Grafika Komputerowa - Ray Tracing

Autor:
Jacek Wieczorek 181043

Prowadzący:
Dr inż. Tomasz Kapłon

Wydział Elektroniki
III rok
Pn TP 8.15 - 11.00

8 stycznia 2012

1 Cel projektu

Celem projektu było zaimplementowanie rekursywnego algorytmu śledzenia promieni *Ray Tracing*.

2 Ray Tracing

Ray Tracing opiera się na analizowaniu poszczególnych promieni emitowanych przez źródło światła w kierunku od źródła światła do rzutni. Następnie wylicza się kolejne kierunki odbicia analizowanego promienia od ścian obiektów, aż do wyznaczania kierunku ostatniego odbicia promienia. Prosta wyznaczana przez ostatni kierunek odbicia analizowanego promienai przecina rzutnię, bądź nie.

3 Opis algorytmu

Głównym elementem w metodzie rekursywnego śledzenia promieni jest funkcja *Trace()*, której zadaniem jest obliczanie koloru piksela dla promienia zaczynającego się w punkcie **p** i biegnącym w kierunku wskazanym przez wektor **v**. W funkcji występuje rekurencja, a parametr **MAX_STEPS** ma zapobiec zapętleniu się algorytmu. Ogólny schemat działania funkcji *Trace()* przedstawiony zsoatał poniżej :

```
color c = Trace( point p, vector d, int step )
{
        color  local, reflected;          // składowe koloru
        point  q;                         // współrzędne punktu
        vector n, r ;                     // współrzędne wektora

        if( step > MAX )                  // przeanalizowano już zadaną liczbę poziomów drzewa
          return( background_color );

        q = Intersect( p, d, status );    // obliczenie punktu przecięcia promienia  i obiektu sceny

        if( status== light_source )       // trafione zostało źródło światła
          return( light_source_color );

        if( status== no_intersection )    // nic nie zostało trafione
          return( background_color );

        n = Normal( q );                  // obliczenie wektora normalnego w punkcie q
        r = Reflect( p, q, n );           // obliczenie kierunku odbicia promienia w punkcie q

        local = Phong( q, n, d );         // obliczenie oświetlenia lokalnego  w punkcie q

        reflected = Trace( q, r, step+1 );  //  obliczenie „reszty" oświetlenia dla punktu q

        return( local + reflected );      // obliczenie całkowitego oświetlenie dla q
}
```

Rysunek 1: Schemat działania funkcji Trace

4 Implementacja

4.1 Trace()

Funkcja Trace() przyjmuje 3 parametry wejściowe - punkt, wektor kierunkowy promienia, k-ta iteracja. Zwaraca następujące statusy :

- -1 : nie trafiono nic
- <-1 : trafiono źródło światła
- 0 : trafiono obiekt

```
1  int Trace(float *p, float *v, int k)
   {

       if( k > MAX_STEPS )
       {
           return -1;
       }

       int con = Intersect(p, v);

11     if(con == -1){
           return con; //nothing
       }
       else if(con < -1) {
           return con;  //light source
       }
       else if (con >=0 ) // is object
       {
           Normal(con);
           Reflect(v);
21         Phong(v, con);
           for(int i=0; i<3; i++)
           {
               color[i] += inters_c[i];;
           }
           Trace (inter, ref, k+1);
       }

       return 0;
   }
```

4.2 Intercest()

Funkcja Intercest ma za zadanie wyznaczyć współrzędne punktu przecięcia promienia *vec* z najbliższym obiektem sceny. Funkcja zwraca -1 gdy promień nie natrafi na żaden obiekt, w przypadku gdy promień natrafi na obiekt - jego numer, a gdy trafione zostanie źródło światła jego numer przeliczony na *-2 - nr*.

Aby wyznaczyć punkt przecięcia promienia z obiektem, skorzystano z równań promienia zapisanych w postaci parametrycznej i powierzchni sfery zapisanych w postaci uwikłanej :

$$x = r_{0x} + r_{dx}u \tag{1}$$

$$y = r_{0y} + r_{dy}u \qquad u > 0 \tag{2}$$

$$z = r_{0z} + r_{dz}u \tag{3}$$

gdzie wektory :

$$[r_{0x} \quad r_{0y} \quad r_{0z}] \tag{4}$$

$$[r_{dx} \quad r_{dy} \quad r_{dz}] \tag{5}$$

Równanie sfery :

$$x^2 + y^2 + z^2 - r^2 = 0 \tag{6}$$

Jeśli podstawić równania 1,2,3 do równania 6 uzyska się równanie kwadratowe ze względu na parametr u. Liczba punktów przecięcia promienia i sfery jest taka jak liczba rozwiązań tego równania. Gdy równanie ma dwa rozwiązania jako wartość parametru u, która odpowiada bliższemu punktowi przecięcia należy wybrać wartość mniejszą. Współrzędne punktu przecięcia uzyskuje się po wstawieniu do równań parametrycznych promienia, wyliczonej z równania kwadratowego wartości parametru u dla bliższego obserwatorowi (rzutni) punktu przecięcia. W tym celu wprowadzony został dodatkowy parametr **l** pozwalający określić, czy dany obiekt jest najbliższym.

```
int Intersect(float *point, float *vec){
    int st = -1;
    float l = FLT_MAX, a, b, c, d, r;



    for(int i=0; i< sphereCount; i++)
    {
        a = vec[0]*vec[0] + vec[1]*vec[1] + vec[2]*vec[2];
10      b = 2*(vec[0]*(point[0] - sphere_pos[i][0]) + vec[1]
            *(point[1] - sphere_pos[i][1])
            + vec[2]*(point[2] - sphere_pos[i][2]));
        c = point[0]*point[0] + point[1]*point[1]
            + point[2]*point[2] - 2*(sphere_pos[i][0]*point[0]
            + sphere_pos[i][1]*point[1] + sphere_pos[i][2]*point[2])
            + sphere_pos[i][0]*sphere_pos[i][0]
            + sphere_pos[i][1]*sphere_pos[i][1] + sphere_pos[i][2]*sphere_pos[i][2]
            - sphere_radius[i]*sphere_radius[i];
        d = b*b-4*a*c;
20
        if (d >= 0 ) {
```

```
            r = (-b - sqrt(d))/(2*a);

        if (r > 0 && r < l) {
            inter[0] = point[0] + r*vec[0];
            inter[1] = point[1] + r*vec[1];
            inter[2] = point[2] + r*vec[2];
            l = sqrt((inter[0]-point[0])*(inter[0]-point[0])
                + (inter[1]-point[1])*(inter[1]-point[1])
                + (inter[2]-point[2])*(inter[2]-point[2]));
            st = i;
            }
        }

    }

    if(st > -1) return st;
    l = FLT_MAX;
    for (int i = 0; i < sourceCount; i++) {
        if ((light_position[i][0] - point[0]) / vec[0]
            == (light_position[i][1] - point[1]) / vec[1]
            == (light_position[i][2] - point[2]) / vec[2])
        {
                r = sqrt((light_position[i][0] - point[0]) * (light_position[i][0] - point[0])
                    + (light_position[i][1] - point[1]) * (light_position[i][1] - point[1])
                    + (light_position[i][2] - point[2]) * (light_position[i][2] - point[2]));

            if (r < l ) {
                inter[0] = light_position[i][0];
                inter[1] = light_position[i][1];
                inter[2] = light_position[i][2];
                l = r;
                st = -2 - i;
            }
        }
    }

    return st;
}
```

4.3 Normal()

Funkcja Normal() słuzy do wyliczenia wektora normalnego do powierzchni obiektu w punkcie wyznaczonym przez funckcję Intercest().

```
void Normal(int sphere)
{
    //normal vector for sphere : [x - x0, y - y0, z - z0]
    for(int i=0; i<3; i++)
    {
        normalVector[i] = inter[i] - sphere_pos[sphere][i];
    }
    Normalization(normalVector);
}
```

4.4 Reflect()

Funkcja Reflect() służy do wyznaczenia wektora jednostkowego opisującego kierunek kolejnego śledzonego promienia. Promień ten powstaje w wyniku odbicia promienia wychodzącego z punktu p i biegnącego do punktu inter na powierzchni obiektu.

```
1  void Reflect(float * v)
   {
       //ref = 2 * cosx * normal vector - inv(v)
       //=> cosx = (normal * inv(v)) / (|normal| * |inv(v)|)
       float inv[3] = {-v[0], -v[1], -v[2]};
       float cos = dotProduct(normalVector, inv);
       for(int i=0; i<3; i++)
       {
           ref[i] = 2 * cos * normalVector[i] - inv[i];
       }
11     Normalization(ref);
   }
```

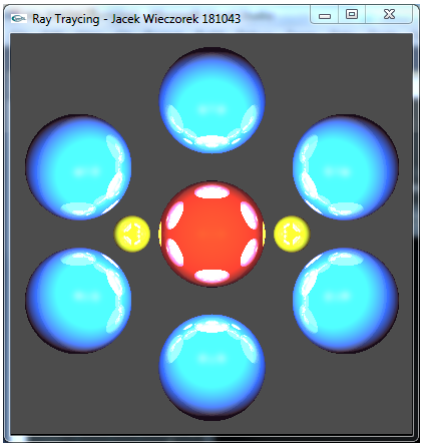
4.5 Phong()

Funkcja Phong() służy do oświetlenia lokalnego punktu. Jest ono sumą oświetleń pochądzących od wszytskich źródeł, widocznych z analizowanego punktu. Jako parametry określające wpływ odległości przyjęto 1.0, 0.01, 0.001.

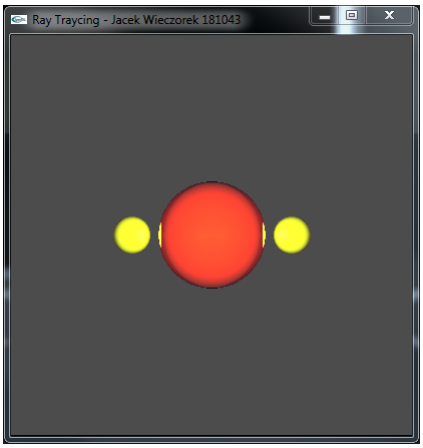
```
   void Phong(float * v, int which)
   {
       float light_vec[3];
       float reflection_vector[3];
       float viewer_v[3] = {-v[0], -v[1], -v[2]};
       float n_dot_l, v_dot_r;

8      for(int i=0; i<3; i++)
       {
           inters_c[i] = 0.0;
       }

       for(int i=0; i<sourceCount; i++)
       {

           for(int j=0; j<3; j++)
           {
18             light_vec[j] = light_position[i][j] - inter[j];
           }

           Normalization(light_vec);
           n_dot_l = dotProduct(light_vec, normalVector);
           for(int j=0; j<3; j++)
           {
               reflection_vector[j] = 2*(n_dot_l)*normalVector[j] - light_vec[j];
           }

28         Normalization(reflection_vector);
           v_dot_r = dotProduct(reflection_vector, viewer_v);

           if(v_dot_r < 0)
```

```
                v_dot_r = 0;

            if (n_dot_l > 0)
            {
                float x = sqrt((light_position[i][0] - inter[0])
                        * (light_position[i][0] - inter[0])
38                      + (light_position[i][1] - inter[1])*(light_position[i][1] - inter[1])
                        + (light_position[i][2] - inter[2])*(light_position[i][2] - inter[2]));
                for(int j=0; j<3; j++)
                {
                    inters_c[j] += (1.0/(1.0 + 0.01*x + 0.001*x*x))
                    * (sphere_diffuse[which][j]*light_diffuse[i][j]*n_dot_l
                    + sphere_specular[which][j]*light_specular[i][j]
                    * pow(double(v_dot_r), (double)sphere_specularhininess[which]))
                    + sphere_ambient[which][j]*light_ambient[i][j];
                }
48          }
            else
            {
                for(int j=0; j<3; j++)
                {
                    inters_c[j] += sphere_ambient[which][j]*global_a[j];
                }
            }
        }
    }
```

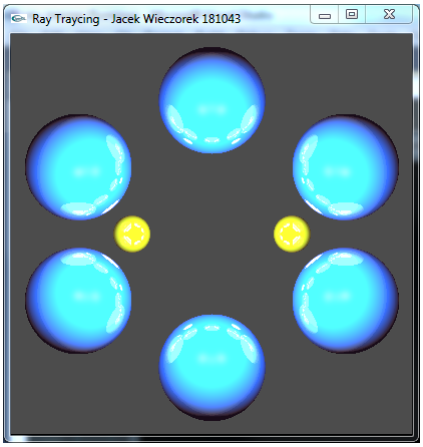
5 Przykładowe obrazy



Rysunek 2: Przykładowy obraz



Rysunek 3: Przykładowy obraz



Rysunek 4: Przykładowy obraz

6 Wnioski

Implementacja algorytmu Ray Taracing nie jest trudnym zadaniem. Jest to jedna z lepszych i łatwiejszych metod śledzenai promieni, niestety nie jest pozbawiona równiez wad. Konieczność analizowania każdego punktu ekranu powoduje, iż jest czasochłonna. Mogą również wystąpić efekty aliasingowe oraz nie wszystkie kierunki padania światła zostają zbadane co może powodować błędy w generowaniu oświetlenia.