POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

JAKUB BALCERZAK
KIERUNEK: INF
SPECJALNOŚĆ: ISK

GRAFIKA KOMPUTEROWA
MINIPROJEKT

Rekursywne śledzenie promieni

Recursive Ray Tracing

PROWADZĄCY PRACĘ:

OCENA PRACY:

WROCŁAW 2019

Spis treści

1. Wstęp teoretyczny 1
2. Implementacja algorytmu rekursywnego śledzenia promieni 4
2.1. Pseudokod 4
2.2. Parametry i zmienne 5
2.3. Funkcje startowe 6
2.4. Wczytywanie danych 7
2.5. Funkcja renderująca scenę 9
2.6. Rekurencyjna funkcja śledzenia promienia 10
2.7. Funkcja Intersect() 12
2.8. Funkcja Normal() 14
2.9. Funkcja Reflect() 14
2.10. Funkcja Phong() 15
2.11. Funkcja normalizująca wektor 17
2.12. Funkcja licząca długość wektora 17
2.13. Funkcja licząca iloczyn skalarny dwu wektorów 17
3. Wyniki działania programu 18
3.1. Scena pierwotna 18
3.2. Scena zmodyfikowana 1 20
3.3. Scena zmodyfikowana 2 22
4. Podsumowanie 24
5. Literatura 26

Spis rysunków

Rysunek 1: scena pierwotna - głębokość rekurencji 0 19
Rysunek 2: scena pierwotna - głębokość rekurencji 1 19
Rysunek 3: scena pierwotna - głębokość rekurencji 10 20
Rysunek 4: scena zmodyfikowana 1 - głębokość rekurencji 1 21
Rysunek 5: scena zmodyfikowana 1 - głębokość rekurencji 100 21
Rysunek 6: scena zmodyfikowana 2 - głębokość rekurencji 1 22
Rysunek 7: scena zmodyfikowana 2 - głębokość rekurencji 100 23

1. Wstęp teoretyczny

Stawiany problem formułuje się następująco: istnieje punkt (x, y, z) obiektu 1, który oświetlany jest bezpośrednio przez źródło światła, ale także przez światło odbite od innych obiektów znajdujących się w jego otoczeniu. Jak wyliczyć zatem kolor oraz jasność piksela (x_p, y_p) będącego rzutem rozważanego punktu na powierzchni obiektu 1?

Odpowiedzią na powyższe pytanie jest metoda śledzenia promieni (ang. ray tracing). Aby ją przedstawić, należy przyjąć wpierw kilka założeń:

- scena zawiera wiele obiektów i wiele źródeł światła,
- możliwe jest obliczenie punktu przecięcia promienia światła oraz powierzchni obiektu,
- zadany jest model oświetlenia lokalnego tj. sposób oddziaływania powierzchni ze światłem.

Ideą metody śledzenia promieni jest badanie fikcyjnego promienia biegnącego od obserwatora przez kolejne piksele ekranu w kierunku obrazowanej sceny. Proces śledzenia – wyliczania kolejnych kierunków biegu promienia, prowadzony jest w oparciu o geometrię przestrzenną analizowanej sceny. Kończy się on w następujących przypadkach:

- śledzony promień nie trafia w żaden obiekt sceny, piksel wypełniany jest kolorem tła,
- promień trafia w obiekt, w którym następuje całkowite odbicie wewnętrzne,
- promień rozprasza się po osiągnięciu matowej powierzchni obiektu,
- nastąpiło ostatnie odbicie promienia, po którym nie trafi on już w żadne obiekty znajdujące się na scenie.

Powyższą ideę można zamknąć w krokowym opisie działania algorytmu:

Krok 1: Przez każdy piksel przeprowadza się promień pierwotny, a następnie wyznacza się promienie wtórne – śledzone. W ten sposób budowane jest drzewo oświetleń, którego korzeń stanowi promień pierwotny, a pozostałe węzły to promienie wtórne.

Krok 2: Dla każdego węzła w drzewie na podstawie analizy promieni testujących – łączących każde źródło światła na scenie z danym węzłem (jeżeli to źródło jest z niego widoczne), wyznaczone zostaje oświetlenie lokalne.

Krok 3: Graf modyfikuje się poprzez przypisanie węzłom obliczonych w kroku drugim odpowiednich oświetleń lokalnych.

Krok 4: Dla każdego węzła sumuje się oświetlenia pochodzące z poziomów „niższych”. Inaczej: oświetlenie węzła „wyższego” jest sumą oświetleń lokalnych węzłów „niższych”. Po obliczeniu wartości ostatniej, rozpatrywany piksel wypełniany jest odpowiednio do oświetlenia wyliczonego w korzeniu drzewa.

Niestety, ray tracing pomimo licznych zalet – dobrego odwzorowywania rzeczywistości, posiada także wady, które główne z nich wymienione zostały poniżej:

- może dochodzić do efektów aliasingu m.in. pomijania „małych" obiektów, zniekształcania ostrych krawędzi,
- duża złożoność obliczeniowa,
- w związku z faktem, że nie rozważa się wszystkich kierunków padania świateł na powierzchnie obiektów, mogą występować błędy w wyznaczaniu oświetleń poszczególnych punktów lub całych obszarów.

Następnym krokiem jest określenie wykonywanych obliczeń geometrycznych. Pierwszym z nich jest wyznaczanie punktu przecięcia promienia z obiektem sceny.

Definiujemy równanie parametryczne promienia:

$$R(u) = R_0 + R_d(u);\ u > 0, \text{ gdzie} \qquad \text{(Równanie 1)}$$

$R_0 = [\ r_{0x}, r_{0y}, r_{0z}]$ – początek promienia (punkt),

$R_d = [\ r_{dx}, r_{dy}, r_{dz}]$ – wektor jednostkowy opisujący kierunek biegu promienia.

Dalej określamy współrzędne punktu leżącego na rozpatrywanym promieniu:

$$x = r_{0x} + r_{dx}\, u \qquad \text{(Równanie 2)}$$

$$y = r_{0y} + r_{dy}\, u$$

$$z = r_{0z} + r_{dz}\, u.$$

W realizowanym projekcie obiektami znajdującymi się na scenie są sfery. Ich równanie prezentuje się następująco:

$$x^2 + y^2 + z^2 = R^2 \qquad \text{(Równanie 3)}$$

Podstawiając pod x, y, z prawe strony równań na współrzędne punktu, otrzymujemy równanie kwadratowe uzależnione od u.

Należy zatem wpierw obliczyć deltę. Jeżeli jej znak jest ujemny to oznacza, że rozpatrywany promień nie przecina w żadnym punkcie powierzchni sfery. W przeciwnym wypadku przecina i konieczne są dalsze obliczenia. Jeżeli delta > 0, to rozwiązanie u stanowi rozwiązanie bliższe. Obliczone u wstawić wystarczy do odpowiednich wzorów (Równanie 2).

Ostatnim elementem jest obliczanie oświetleń punktów. W tym celu należy wybrać model, który służyć będzie do wyznaczania oświetleń lokalnych np. model Phonga. Wymieniony model wymaga szeregu współczynników i czterech wektorów jednostkowych:

- N – wektor normalny do powierzchni sfery w analizowanym punkcie. Można wyznaczyć go poprzez różniczkowanie równania sfery,
- L – wektor kierunku padania światła, który jest znany,
- V – wektor kierunku obserwacji, który również jest znany,
- R – wektor kierunku światła odbitego, do jego wyliczenia stosuje się następującą zależność:

$$L + R = 2 \mid L \mid \cos(\alpha)\, N \qquad \text{(Równanie 4)}$$

Równanie 4 przekształca się do postaci:

$$R = 2 \cos(\alpha)\, N - L \qquad \text{(Równanie 5)}$$

Przed ostatecznym wyznaczeniem oświetlenia w punkcie sprawdza się jeszcze znak iloczynu skalarnego wektorów N i L. Jeżeli jest on równy zero, to oznacza, że N i L są prostopadłe, więc punkt nie jest oświetlany, a uwzględniane jest wyłącznie światło rozproszone. Analogicznie dla iloczynu ujemnego, gdyż wskazuje on na to, że dany punkt nie jest widoczny dla obserwatora. Gdy iloczyn jest dodatni wówczas wyznacza się oświetlenie lokalne punktu. Jest ono sumą oświetleń pochodzących od wszystkich źródeł światła. Tutaj do obliczeń wykorzystywany jest przyjęty uprzednio model.

Istnieją sposoby upraszczania obliczeń w metodzie śledzenia promieni. Wyróżnia się trzy podstawowe typy tychże sposobów:

1. redukcja kosztów śledzenia pojedynczego promienia:
   - metoda brył otaczających
2. redukcja liczby przecięć na drodze promienia
3. redukcja gęstości próbkowania:
   - metoda redukcji liczby promieni pierwotnych.

2. Implementacja algorytmu rekursywnego śledzenia promieni

W projekcie należało zaimplementować szereg funkcji, które wszystkie razem realizują procedurę śledzenia promieni i umożliwiają weryfikację ich działania w postaci wyświetlane grafiki zestawu 9-u sfer oświetlonych przez 5 źródeł światła.

Pierwsze prezentowane podpunkty odpowiadają ogólnemu przebiegowi programu. Kolejne, od funkcji rekurencyjnej poczynając, są odpowiedzialne za wyspecjalizowane działanie algorytmu.

Każda funkcja została opatrzona należytym komentarzem, który pozwala prześledzić przebieg operacji, jakie ma ona do wykonania. Dodatkowo, jeżeli było to niezbędne, na początku każdego podpunktu działanie odpowiednich metod zostaje zaprezentowane w skrótowym jego opisie.

2.1. Pseudokod

Poniższy pseudokod pochodzi ze strony przedmiotu:

```
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 color[0][0][0];
 q  = Intersect(p,  d,  status);//obliczenie  pubnktu  przecięcia  promienia
i obiektu sceny
 if (status == light_source)//trafione źródło światła
    return light_source_color;
 if (status == no_intersection && step == 0)//nic nie zostało trafione
    return background_color;
 n = Normal(q);//obliczenie wektora normalnego w punkcie q
 r = Reflect(p, q, n); //obliczenie punktu odbicia promienia w punkcie q
 local = Phong(q, n, d); //oblieczenie oświetlenia lokalnego w pubkcie q
 reflected = Trace(q, r, step + 1); //obliczenie "reszty" oświetlenia dla
punktu q
 return (local + reflected); //obliczenie całkowitego oświetlenia dla q
}
```

2.2. Parametry i zmienne

```
typedef float punkt[3];
//głębokość rekurencji
#define MAX 100
int step = 0;
//rozmiary okna
int horizontal = 0;
int vertical = 0;
float viewport_size = 18.0;
//punkt początkowy
punkt starting_point;
punkt starting_directions = { 0.0, 0.0, -1.0 };
//pomocnicze struktury do pracy z typem definiowanym
struct punktSphereStruct {
    punkt pos;
    int sphere;
    int light;
    int status;
};
struct punktStruct {
    punkt pos;
};
//parametry wiate
float light_position[5][3];
float light_specular[5][3];
float light_diffuse[5][3];
float light_ambient[5][3];
//kolor wypenienia ta
float background_color[3];
//parametry rozproszenia globalnego
float global_a[3];
//parametry sfer
float sphere_radius[9];
float sphere_position[9][3];
float sphere_specular[9][3];
float sphere_diffuse[9][3];
float sphere_ambient[9][3];
float sphere_shiness[9];
//biecy kolor piksela - struktura GLubyte przechowuje info
GLubyte pixel[1][1][3];
```

2.3. Funkcje startowe

```
//funkcja inicjacyjna
void Myinit(void)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-viewport_size / 2, viewport_size / 2, -viewport_size / 2,
             viewport_size / 2, -viewport_size / 2, viewport_size / 2);
    glMatrixMode(GL_MODELVIEW);
}
//funkcja main
int main(void)
{
    ReadFile();
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(horizontal, vertical);
    glutCreateWindow("MINIPROJEKT - Ray Tracing");
    Myinit();
    glutDisplayFunc(RenderScene);
    glutMainLoop();
}
```

2.4. Wczytywanie danych

Parametry sfer, źródeł światła oraz inne opisane w poprzednich punktach są zdefiniowane w pliku „scene.txt". Format zapisu danych jest określony na stronie przedmiotu i składa się z następujących elementów:

- Linia 1: rozmiar obrazu w pikselach (szerokość i wysokość):

  dimensions *width, height*

- Linia 2: kolor tła, czyli składowe R, G, B koloru tła:

  background *BR, BG, BB*

- Linia 3: dane opisujące globalne światło rozproszone. Składowe kolorów podstawowych R, G, B określające intensywności świecenia źródła:

  global *IgR, IgG, IgB*

- Linie 4, 5, ..., k: dane opisujące sfery. Promień sfery, współrzędne jej środka, współczynniki materiałowe powierzchni dla światła kierunkowego (specular), światła rozproszonego (diffuse) i otoczenia (ambient) oraz współczynnik połysku:

  sphere *r, x0, y0, z0, ksR, ksG, ksB, kdR, kdG, kdB, kaR, kaG, kaB, n*

- Linia k+1, k+2, ..., n: Dane opisujące źródła światła. Współrzędne punktu, w którym umieszczone jest źródło. Składowe kolorów podstawowych R, G, B określające intensywności świecenia źródła dla światła otoczenia (ambient), światła rozproszonego (diffuse) i kierunkowego (specular):

  source *xs, ys, zs, IsR, IsR, IsB, IdR, IdG, IdB, IaR, IaG, IaB*

```
//funkcja wczytuje dane z pliku tekstowe
//nazwa pliku: scena.txt
void ReadFile(void)  //odczytywanie danych z pliku
{
    fstream infile;
    infile.open("scene.txt");
    string tmp;
    if (infile.is_open() != true)
    {
        cout << "Nie udalo wczytac sie pliku";
    }
    else
    {
        infile >> tmp;
        infile >> horizontal;
        infile >> vertical;
```

```
infile >> tmp;
for (int i = 0; i < 3; i++) infile >> background_color[i];
infile >> tmp;
for (int i = 0; i < 3; i++) infile >> global_a[i];
for (int i = 0; i < 9; i++)

 {
            infile >> tmp;
            infile >> sphere_radius[i];
            infile >> sphere_position[i][0];
            infile >> sphere_position[i][1];
            infile >> sphere_position[i][2];
            infile >> sphere_specular[i][0];
            infile >> sphere_specular[i][1];
            infile >> sphere_specular[i][2];
            infile >> sphere_diffuse[i][0];
            infile >> sphere_diffuse[i][1];
            infile >> sphere_diffuse[i][2];
            infile >> sphere_ambient[i][0];
            infile >> sphere_ambient[i][1];
            infile >> sphere_ambient[i][2];
            infile >> sphere_shiness[i];
        }
        for (int i = 0; i < 5; i++)
        {
            infile >> tmp;
            infile >> light_position[i][0];
            infile >> light_position[i][1];
            infile >> light_position[i][2];
            infile >> light_specular[i][0];
            infile >> light_specular[i][1];
            infile >> light_specular[i][2];
            infile >> light_diffuse[i][0];
            infile >> light_diffuse[i][1];
            infile >> light_diffuse[i][2];
            infile >> light_ambient[i][0];
            infile >> light_ambient[i][1];
            infile >> light_ambient[i][2];
        }
        infile.close();
    }
}
```

2.5. Funkcja renderująca scenę

```
//funkcja renderujca obraz sceny
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
    //zagniedone pętle przetwarzają okno
    //przesuwajc się od lewej do prawej krawędzi
    //schodząc z górnej do dolnej krawędzi
    //na bieżąco przetwarzany jest - obliczany jest kolor jednego piksela
    float pix_x, pix_y;                 //bieżące współrzędne piksela
    int horizontal_2 = horizontal / 2;  //połowa rozmiaru okna
    int vertical_2 = vertical / 2;      //pozwala określić kierunek
przetwarzania okna
    punktStruct color;                  //pomocnicza zmienna przechowująca
kolor piksela - wynikowa funkcji  śledzenia
    for (int y = vertical_2; y > -vertical_2; y--)
    {
        for (int x = -horizontal_2; x < horizontal_2; x++)
        {
            //obliczenie pozycji kolejnego piksela w kontekście okna
obserwatora
            pix_x = (float)x / (horizontal / viewport_size);
            pix_y = (float)y / (vertical / viewport_size);
            //punkt - piksel ten ma współrzędne x y z odpowiednio
            //wg instrukcji laboratoryjnej
            starting_point[0] = pix_x;
            starting_point[1] = pix_y;
            starting_point[2] = viewport_size;
            //wywołanie dla bieżącego piksela oraz promienia obserwacji
funkcji rekurencyjnej liczącej jego przebieg
            //tym sposobem otrzymujemy trzy dane do przetworzenia w celu
wyłuskania koloru
            color = Trace(starting_point, starting_directions, step);
            //wyznaczanie koloru piksela
            //wg instrukcji laboratoryjnej
            if (color.pos[0] > 1)
                pixel[0][0][0] = 255;
            else
                pixel[0][0][0] = color.pos[0] * 255;

            if (color.pos[1] > 1)
                pixel[0][0][1] = 255;
            else
                pixel[0][0][1] = color.pos[1] * 255;

            if (color.pos[2] > 1)
                pixel[0][0][2] = 255;
            else
                pixel[0][0][2] = color.pos[2] * 255;
            //inkrementacja pozycji rastrowej
            glRasterPos3f(pix_x, pix_y, 0);
            glDrawPixels(1, 1, GL_RGB, GL_UNSIGNED_BYTE, pixel);
        }
    }
    glFlush();
}
```

2.6. Rekurencyjna funkcja śledzenia promienia

Zadaniem funkcji Trace() jest obliczanie koloru piksela (punktu) dla promienia zaczynającego się w punkcie p i biegnącego w kierunku wskazywanym przez wektor v. Dzięki zawartej w tej funkcji rekurencji możliwe jest obliczenie kolorów dla kolejnych punktów przecięć na drodze śledzonego promienia. Warunkiem końcowym rekurencji jest przekroczenie głębokości rekurencji określane w parametrze next_step.

```
//funckja wylicza kolejne punkty przecieca dla sledzonego promienia
//p - jest zrodem promienia
//v - wskazuje jego kierunek
punktStruct Trace(punkt p, punkt v, int next_step)
{
    punkt q;  //punkt przeciecia na powierzchni sfery
    punkt n;  //wektor znormalizowany do powierzchni obiektu w punkcie q
    punkt r;  //wektor odbicia
    //zmienne pomocnicze
    int status;             //werykacja w co trafil promien
    int light_number;       //numer zrodla swiatla
    int sphere_number;      //numer sfery
    //oswietlenie lokalne punktu
    punktStruct local;
    local.pos[0] = 0.0;
    local.pos[1] = 0.0;
    local.pos[2] = 0.0;
    //oswietlenie punktu pochodzace z promieni odbitych
    punktStruct reflected;
    reflected.pos[0] = 0.0;
    reflected.pos[1] = 0.0;
    reflected.pos[2] = 0.0;
    //jezeli osiagnieto pozadana gebokosc rekurencji dla danego promienia,
a on nie trafil
    //w zaden obiekt na akualnym etapie, to miejsce to bedzie kolorwane
kolorem tla
    if (next_step > MAX)
    {
        local.pos[0] += background_color[0];
        local.pos[1] += background_color[1];
        local.pos[2] += background_color[2];
        return local;
    }
    //wyliczenie punktu przeciecia dla skierowanego promienia
    //wychodzacego z punktu p
    //skierowanego wedug wektora v
    punktSphereStruct tempSS = Intersect(p, v);
    q[0] = tempSS.pos[0];
    q[1] = tempSS.pos[1];
    q[2] = tempSS.pos[2];
    status = tempSS.status;
    light_number = tempSS.light;
    sphere_number = tempSS.sphere;
```

```
//1 - promien trafil w zrodlo swiatla
    if (status == 1)
    {
        local.pos[0] += light_specular[light_number][0];
        local.pos[1] += light_specular[light_number][1];
        local.pos[2] += light_specular[light_number][2];
        return local;
    }
    //2 - nic nie zostalo trafione przez promienie
    //piksel przyjmuje wartosci barwy tla
    if (status == 2)
    {
        local.pos[0] += background_color[0];
        local.pos[1] += background_color[1];
        local.pos[2] += background_color[2];
        return local;
    }
    //wyliczenie wektora znormalizowanego do powierzchni w punkcie q sfery
sphere
    punktStruct tempS = Normal(q, sphere_number);
    n[0] = tempS.pos[0];
    n[1] = tempS.pos[1];
    n[2] = tempS.pos[2];
    //wyliczenie kierunku promienia odbitego od powierzchni obiektu
    tempS = Reflect(p, q, n);
    r[0] = tempS.pos[0];
    r[1] = tempS.pos[1];
    r[2] = tempS.pos[2];
    //owietlenie lokalne wyliczane jest z modelu Phonga dla danej sfery w
danym punkcie
    local = Phong(q, n, sphere_number);
    //dalsze obliczenia, czyli obliczenia majace uzyskac poprzez rekurencje
    //cakowita wartosc oswietlenia w biezacym punkcie
    reflected = Trace(q, r, next_step + 1);
    //wyliczenie calkowitego oswietlenia w punkcie rysowanym
    local.pos[0] += reflected.pos[0];
    local.pos[1] += reflected.pos[1];
    local.pos[2] += reflected.pos[2];
    return local;
};
```

2.7. Funkcja Intersect()

Funkcja Intercest() ma za zadanie wyznaczenie współrzędnych intersected_vec punktu przecięcia z najbliższym obiektem sceny. Argumentami funkcji są punkt p będący początkiem promienia i kierunek promienia v. Zmienna status przyjmuje następujące wartości:

- 1 – gdy promień przecina źródło światła,
- 2 – gdy promień chybia i trafia w próżnię,
- 3 – gdy promień przecina powierzchnię sfery.

```
//funkcja wyznaczajaca punkt przeciecia (pierwszy na drodze)
//sledzonego promienia oraz obiektu (tu sfery/swiata/prozni) w ktory
promien trafia
punktSphereStruct Intersect(punkt p, punkt v)
{
    //utworzenie obiektu definujacego punkt przeciecia
    //struktura zawiera wszytsko co jednoznacznie pozwala
    //taki punkt wyswietlic w oknie
    punktSphereStruct intersected_vec;
    intersected_vec.pos[0] = 0.0;
    intersected_vec.pos[1] = 0.0;
    intersected_vec.pos[2] = 0.0;
    intersected_vec.sphere = 0;
    intersected_vec.light = 0;
    intersected_vec.status = 0;
    //zmienne pomocnicze do obliczen
    float x, y, z;
    //okreslenie czy promien trafil w zrodo
    for (int i = 0; i < 5; i++)
    {
        //wyliczenie wektora kierunkowego punkt -> zrodo swiatla
        x = light_position[i][0] - p[0];
        y = light_position[i][1] - p[1];
        z = light_position[i][2] - p[2];
        //sprawdzenie czy wektor powyzszy jest tozsamy z aktualnym
kierunkiem v promienia z punktu p
        if ((x / v[0]) == (y / v[1]) && (y / v[1]) == (z / v[2]))
        {
            //pozycja uderzonego zroda swiatla
            intersected_vec.pos[0] = light_position[i][0];
            intersected_vec.pos[1] = light_position[i][1];
            intersected_vec.pos[2] = light_position[i][2];
            //ktore dokladnie zrodlo
            intersected_vec.light = i;
            intersected_vec.status = 1;
            return intersected_vec;
        }
    }
```

```
    //sprawdzenie czy promien trafil w ktorakolwiek ze sfer
    float a, b, c, delta, u;//zestaw zmiennych do obliczen testowych
    for (int i = 0; i < 9; i++)
    {
        //przeprowadzane poniej obliczenia przebiegaja zgodnie ze wzorami
        //podanymi w instrukcji laboratoryjnej
        //obliczenie parametr□unkcji kwadratowej
        a = v[0] * v[0] + v[1] * v[1] + v[2] * v[2];
        b = 2 * ((p[0] - sphere_position[i][0])*v[0] + (p[1] -
sphere_position[i][1])*v[1] + (p[2] - sphere_position[i][2])*v[2]);
        c = (p[0] * p[0] + p[1] * p[1] + p[2] * p[2])
            + (sphere_position[i][0] * sphere_position[i][0] +
sphere_position[i][1] * sphere_position[i][1] + sphere_position[i][2] *
sphere_position[i][2])
            - 2 * (p[0] * sphere_position[i][0] + p[1] *
sphere_position[i][1] + p[2] * sphere_position[i][2])
            - sphere_radius[i] * sphere_radius[i];
        //wyliczenie delty
        delta = b * b - 4 * a*c;
        //jeeli delta >= 0 to istniej rozwiazanie/rozwizania (nalezy wziac
to blizsze w przypadku istnienia dwoch)
        if (delta >= 0)
        {
            //brane jest rozwiazanie blizsze [jak wyzej opisano]
            u = (-b - sqrt(delta)) / (2 * a);
            if (u > 0)
            {
                //wyliczenie wspolrzednych punktu, gdzie padl promien
                intersected_vec.pos[0] = p[0] + u * v[0];
                intersected_vec.pos[1] = p[1] + u * v[1];
                intersected_vec.pos[2] = p[2] + u * v[2];
                //z zaznaczeniem, ktorej to sfery dotyczy
                intersected_vec.sphere = i;
                intersected_vec.status = 3;
                break;
            }
        }
    }
    //jezeli wartosc status nie bya modyfikowana, to znaczy to,
    //ze promien zaginal w przestrzeni tj. nie dotarl do zadnego obiektu
    if (intersected_vec.status == 0)
    {
        intersected_vec.status = 2;
    }
    return intersected_vec;
}
```

2.8. Funkcja Normal()

```
//funkcja licząca wektor normalny do powierzchni w podanym punkcie - q dla
obiektu o numerze sphere
punktStruct Normal(punkt v, int sphere_number)
{
    punktStruct normal_vec;
    for (int i = 0; i < 3; i++)
    {
        normal_vec.pos[i] = (v[i] - sphere_position[sphere_number][i]) /
sphere_radius[sphere_number];
    }
    return normal_vec;
}
```

2.9. Funkcja Reflect()

Funkcja Reflect() wyznacza wektor jednostkowy opisujący kierunek kolejnego śledzonego promienia. Powstaje on w wyniku odbicia promienia wychodzącego z punktu p i biegnącego do punktu q na powierzchni rozważanego obiektu. Dla wyznaczenia poszukiwanego wektora należy znać: kierunek promienia wyznaczony przez parę punktów p i q oraz wektor normalny n do powierzchni w punkcie q.

```
//funkcja liczaca wektor odbicia
punktStruct Reflect(punkt p, punkt q, punkt n)
{
    punktStruct reflect_vec; //wektor odbicia
    punkt direct_vec;        //wektor kierunkowy
    direct_vec[0] = p[0] - q[0];
    direct_vec[1] = p[1] - q[1];
    direct_vec[2] = p[2] - q[2];
    //pomocnicza konwersja
    punktStruct temp = Normalization(direct_vec);
    direct_vec[0] = temp.pos[0];
    direct_vec[1] = temp.pos[1];
    direct_vec[2] = temp.pos[2];
    //obliczenie iloczynu skalaranego w celu podstawienia wartosci do wzoru
    float r_dot_d;
    r_dot_d = dotProduct(direct_vec, n);
    //obliczenie wektora promienia odbitego
    reflect_vec.pos[0] = 2 * (r_dot_d)* n[0] - direct_vec[0];
    reflect_vec.pos[1] = 2 * (r_dot_d)* n[1] - direct_vec[1];
    reflect_vec.pos[2] = 2 * (r_dot_d)* n[2] - direct_vec[2];
    //jezeli wektor nie jest jednostkowy, to nalezy go wpierw znormalizowac
    if (vector_length(reflect_vec) > 1.0)
        return Normalization(reflect_vec.pos);
    else return reflect_vec;
}
```

2.10. Funkcja Phong()

Zadaniem funkcji Phong() jest wyznaczenie oświetlenia lokalnego w punkcie q. Oświetlenie to jest sumą oświetlenia pochodzącego ze wszystkich osiągalnych tj. widocznych źródeł światła.

```
//uzywajc modelu Phonga funkcja oblicza oswietlenie lokalne danego punktu
//q - obliczony punkt przeciecia
//n - wektor normalny do plaszczyzny w punkcie q
punktStruct Phong(punkt q, punkt n, int sphere)
{
    //wyliczony zostanie kolor punktu
    //inicjalizacja dla bezpieczenstwa
    punktStruct color;
    color.pos[0] = 0.0;
    color.pos[1] = 0.0;
    color.pos[2] = 0.0;
    //pomocnicze zmienne
    punkt light_vec;        //wektor skierowany ze zwrotem do zrodla
    punkt reflection_vec;   //wektor promienia odbitego
    punkt viewer_vec;       //wektor skierowany do obserwatora
    viewer_vec[0] = 0.0;
    viewer_vec[1] = 0.0;
    viewer_vec[2] = 1.0;
    //odpowiednie iloczny skalarane
    float n_dot_l, v_dot_r;
    //wspolczynniki okreslajace wplyw odleglosci zrodla swiatla na
oswietlenie punktu
    float a, b, c, scale;
    a = 1.0;
    b = 0.1;
    c = 0.01;
    scale = 1 / (a + b + c);
    //obliczenia oswietlenia danego punktu musz zostac przeprowadzone dla
kazdego zrodla swiatla
    for (int i = 0; i < 5; i++)
    {
        //wyliczenie kierunku wektora z punktu przeciecia w kierunku zrodla
swiatla
        light_vec[0] = light_position[i][0] - q[0];
        light_vec[1] = light_position[i][1] - q[1];
        light_vec[2] = light_position[i][2] - q[2];
        //normalizacja wyliczonego wektora
        punktStruct temp = Normalization(light_vec);
        light_vec[0] = temp.pos[0];
        light_vec[1] = temp.pos[1];
        light_vec[2] = temp.pos[2];
        //iloczyn skalarny
        n_dot_l = dotProduct(light_vec, n);
```

```
//wg wzorow z instrukjci laboratoryjnej
        //nastepuje wyliczenie kierunku odbicia
        reflection_vec[0] = 2 * (n_dot_l)*n[0] - light_vec[0];
        reflection_vec[1] = 2 * (n_dot_l)*n[1] - light_vec[1];
        reflection_vec[2] = 2 * (n_dot_l)*n[2] - light_vec[2];
//nastepnie wektor odbica jest takze normalizowany
//zgodnie z wymogami modelu Phonga, gdzie kazdy wektor musi taki byc
        temp = Normalization(reflection_vec);
        reflection_vec[0] = temp.pos[0];
        reflection_vec[1] = temp.pos[1];
        reflection_vec[2] = temp.pos[2];
        //iloczyn skalarany
        v_dot_r = dotProduct(reflection_vec, viewer_vec);
        //sprawdzenie, czy punkt jest w ogole widoczny przez obserwatora
        if (v_dot_r < 0)
            v_dot_r = 0;
        //sprawdzenie czy punkt na powierzchni sfery jest oswietlany przez
zrodlo
        if (n_dot_l > 0)
        {
            //punkt jest oswietlany dalej...
            //obliczenia wg modelu Phonga zdefiniowanego w instrukcji
laboratoryjnej
            color.pos[0] += scale * (sphere_diffuse[sphere][0] *
light_diffuse[i][0] * n_dot_l + sphere_specular[sphere][0] *
light_specular[i][0] * pow(double(v_dot_r),
double(sphere_shiness[sphere])))
                          + sphere_ambient[sphere][0] * light_ambient[i][0]
+ sphere_ambient[sphere][0] * global_a[0];
            color.pos[1] += scale * (sphere_diffuse[sphere][1] *
light_diffuse[i][1] * n_dot_l + sphere_specular[sphere][1] *
light_specular[i][1] * pow(double(v_dot_r),
double(sphere_shiness[sphere])))
                          + sphere_ambient[sphere][1] * light_ambient[i][1]
+ sphere_ambient[sphere][1] * global_a[1];
            color.pos[2] += scale * (sphere_diffuse[sphere][2] *
light_diffuse[i][2] * n_dot_l + sphere_specular[sphere][2] *
light_specular[i][2] * pow(double(v_dot_r),
double(sphere_shiness[sphere])))
                          + sphere_ambient[sphere][2] * light_ambient[i][2]
+ sphere_ambient[sphere][2] * global_a[2];
        }
        else{
            //w przeciwnym wypadku punkt nie jest oswietlany
            //jego oswietlenie wynika wyacznie ze swiatla rozproszonego
            color.pos[0] += sphere_ambient[sphere][0] * global_a[0];
            color.pos[1] += sphere_ambient[sphere][1] * global_a[1];
            color.pos[2] += sphere_ambient[sphere][2] * global_a[2];
        }
    }
    return color;
}
```

2.11. Funkcja normalizująca wektor

```
//funkcja normalizujaca podany wektor v -- dzielenie wektora przez jego
dlugosc
punktStruct Normalization(punkt v)
{
    //wktor znormalizowany
    //inicjacja dla bezpieczenstwa
    punktStruct normalized_vec;
    normalized_vec.pos[0] = 0.0;
    normalized_vec.pos[1] = 0.0;
    normalized_vec.pos[2] = 0.0;
    //algorytm wg instrukcji laboratoryjnej
    float d = 0.0;
    int i;
    for (i = 0; i < 3; i++)
        d += v[i] * v[i];
    d = sqrt(d);
    if (d > 0.0)
        for (i = 0; i < 3; i++)
            normalized_vec.pos[i] = v[i] / (d*1.0);

    return normalized_vec;
}
```

2.12. Funkcja licząca długość wektora

```
//pomocnicza funckja liczaca dugosc wektora
float vector_length(punktStruct v)
{
    return (v.pos[0] * v.pos[0] + v.pos[1] * v.pos[1] + v.pos[2] *
v.pos[2]);
}
```

2.13. Funkcja licząca iloczyn skalarny dwu wektorów

```
//funkcja liczaca iloczyn skalarny dwoch wektorow
float dotProduct(punkt p, punkt q)
{
    return (p[0] * q[0] + p[1] * q[1] + p[2] * q[2]);
}
```

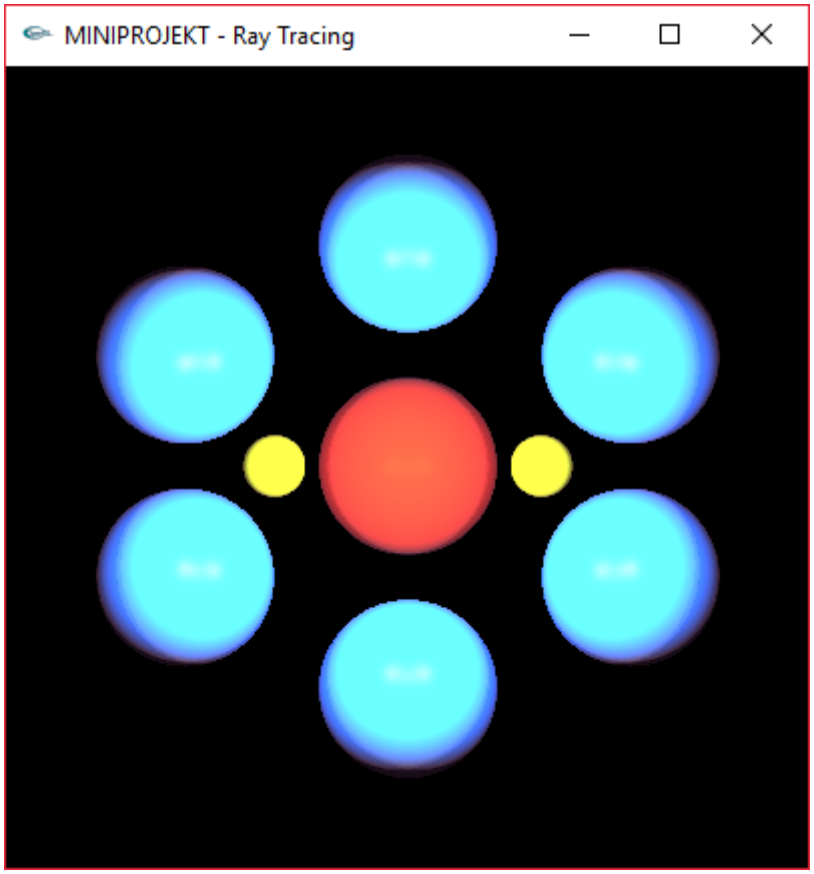
3. Wyniki działania programu

W pierwszym podpunkcie prezentowane są grafiki wygenerowane dla pliku podstawowego pochodzącego ze strony przedmiotu. W kolejnych dwu zmianie ulegają położenia małych i dużych sfer a także źródeł światła.

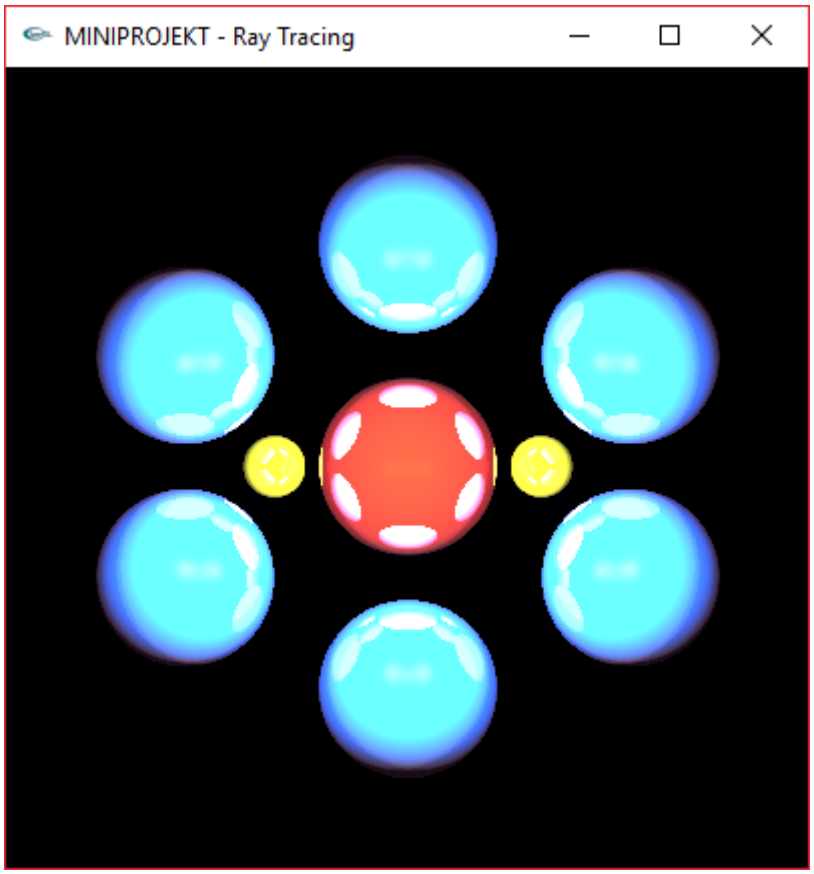
W każdym podpunkcie zawarta zostaje treść bieżącego pliku „scene.txt” w celu weryfikacji wykonania się algorytmu dla prezentowanych danych wejściowych. Dodatkowo, dla każdego zestawu rozpatrywano proces śledzenie promieni dla dwóch lub trzech głębokości rekurencji.

3.1. Scena pierwotna

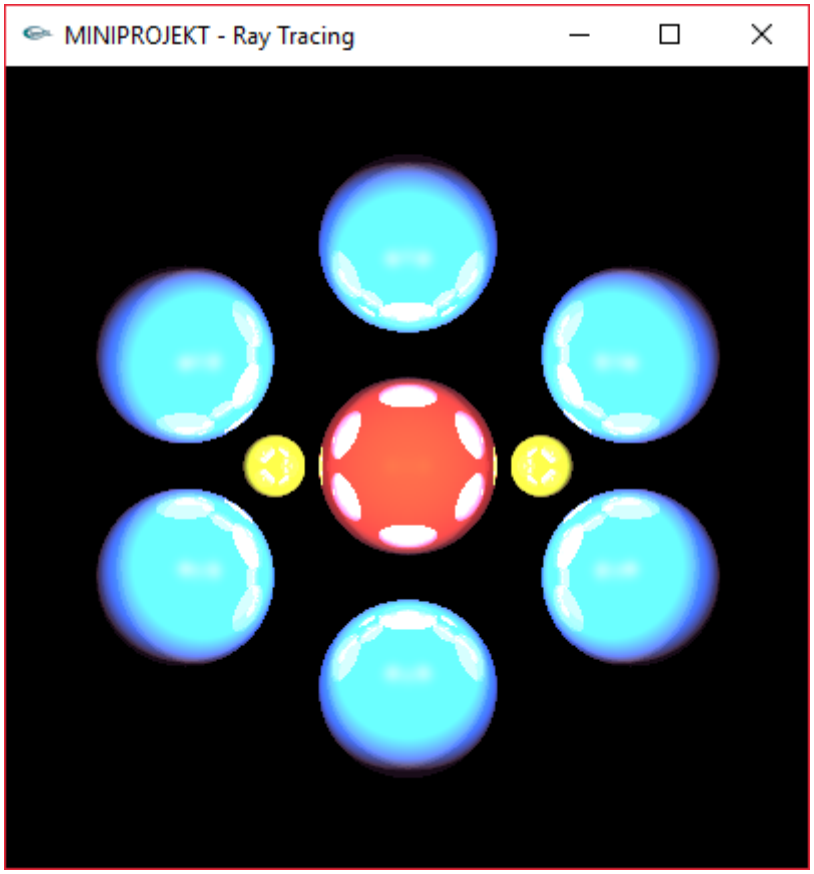
```
dimensions 400 400
background 0.0 0.0 0.0
global     0.1 0.1 0.1
sphere     0.7   3.0    0.0  -5.0     0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere     0.7  -3.0    0.0  -5.0     0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere     2.0   0.0    0.0  -3.0     0.8 0.1 0.0  0.8 0.1 0.0  0.2 0.1 0.2  40
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
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
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
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
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
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
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
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
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
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
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
```



Rysunek 1: scena pierwotna - głębokość rekurencji 0



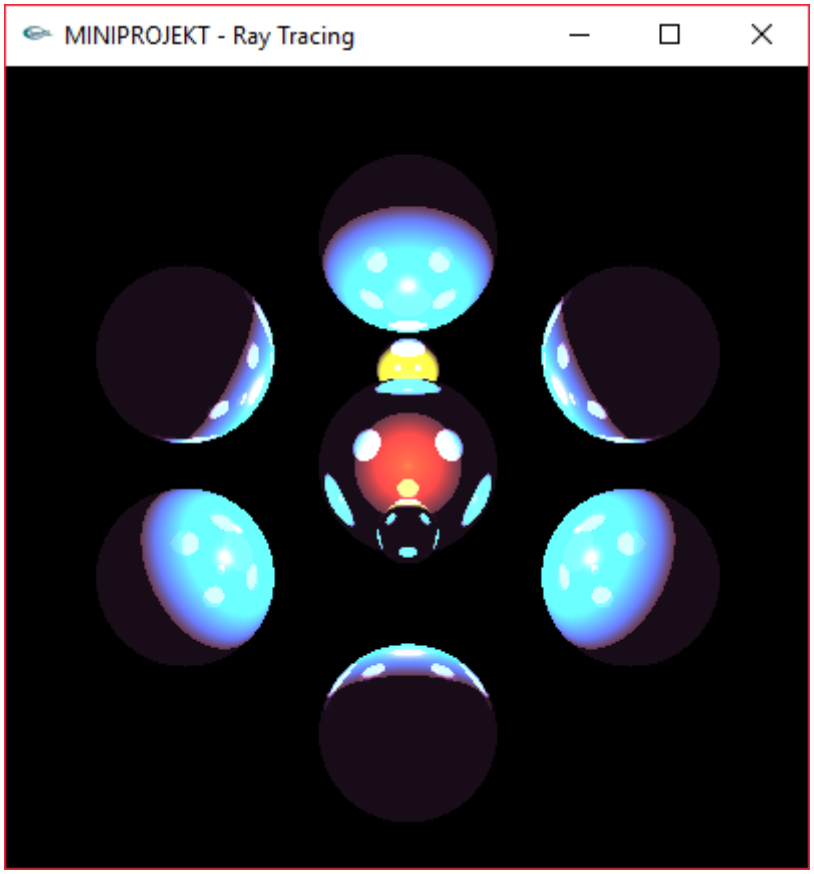
Rysunek 2: scena pierwotna - głębokość rekurencji 1



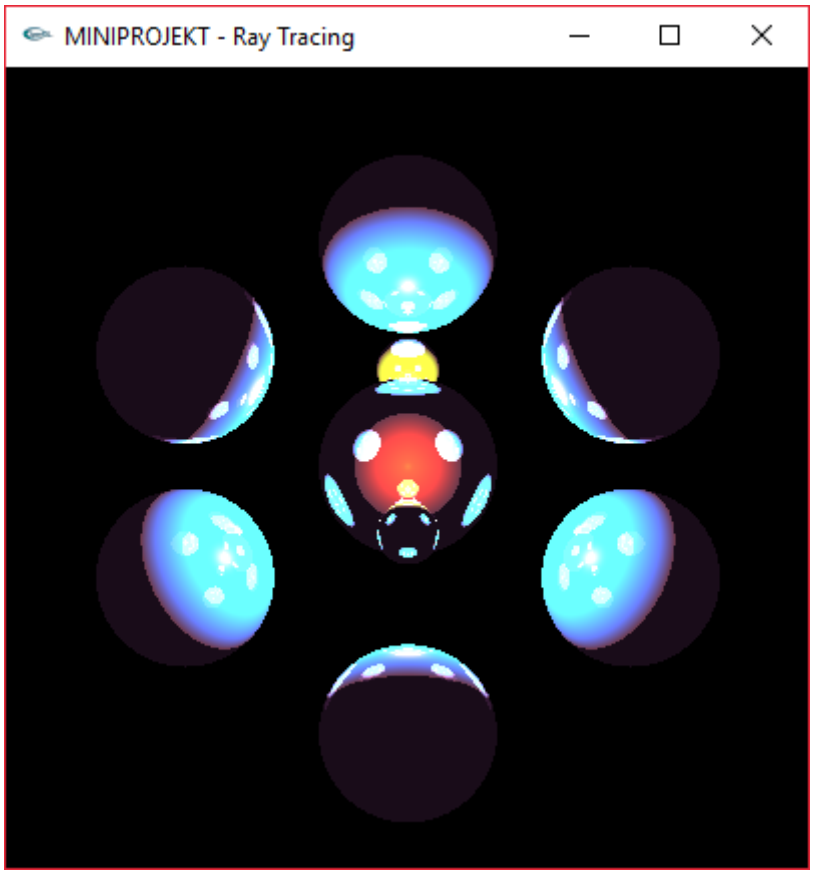
Rysunek 3: scena pierwotna - głębokość rekurencji 10

3.2. Scena zmodyfikowana 1

```
dimensions 400 400
background 0.0 0.0 0.0
global      0.1 0.1 0.1
sphere      0.7  0.0  -1.5   2.6    0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere      2.0   0.0   0.0  -1.0    0.8 0.1 0.0  0.8 0.1 0.0  0.2 0.1 0.2  40
sphere      0.7  0.0   2.2  -3.5    0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere      2.0   0.0 -6.0   4.0    0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
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
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
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
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
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
source      0.0   0.0  1.5         0.2 0.2 0.2  0.4 0.4 0.4  0.2 0.2 0.2
source      0.0   0.0  1.5         0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
source      0.0   0.0  1.5         0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
source      0.0   0.0  1.5         0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
source      0.0   0.0  1.5         0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
```



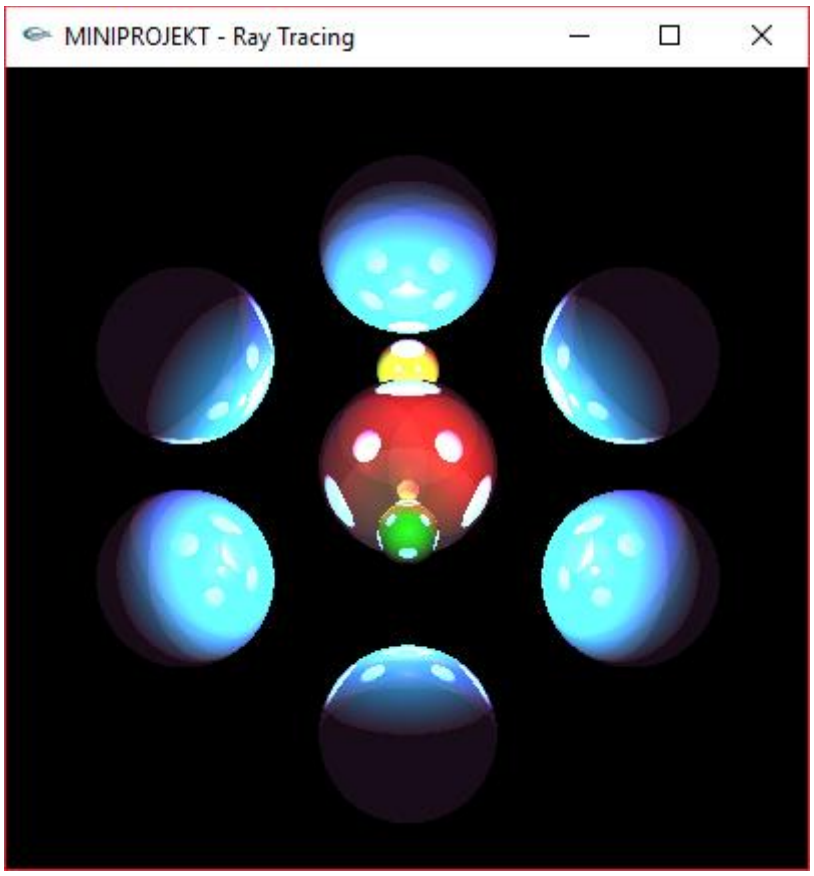
Rysunek 4: scena zmodyfikowana 1 - głębokość rekurencji 1



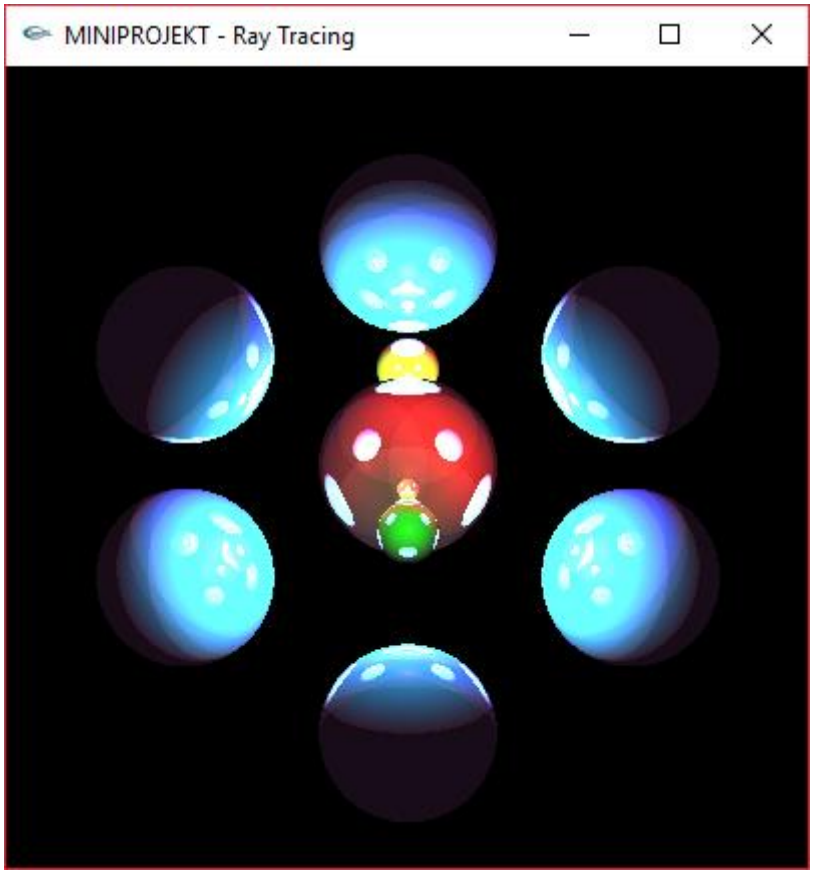
Rysunek 5: scena zmodyfikowana 1 - głębokość rekurencji 100

3.3. Scena zmodyfikowana 2

```
dimensions 400 400
background 0.0 0.0 0.0
global      0.1 0.1 0.1
sphere      0.7   0.0   -1.5  2.6     0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere      2.0   0.0    0.0 -1.0     0.8 0.1 0.0  0.8 0.1 0.0  0.2 0.1 0.2  40
sphere      0.7   0.0    2.2 -3.5     0.8 0.2 0.0  0.7 1.0 0.0  0.2 0.1 0.2  40
sphere      2.0   0.0   -6.0  4.0     0.8 0.2 0.0  0.0 0.7 1.0  0.2 0.1 0.2  40
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
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
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
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
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
source     -2.0   0.0    1.5          0.2 0.2 0.2  0.4 0.4 0.4  0.2 0.2 0.2
source      2.0   0.0    1.5          0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
source      0.0   2.0    1.5          0.2 0.2 0.2  1.0 0.0 1.0  0.3 0.3 0.1
source      0.0  -2.0    1.5          0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
source      0.0  -1.0    6.0          0.2 0.2 0.2  0.0 1.0 1.0  0.4 0.5 0.3
```



Rysunek 6: scena zmodyfikowana 2 - głębokość rekurencji 1



Rysunek 7: scena zmodyfikowana 2 - głębokość rekurencji 100

4. Podsumowanie

W niniejszym projekcie udało zrealizować się postawione zadanie. Program poprawnie wczytuje z odpowiednio sformatowanego pliku tekstowego dane wejściowe, przetwarzając je na właściwe obiekty, czego dowodzą rysunki załączone w punkcie 3. sprawozdania. Podpunkty 2.6. – 2.10. zawierają funkcje, które należało zaimplementować w programie wg instrukcji projektowej zawartej na stronie przedmiotu. Idea oraz realizacja Ray-Tracing'u zostały przybliżone we wstępie teoretycznym. Ze względu na niewystarczającą ilość czasu nie została zrealizowana implementacja własnej sceny z dodatkowymi obiektami.

Zaimplementowanie podstawowej wersji rekurencyjnego Ray-Tracing'u nie jest zadaniem trudnym, jednak programista podchodząc do niego musi zrozumieć dokładnie operacje matematyczne, które stanowią fundament dalszego działania. Bez świadomości prowadzonych obliczeń implementacja narażona jest na błędy, zwiększenie złożoności obliczeniowej, czy inne niepożądane skutki będące brakiem wiedzy o problemie. Zatem przemyślane i bazujące na znajomości zagadnienia podejście jest kluczem do dobrej implementacji algorytmu.

Na komputerze, na którym odpalany był program zbudowanie sceny w oknie zajmowało przeciętnie 5 – 7 s. Czasu tego nie można określić mianem zadowalającego, gdyż wyświetlane obiekty są statyczne. Ciężko określić wzrost nakładu czasowego dla przypadku, gdy obiekty były by dynamiczne i poruszały się po ekranie. Wysoce możliwe jest, że wówczas wystąpił by efekt wyświetlania klatkowego.

Wprowadzenie na scenę większej ilości obiektów, większej ilości źródeł światła, czy zwiększenie głębokości rekurencji skutkuje wzrostem złożoności zarówno pamięciowej, jak i przede wszystkim czasowej. Wówczas wyliczenie oświetlenia lokalnego musi być przeprowadzone dla większej liczby świateł. Promień odbity śledzony jest dokładniej, co również generuje dodatkowy nakład złożoności problemu. Co więcej, problem ten może zostać skomplikowany także ze względu na parametry światła i własności materiałowe oraz optyczne obiektów.

Na klasę złożoności algorytmu ma wpływ także bezpośrednio sposób jego implementacji, przez co rozumie się: metody przepływu danych pomiędzy funkcjami, sposób reprezentacji obiektów przestrzeni, użyte struktury danych.

Na koniec należy jeszcze przybliżyć i przeanalizować rysunki zawarte w punkcie 3. sprawozdania, które odzwierciedlają działanie Ray-Tracingu'u dla zdefiniowanych scen.

W podpunkcie 3.1. rozpatrywana jest scena zdefiniowana na stronie przedmiotu, stąd określenie ‘pierwotna’. Widać tu, że w przypadku, gdy głębokość rekurencji jest zerowa (Rys.1), czyli promień odbity nie jest śledzony, mamy do czynienia ze zwykłym oświetleniem sceny, pochodzącym wyłącznie od punktów ją oświetlających. Dla oświetlanego obiektu nie mają wówczas znaczenia wszelkie inne obiekty znajdujące się wokół niego. Gdy zaczniemy śledzić przebieg pierwszego odbicia promienia (Rys.2), to uzyskujemy pożądany efekt. Oświetlenie punktu staje się sumą wszystkich czynników zewnętrznych mających na nie wpływ.

Kolejne zwiększenie głębokości rekurencji do dziesięciu pozwala uzyskać bardziej gładkie krawędzie. W szczególności zauważalne jest to, jeśli przyjrzymy się niebieskim sferom i odbijającym się w nich pozostałym obiektom. Porównując rysunki Rys.2 i Rys. 3 widać gołym okiem, że linie obiektów odbitych są gładsze na Rys.3.

W podpunkcie 3.2. trzy niebieskie sfery zostały przesunięte wzdłuż osi Z w kierunku obserwatora. Kolejne trzy w kierunku przeciwnym. Małe sfery zostały umieszczone po przeciwległych stronach sfery środkowej, a wszystkie źródła skumulowane w punkcie pomiędzy bliższa małą sferą, a środkową. Widać tu, że promienie lokalne, a także odbite nie docierają do wszystkich punktów powierzchni obiektów. Co więcej, nawet procedura śledzenia promieni nie pomogła w oświetleniu niewidocznych obszarów.

W kolejnym podpunkcie (3.3.) położenie sfer nie uległo zmianie, jednak źródła światła zostały rozłożone na płaszczyźnie XY. Wychylenie ich z poprzedniej pozycji sprawiło, że promienie odbite od sfer niebieskich oświetliły sferę czerwoną. Ciekawym zjawiskiem jest, że tylna połówka bliższej sfery żółtej ma kolor zielony. Jest to skutek nałożenia się właściwości materiału małej sfery, odbijającego światło o barwie żółtej z promieniami światła odbijanymi od powierzchni niebieskiej.

Omawiane powyżej dwa podpunkty zostały wykonane dla dwu głębokości rekurencji: 1 i 100. W obu przypadkach zwiększenie stopnia prowadzi do otrzymania gładszych krawędzi i dokładniejszego odtworzenia charakteru obiektów. Na Rys.6 widoczne jest odbicie sfery żółtej w sferze czerwonej, ale na Rys. 7 odbicie to posiada bardziej przestrzenną strukturę. Sfera czerwona działa podobnie do lustra. Jednocześnie, co zasługuje na uwagę, na lustrzanym odbiciu sfery żółtej widoczne są dwa jasne punkty, które pochodzą z promieni odbitych od dwóch dolnych sfer niebieskich. Pomimo, że wspomniane punkty oświetlenia nie były by widoczne z perspektywy obserwatora, to są, dzięki zastosowanej metodzie śledzenia promieni.

5. Literatura

[1] Wykład 12 (hasło: lab229):

http://www.zsk.iiar.pwr.wroc.pl/zsk/dyd/intinz/gk/wyklady/

[2] Instrukcja laboratoryjna:

http://www.zsk.iiar.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_7_dz/

[3] „Introduction to Ray Tracing: a Simple Method for Creating 3D Images”:

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/how-does-it-work