/cpp_src/obj_io/obj_io.html

4. Zadanie

Proszę zaimplementować klasę trójkąta, będącą konkretyzacją klasy abstrakcyjnej prymityw, zawierającą dowolny algorytm obliczania przecięcia promienia z trójkątem, a następnie zdefiniować klasę siatki geometrycznej (mesh) zawierającej listę trójkątów.

Proszę stworzyć w programie prosty parser formatu OBJ.

Na tym etapie, program powinien umożliwiać rendering sceny zawierającej, oprócz sfery i płaszczyzny, dowolny ostrosłup wczytany z pliku OBJ.

5. Bibliografia

[1] Karol Myszkowski. “Lighting reconstruction using fast and adaptive density estimation techniques”. Rendering Techniques ’97 (Proceedings of the Eighth Eurographics Workshop on Rendering), pages 321–326. Springer Verlag, 1997.

[2] Henrik Wann Jensen. Global illumination using photon maps. Rendering Techniques ’96 (Proceedings of the Seventh Euro-graphics Workshop on Rendering), pages 21–30. Springer Verlag, 1996.

[3] Henrik Wann Jensen. “Importance driven path tracing using the photon map”. Rendering Techniques ’95 (Proceedings of the Sixth Euro- graphics Workshop on Rendering), pages 326–335. Springer Verlag, 1995.

[4] Lawrence, J., Rusinkiewicz, S., and Ramamoorthi, R. Efficient brdf important sampling using a factored representation. In ACM Transaction of Graphics. Siggraph 2004, number 3, pages 496–505.

[5] A. Keller and I. Wald. Efficient Importance Sampling Techniques for the Photon Map. In *Proceedings of Vision, Modeling and Visualisation*, pages 271–279. IOS Press, 2000.

[6] Henrik Wann Jensen. Realistic Image Synthesis Using Photon Mapping. AK Peters, 2001, ISBN: 1568811470

[7] Ward, Gregory J. and Paul S. Heckbert: ”Irradiance Gradients”. In Proceedings of the Third Eurographics Workshop on Rendering, pp. 85-98, Bristol 1992.

[8] Henrik Wann Jensen and Niels Jørgen Christensen. “Efficiently Rendering Shadows using the Photon Maps”. In Proceedings of Compugraphics’95, pages 285–291, Alvor, December 1995.

[9] http://www.martinreddy.net/gfx/3d/OBJ.spec