POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka

GRAFIKA KOMPUTEROWA

Ray Tracing

AUTOR:

Paweł Twardawa

Prowadzący:

Dr inż. Marek Woda

OCENA PRACY:

Wrocław, 28.01.2018

Spis treści

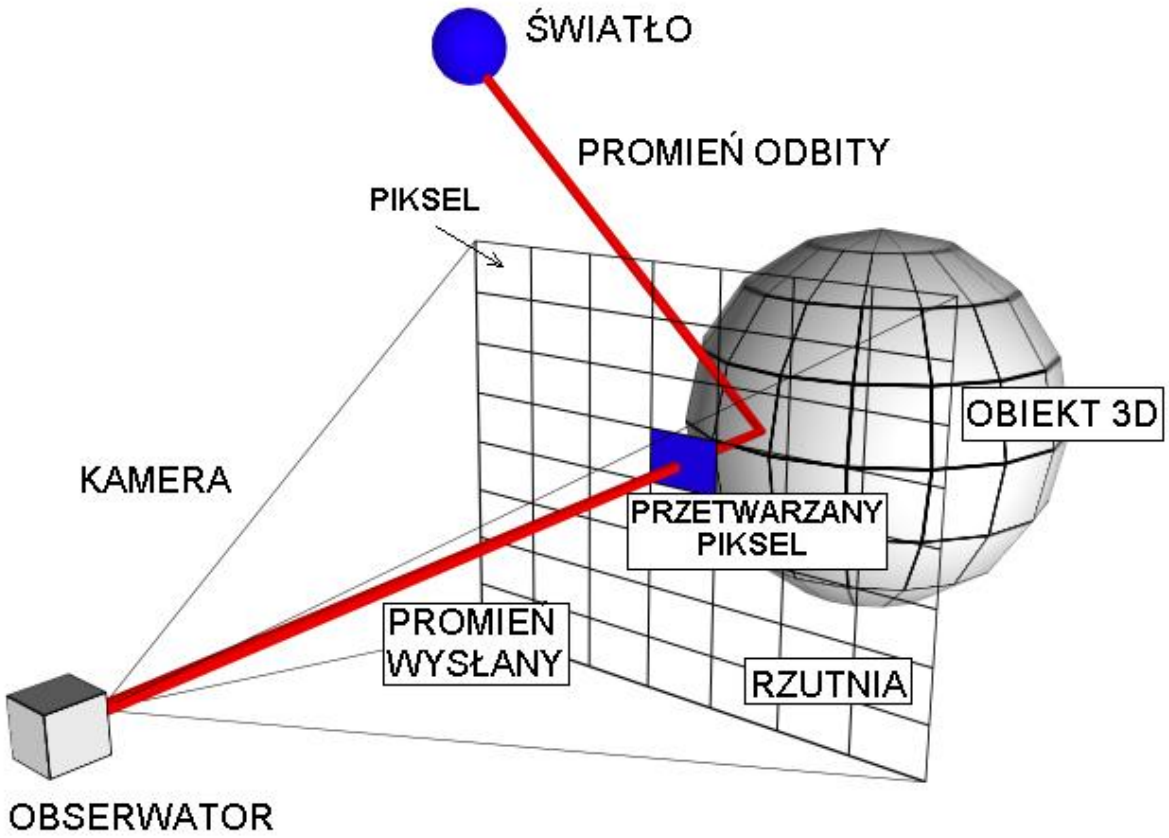
1. Wstęp 3
2. Analiza Problemu 3
6. Dane wejściowe 4
7. Implementacja algorytmu 5
8. Demonstracja aplikacji 10
9. Wnioski 11
Spis rysunków 12

1. Wstęp

Celem projektu było zaimplementowanie algorytmu rekursywnego śledzenia promieni oraz wyświetlenie sceny wczytanej z pliku .txt. Scena powinna składać się z sfer i źródeł światła. Jako sposób rzutowania przyjmuje się rzutowanie równoległe.

2. Analiza Problemu

Śledzenie promieni (ang. ray tracing) to technika generowania fotorealistycznych obrazów scen trójwymiarowych opierająca się na analizowaniu tylko tych promieni światła, które trafiają bezpośrednio do obserwatora. W rekursywnym śledzeniu promieni bada się dodatkowo promienie odbite, zwierciadlane oraz załamane.



Rysunek 1 – Wizualizacja wysyłania promieni

Algorytm śledzenia promieni:

1. Z punktu w którym znajduje się kamera wypuszczany jest promień (półprosta) w kierunku rzutni. Rzutnia podzielona jest na piksele, jeden (lub więcej) promieni przechodzi przez każdy piksel.
2. Wyszukiwane są wszystkie przecięcia promienia z obiektami.
3. Spośród uzyskanych punktów przecięcia wybiera się ten, który leży najbliżej kamery.
4. Punkt ten jest następnie przetwarzany.
   a) Najpierw są wypuszczane promienie z tego punktu w kierunku każdego ze świateł na scenie, by określić które oświetlą przetwarzany punkt. Na tym etapie można wyznaczyć cienie, testując czy odcinek pomiędzy punktem przecięcia, a światłem przecina jakiś obiekt - innymi słowy, czy jakiś obiekt zasłania konkretne światło.

b) Następnie dla wszystkich "widocznych" świateł, oblicza się, jasność punktu. Dodatkowo uwzględnia się takie parametry jak kolor punktu (np. odczytany z tekstury).

5. Jeśli obiekt jest przezroczysty to z tego punktu mogą zostać wypuszczone dodatkowe promienie (rekursywny ray tracing) - może to być zarówno promień odbity, jak i promień załamany - dla tych promieni algorytm jest powtarzany od punktu 2.

6. Dane wejściowe

Śledzenie promieni odbywa się dla sceny wczytanej z pliku. Plik zawiera informacje o rozmiarze obrazu, kolorze tła, źródłach światła oraz obiektach znajdujących się na scenie. Wczytywany plik ma następującą postać:

```
1.  dimensions 400 400
2.  background 0.3 0.3 0.3
3.  global 0.1 0.1 0.1
4.  sphere 0.7   3.0    0.0  -5.0      0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
5.  sphere 0.7  -3.0    0.0  -5.0      0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
6.  sphere 2.0   0.0    0.0  -3.0      0.8 0.1 0.0  0.8 0.1 0.0  0.2 0.1 0.2  40
7.  sphere 2.0   0.0   -5.0  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
8.  sphere 2.0   0.0    5.0  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
9.  sphere 2.0  -5.0    2.5  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
10. sphere 2.0  -5.0   -2.5  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
11. sphere 2.0   5.0   -2.5  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
12. sphere 2.0   5.0    2.5  -3.0      0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
13. source       0.0    0.0  15.0      0.2 0.2 0.2  0.4 0.4 0.4  0.2 0.2 0.2
14. source      -5.0    0.0  10.0      0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
15. source       5.0    0.0  10.0      0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
16. source       5.0    0.0  12.0      0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
17. source      -5.0    0.0  12.0      0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
```

Linia 1 zawiera rozmiar obrazu w pikselach
Linia 2 składowe R, G, B koloru tła
Linia 3 Dane opisująca globalne światło rozproszone
Linie 4 -12 Dane opisujące sfery. Promień sfery, współrzędne jej środka, współczynniki materiałowe powierzchni dla światła kierunkowego (specular), światła rozproszonego (diffuse) i otoczenia (ambient) oraz współczynnik połysku.
Linie 13 – 17 Dane opisujące źródła światła. Współrzędne źródła światła, składowe kolorów podstawowych R, G, B opisujące intensywność świecenia źródła światła otoczenia (ambient), światła rozproszonego (diffuse) i kierunkowego (specular).

7. Implementacja algorytmu

Ray tracing został zaimplementowany zgodnie z założeniami projektu. Funkcja przedstawiona na listingu 1 służy do wyznaczania koloru piksela obrazu wyświetlanego na ekranie. Funkcja jest wywoływana rekurencyjnie aż do osiągnięcia limitu rekurencji lub braku obiektów na jej drodze.

Listing 1.

```
1.  vecT Trace(vec p, vec d, int step)
2.  {
3.      vec v_normal; // wektor normalny do powierzchni obiektu
4.      vec normal; // wektor znormalizowany
5.      vec reflect; // wektor odbity od powierzchni
6.
7.      insersection status; //status trafienie promienia
8.      int nr_light; // numer zrodla swiatla
9.      int nr_sphere; // numer sfery
10.
11.     vecT color;
12.     color.tab[0] = 0.0;
13.     color.tab[1] = 0.0;
14.     color.tab[2] = 0.0;
15.
16.     vecT reflected;
17.     reflected.tab[0] = 0.0;
18.     reflected.tab[1] = 0.0;
19.     reflected.tab[2] = 0.0;
20.
21.     if (step > DEEP)
22.     {
23.         return color;
24.     }
25.
26.     punktTabsphere q_loc;
27.     q_loc = Intersect(p, d);
28.     v_normal[0] = q_loc.tab[0];
29.     v_normal[1] = q_loc.tab[1];
30.     v_normal[2] = q_loc.tab[2];
31.     nr_sphere = q_loc.sphere;
32.     nr_light = q_loc.light;
33.     status = q_loc.status;
34.
35.     //zrodlo swiatla
36.     if (status == LIGHT)
37.     {
38.         color.tab[0] += light_specular[nr_light][0];
39.         color.tab[1] += light_specular[nr_light][1];
40.         color.tab[2] += light_specular[nr_light][2];
41.         return color;
42.     }
43.
44.     //nic nie zostalo trafione
45.     if (status == NO_INTERSECTION)
46.     {
47.         color.tab[0] += background_color[0];
48.         color.tab[1] += background_color[1];
49.         color.tab[2] += background_color[2];
50.         return color;
51.     }
52.
53.     //wyliczenie wektora znormalizowanego do powierzchni
54.     vecT n_loc;
55.     n_loc = Normal(v_normal, nr_sphere);
56.     normal[0] = n_loc.tab[0];
57.     normal[1] = n_loc.tab[1];
```

```
58.     normal[2] = n_loc.tab[2];
59.
60.     //wyliczenie promienia odbitego
61.     vecT r_loc;
62.     r_loc = Reflect(p, v_normal, normal);
63.     reflect[0] = r_loc.tab[0];
64.     reflect[1] = r_loc.tab[1];
65.     reflect[2] = r_loc.tab[2];
66.
67.     color = Phong(v_normal, normal, d, nr_sphere);
68.
69.     reflected = Trace(v_normal, reflect, step + 1);
70.
71.
72.     color.tab[0] *= 0.5;
73.     color.tab[1] *= 0.5;
74.     color.tab[2] *= 0.5;
75.
76.     color.tab[0] += 0.6* reflected.tab[0];
77.     color.tab[1] += 0.6 * reflected.tab[1];
78.     color.tab[2] += 0.6 * reflected.tab[2];
79.
80.
81.     return color;
82. };
```

Funkcje wywoływane przez Trace:

Intersect – funkcja odpowiedzialna jest za znalezienie punktu przecięcia śledzonego promienia z obiektem sceny oraz zwrócenie odpowiedniego statusu. Funkcja zwraca trzy statusy trafienia. Jeżeli śledzony promień trafi w źródło światła zwraca status LIGHT, jeżeli w sfere SPHERE, w przypadku nie trafienia promienia w żaden obiekt zwraca status NO_ INTERSECTION. Punkt przecięcia śledzonego promienia ze sferą realizowane jest za pomocą przedstawionych równań:

$a = v_x^2 + v_y^2 + v_z^2$, gdzie v – wektor obserwatora

$b = 2 * (v_x * (p_x - o_x) + v_y * (p_y - o_y) + v_z * (p_z - o_z))$, gdzie p - położenia obserwatora, o - środek sfery

$c = (p_x - o_x)^2 + (p_y - o_y)^2 + (p_z - o_z)^2 - pr^2$, gdzie pr – promień sfery

$\Delta = b^2 - 4 * a * c$

Jeżeli delta jest równa lub większa od zera, to wyliczamy rozwiązania i sprawdzamy czy są dodatnie. Jeśli tak obliczamy współrzędne punktu i zwracamy status.

Listing 2.

```
1.  punktTabsphere Intersect(vec punktStart, vec direct)
2.  {
3.      punktTabsphere result;
4.      result.tab[0] = 0.0;
5.      result.tab[1] = 0.0;
6.      result.tab[2] = 0.0;
7.      result.sphere = 0;
8.      result.light = 0;
9.      result.status = NO_INTERSECTION;
10.
11.     int status = 0;
12.     //int lights = 0;
13.
```

```
14.     float a, b, c, delta, r;
15.
16.     for (int lights = 0; lights < 6; lights++)
17.     {
18.         float x, y, z;
19.         x = light_position[lights][0] - punktStart[0];
20.         y = light_position[lights][1] - punktStart[1];
21.         z = light_position[lights][2] - punktStart[2];
22.
23.         if ((x / direct[0]) == (y / direct[1]) && (y / direct[1]) == (z / direct[2]))
24.         {
25.             result.tab[0] = light_position[lights][0];
26.             result.tab[1] = light_position[lights][1];
27.             result.tab[2] = light_position[lights][2];
28.             result.status = LIGHT;
29.             return result;
30.         }
31.     }
32.
33.     for (int sphere = 0; sphere < 9; sphere++)
34.     {
35.         a = direct[0] * direct[0] + direct[1] * direct[1] + direct[2] * direct[2];
36.
37.         b = 2 * ((punktStart[0] - sphere_xyz[sphere][0])*direct[0] + (punktStart[1]
    - sphere_xyz[sphere][1])*direct[1] + (punktStart[2] - sphere_xyz[sphere][2])*direct[2];
38.
39.         c = (punktStart[0] * punktStart[0] + punktStart[1] * punktStart[1] + punktStart
    [2] * punktStart[2]) + (sphere_xyz[sphere][0] * sphere_xyz[sphere][0]
    + sphere_xyz[sphere][1] * sphere_xyz[sphere][1] + sphere_xyz[sphere][2]
    * sphere_xyz[sphere][2]) - 2 * (punktStart[0] * sphere_xyz[sphere][0]
    + punktStart[1] * sphere_xyz[sphere][1] + punktStart[2] * sphere_xyz[sphere][2])
    - sphere_radius[sphere] * sphere_radius[sphere];
40.
41.         delta = b*b - 4 * a*c;
42.
43.         if (delta >= 0)
44.         {
45.             r = (-b - sqrt(delta)) / (2 * a);
46.
47.             if (r > 0)
48.             {
49.                 result.tab[0] = punktStart[0] + r*direct[0];
50.                 result.tab[1] = punktStart[1] + r*direct[1];
51.                 result.tab[2] = punktStart[2] + r*direct[2];
52.                 result.sphere = sphere;
53.                 result.status = SPHERE;
54.                 break;
55.             }
56.         }
57.     }
58.
59.     return result;
60. }
```

Normal – oblicza znormalizowany wektor sfery za pomocą iloczynu wektorowego.

Listing 3.

```
vecT Normal(vec q, int sphere)
{
    vecT wynik;

    for (int i = 0; i < 3; i++)
    {
        wynik.tab[i] = (q[i] - sphere_xyz[sphere][i]) / sphere_radius[sphere];
    }

    return wynik;
}
```

Reflect – odpowiedzialna jest za znalezienie promienia odbitego od rozpatrywanego obiektu. Promień ten obliczamy z następującego wzoru:

$$\vec{R} = 2 * \cos\Theta * \vec{N} - \vec{L}$$

Listing 4.

```
vecT Reflect(vec startpunkt, vec q_sphere, vec normal)
{
    vecT result;
    vec ray;


    // oblicz wektor obiekt-obserwator
    ray[0] = startpunkt[0] - q_sphere[0];
    ray[1] = startpunkt[1] - q_sphere[1];
    ray[2] = startpunkt[2] - q_sphere[2];

    Normalization(ray);

    float n_dot_v;
    n_dot_v = dotProduct(ray, normal);

    //wyznacz promien odbity
    result.tab[0] = 2 * (n_dot_v)* normal[0] - ray[0];
    result.tab[1] = 2 * (n_dot_v)* normal[1] - ray[1];
    result.tab[2] = 2 * (n_dot_v)* normal[2] - ray[2];


    //jezeli wektor nie jest znormalizowany
    if (vector_length(result) > 1.0)
    {
        //return Normalization(result.tab);
        Normalization(result.tab);
        return result;
    }
    else
    {
        return result;
    }
}
```

Phong - funkcja oblicza oświetlenie lokalnego punktu. Kolor końcowy obliczanego punktu obliczany jest na podstawie sumy wszystkich widocznych z danego punktu źródeł światła wraz z uwzględnieniem rzucanego cienia przez inne obiekty. Badanie rzucanego cienia odbywa się za pomocą funkcji Intersect, która sprawdza czy pomiędzy badanym obiektem a źródłem światła nie znajduje się żaden obiekt. Oświetlenie wylicza się z następującego wzoru:

$$I_R = k_{aR} \cdot I_{aR} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dR} \cdot I_{dR} \cdot (\overline{N} \cdot \overline{L}) + k_{sR} \cdot I_{sR}(\overline{R} \cdot \overline{V})^n\right)$$

$$I_G = k_{aG} \cdot I_{aG} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dG} \cdot I_{dG} \cdot (\overline{N} \cdot \overline{L}) + k_{sG} \cdot I_{sG}(\overline{R} \cdot \overline{V})^n\right)$$

$$I_B = k_{aB} \cdot I_{aB} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dB} \cdot I_{dB} \cdot (\overline{N} \cdot \overline{L}) + k_{sB} \cdot I_{sB}(\overline{R} \cdot \overline{V})^n\right)$$

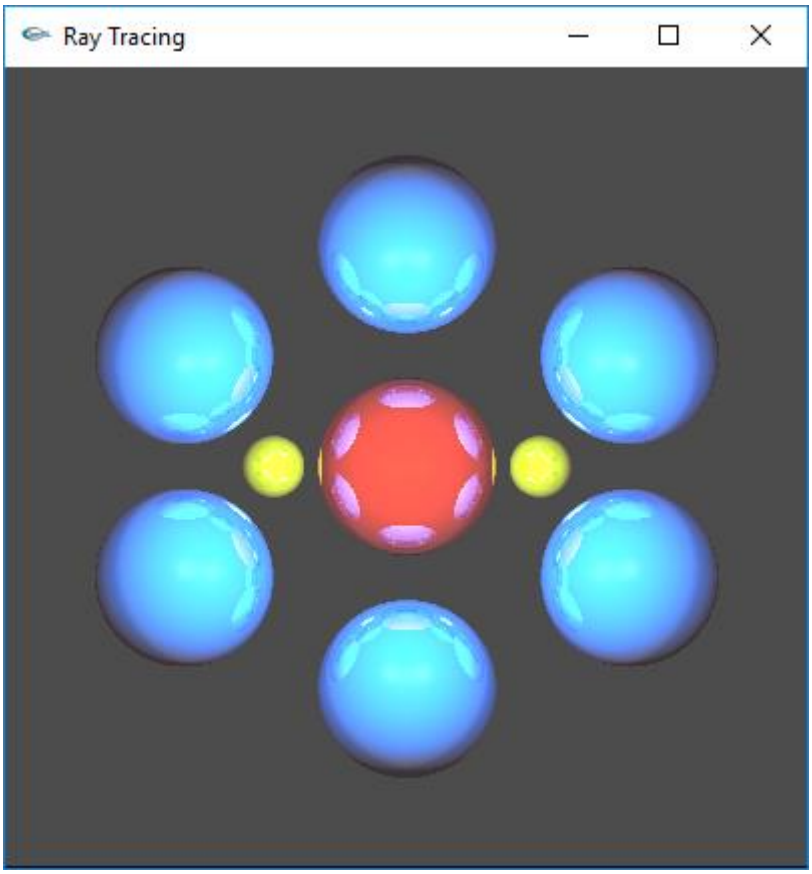
K – współczynnik materiału, *I* – natężenia źródeł światła, *n* – liczba świateł, *R* wektor odbicia światła, *v* wektor obserwatora, *n* wektor normalny.

Listing 5.

```
1.  vecT Phong(vec q, vec n, vec dest, int nr_sphere)
2.  {
3.      vecT color;
4.      color.tab[0] = 0.0;
5.      color.tab[1] = 0.0;
6.      color.tab[2] = 0.0;
7.
8.      vec lightVector;                    // wektor wskazujący źródeł
9.      vec reflectionVector;             // wektor kierunku odbicia światła
10.     vec viewer_v = { 0.0, 0.0, 1.0 };   // wektor kierunku obserwacji
11.     float doPLight, doPReflect;               // zmienne pomocnicze
12.
13.     float a, b, c, wspl;
14.     a = 1.0;
15.     b = 0.1;
16.     c = 0.01;
17.     wspl = 1 / (a + b + c);
18.
19.
20.     for (int k = 0; k < 5; k++) // liczba zrodel swiatla
21.     {
22.         //wektor swiatla i jego dlugosc
23.         lightVector[0] = light_position[k][0] - q[0];
24.         lightVector[1] = light_position[k][1] - q[1];
25.         lightVector[2] = light_position[k][2] - q[2];
26.
27.         vecT light_vec_loc;
28.         Normalization(lightVector);
29.
30.         doPLight = dotProduct(lightVector, n);
31.
32.         reflectionVector[0] = 2 * (doPLight)*n[0] - lightVector[0];
33.         reflectionVector[1] = 2 * (doPLight)*n[1] - lightVector[1];
34.         reflectionVector[2] = 2 * (doPLight)*n[2] - lightVector[2];
35.
36.         vecT reflection_vector_loc;
37.         Normalization(reflectionVector);
38.
```

```
39.         doPReflect = dotProduct(reflectionVector, viewer_v);
40.
41.         if (doPReflect < 0)                 // obserwator nie widzi oświetlanego
    punktu
42.             doPReflect = 0;
43.
44.         // sprawdzenie czy vec na powierzchni sfery jest oświetlany przez źródło
45.
46.         if (doPLight > 0)    // vec jest oświetlany,
47.         {                   // oświetlenie wyliczane jest ze wzorów dla modelu Phonga
48.
49.             color.tab[0] += wspl*(sphere_diffuse[nr_sphere][0] * light_diffuse[k][0] *
    doPLight + sphere_specular[nr_sphere][0] *light_specular[k][0] * pow(double(doPReflect)
    , double(sphere_specularhininess[nr_sphere]))) + sphere_ambient[nr_sphere][0] * light_a
    mbient[k][0] + sphere_ambient[nr_sphere][0] * global_a[0];
50.
51.             color.tab[1] += wspl*(sphere_diffuse[nr_sphere][1] * light_diffuse[k][1] *
    doPLight + sphere_specular[nr_sphere][1] *light_specular[k][1] * pow(double(doPReflect)
    , double(sphere_specularhininess[nr_sphere]))) + sphere_ambient[nr_sphere][1] * light_a
    mbient[k][1] + sphere_ambient[nr_sphere][1] * global_a[1];
52.
53.             color.tab[2] += wspl*(sphere_diffuse[nr_sphere][2] * light_diffuse[k][2] *
    doPLight + sphere_specular[nr_sphere][2] *light_specular[k][2] * pow(double(doPReflect)
    , double(sphere_specularhininess[nr_sphere]))) + sphere_ambient[nr_sphere][2] * light_a
    mbient[k][2] + sphere_ambient[nr_sphere][2] * global_a[2];
54.         }
55.         else                // vec nie jest oświetlany
56.         {                   // uwzględniane jest tylko światło rozproszone
57.
58.             color.tab[0] += sphere_ambient[nr_sphere][0] *global_a[0];
59.             color.tab[1] += sphere_ambient[nr_sphere][1] *global_a[1];
60.             color.tab[2] += sphere_ambient[nr_sphere][2] *global_a[2];
61.
62.         }
63.
64.     }
65.
66.     return color;
67. }
```

1. Demonstracja aplikacji



Rysunek 2 - efekt działania aplikacji

8. Wnioski

Implementacja algorytmu śledzenia promieni okazała się prosta w implementacji jednak nie odbyło się bez problemów. Pierwszym z nich było zbyt jasne oświetlenie sceny, jednak udało się go rozwiązać dobierając właściwe parametry światła w funkcji Trace. Kolejny to brak cienia na żółtych sferach, prawdopodobnie jest spowodowany błędem w obliczaniu światła odbitego, niestety problemu nie udało się rozwiązać. Pomimo napotkania problemów wygenerowana scena jest zbliżona do wzorcowej sceny.

Spis rysunków

Rysunek 1 – Wizualizacja wysyłania promieni 3
Rysunek 2 - efekt działania aplikacji 11