pwr**POLITECHNIKA WROCŁAWSKA**

**Katedra Informatyki Technicznej**

**Zakład Systemów Komputerowych i Dyskretnych**

**Grafika komputerowa i komunikacja człowiek – komputer**

**Kurs: INEK00012L**

**Sprawozdanie z projektu**

**„Metoda śledzenia promieni (Ray Tracing)”**

|  |  |
| --- | --- |
| **Wykonał:** | **Adrian Frydmański** |
| **Termin:** | **WT/P 12:00-15:00** |
| **Data wykonania ćwiczenia:** | **24 I 2016** |
| **Data oddania sprawozdania:** | **25 I 2016** |
| **Ocena:** |  |

|  |
| --- |
| **Uwagi prowadzącego:** |

Spis treści

[Wstęp teoretyczny 2](#_Toc441468221)

[Kod źródłowy 2](#_Toc441468222)

[Działanie 9](#_Toc441468223)

[Podsumowanie 14](#_Toc441468224)

# Wstęp teoretyczny

Celem projektu było wykorzystanie metody śledzenia promieni – ray tracingu – do stworzenia obrazu trójwymiarowego.

Algorytm rekursywnego śledzenia promieni różni się od algorytmu używanego w poprzednich zadaniach. W prostym algorytmie (Ray Casting) promień biegł od obserwatora, przez punkt na rzutni w głąb sceny i był śledzony tylko do ewentualnego pierwszego przecięcia z obiektem sceny. W rekursywnej metodzie śledzenia promieni analizowany promień po trafieniu w pierwszy obiekt sceny śledzony jest dalej. Po trafieniu promienia w obiekt, wylicza się kierunek promienia odbitego i sprawdza, czy nie trafia on w kolejny obiekt itd.

# Kod źródłowy

#include "stdafx.h" // odpowiednie biblioteki w nagłówku

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// definiowanie struktur

struct something // obiekt

{

vector <float> position;

vector <float> specular;

vector <float> diffuse;

vector <float> ambient;

};

struct sphere : public something // sfera

{

float radius;

float specularshininess;

};

struct source : public something {}; // źródło światła

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Zmienne globalne

typedef float point[3]; // wektor 3 liczb

point background; // kolor tła

point globallight; // kolor globalnego oświetlenia

int im\_size\_x, im\_size\_y; // Rozmiar obrazu w pikselach

float viewport\_size\_x = 15.0; // Rozmiary okna obserwatora

float viewport\_size\_y = 15.0;

float global\_a[] = { globallight[0], globallight[1], globallight[2] }; // Parametry światła rozproszonego

float starting\_point[3]; // Parametry śledzonego promienia - punkt startowy

float starting\_directions[] = { 0.0, 0.0, -1.0 }; // i kierunek

int iter = 8; // liczba iteracji

float inter[3];

int inters;

float inters\_c[3];

float kon[] = { 0.0, 0.0, 0.0 };

float reflection\_vector[3];

float normal\_vector[3];

float reflection[3];

int typ = -1;

GLubyte pixel[1][1][3];

vector <sphere> spheres; // sfery

vector <source> sources; // źródła światła

int displayMode = 1; // tryb wyświetlania (1 - bez zmiany proporcji, 2 - dopasowanie)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja przeprowadza normalizację wektora

void Normalization(point p)

{

float d = 0.0;

int i;

for (i = 0; i<3; i++)

d += p[i] \* p[i];

d = sqrt(d);

if (d>0.0)

for (i = 0; i<3; i++)

p[i] /= d;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja oblicza iloczyn skalarny wektorów

float dotProduct(point p1, point p2)

{

return (p1[0] \* p2[0] + p1[1] \* p2[1] + p1[2] \* p2[2]);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja oblicza oświetlenie punktu na powierzchni sfery zgodnie

// z modelem Phonga

void Phong(int typ, float \*viewer\_v, float \*normal\_vector)

{

float wektor[3];

wektor[0] = normal\_vector[0];

wektor[1] = normal\_vector[1];

wektor[2] = normal\_vector[2];

float light\_vec[3]; // wektor wskazujący źródło światła

float n\_dot\_l, v\_dot\_r; // zmienne pomocnicze

inters\_c[0] = 0; inters\_c[1] = 0; inters\_c[2] = 0; //zerujemy wektory

for (int i = 0; i < sources.size(); i++)

{

light\_vec[0] = sources[i].position[0] - inter[0];

light\_vec[1] = sources[i].position[1] - inter[1];

light\_vec[2] = sources[i].position[2] - inter[2];

Normalization(light\_vec); // normalizacja wektora kierunku

// świecenia źródła

n\_dot\_l = dotProduct(light\_vec, wektor);

reflection\_vector[0] = 2 \* (n\_dot\_l)\*(wektor[0]) - light\_vec[0];

reflection\_vector[1] = 2 \* (n\_dot\_l)\*(wektor[1]) - light\_vec[1];

reflection\_vector[2] = 2 \* (n\_dot\_l)\*(wektor[2]) - light\_vec[2];

Normalization(reflection\_vector);

v\_dot\_r = dotProduct(reflection\_vector, viewer\_v);

if (n\_dot\_l > 0) // punkt jest oświetlany,

{ // oświetlenie wyliczane jest ze wzorów dla modelu Phonga

float x = sqrt((sources[i].position[0] - inter[0])\*(sources[i].position[0] - inter[0]) + (sources[i].position[1] - inter[1])\*(sources[i].position[1] - inter[1]) + (sources[i].position[2] - inter[2])\*(sources[i].position[2] - inter[2]));

inters\_c[0] += (1.0 / (1.0 + 0.01\*x + 0.001\*x\*x))\*(spheres[typ].diffuse[0] \* sources[i].diffuse[0] \* n\_dot\_l + spheres[typ].specular[0] \* sources[i].specular[0] \* pow(double(v\_dot\_r), 20.0)) + spheres[typ].ambient[0] \* sources[i].ambient[0] + spheres[typ].ambient[0] \* global\_a[0];

inters\_c[1] += (1.0 / (1.0 + 0.01\*x + 0.001\*x\*x))\*(spheres[typ].diffuse[1] \* sources[i].diffuse[1] \* n\_dot\_l + spheres[typ].specular[1] \* sources[i].specular[1] \* pow(double(v\_dot\_r), 20.0)) + spheres[typ].ambient[1] \* sources[i].ambient[1] + spheres[typ].ambient[1] \* global\_a[1];

inters\_c[2] += (1.0 / (1.0 + 0.01\*x + 0.001\*x\*x))\*(spheres[typ].diffuse[2] \* sources[i].diffuse[2] \* n\_dot\_l + spheres[typ].specular[2] \* sources[i].specular[2] \* pow(double(v\_dot\_r), 0.0)) + spheres[typ].ambient[2] \* sources[i].ambient[2] + spheres[typ].ambient[2] \* global\_a[2];

}

else // punkt nie jest oświetlany

{ // uwzględniane jest tylko światło rozproszone

inters\_c[0] += spheres[typ].ambient[0] \* global\_a[0];

inters\_c[1] += spheres[typ].ambient[1] \* global\_a[1];

inters\_c[2] += spheres[typ].ambient[2] \* global\_a[2];

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja normalizująca wektor

void Normal(int typ, float \*normal\_vector)

{

normal\_vector[0] = inter[0] - spheres[typ].position[0];

normal\_vector[1] = inter[1] - spheres[typ].position[1];

normal\_vector[2] = inter[2] - spheres[typ].position[2];

Normalization(normal\_vector);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Liczenie odbicia

void Reflect(float \*normal\_vector, float \*v)

{

float il;

float invert[3];

invert[0] = -v[0]; invert[1] = -v[1]; invert[2] = -v[2];

Normalization(invert);

il = dotProduct(invert, normal\_vector);

reflection[0] = 2 \* (il)\*normal\_vector[0] - invert[0];

reflection[1] = 2 \* (il)\*normal\_vector[1] - invert[1];

reflection[2] = 2 \* (il)\*normal\_vector[2] - invert[2];

Normalization(reflection);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Wyznaczenie współrzędnych punktu przecięcia z najbliższym obiektem sceny

int Intercest(float \*p, float \*v, int typ)

{

float r, a, b, c, d, r1, r2, min, dl, status, pier;

status = -1;

min = 0;

float tmp[3];

for (int i = 0; i < spheres.size(); i++) {

if (i != typ)

{

r = -2.0;

a = v[0] \* v[0] + v[1] \* v[1] + v[2] \* v[2];

b = 2 \* (p[0] \* v[0] + p[1] \* v[1] + p[2] \* v[2] - spheres[i].position[0] \* v[0] - spheres[i].position[1] \* v[1] - spheres[i].position[2] \* v[2]);

c = p[0] \* p[0] + p[1] \* p[1] + p[2] \* p[2] - 2 \* (spheres[i].position[0] \* p[0] + spheres[i].position[1] \* p[1] + spheres[i].position[2] \* p[2]) + spheres[i].position[0] \* spheres[i].position[0] + spheres[i].position[1] \* spheres[i].position[1] + spheres[i].position[2] \* spheres[i].position[2] - spheres[i].radius \* spheres[i].radius;

d = b\*b - 4 \* a\*c;

if (d >= 0) {

pier = sqrt(d);

r1 = (-b - pier) / (2 \* a);

r2 = (-b + pier) / (2 \* a);

if (d == 0 && r1 >= 0)

r = r1;

else

if (r1>0 || r2>0)

if (r1>r2)

if (r2>0)

r = r2;

else

r = r1;

else

if (r1>0)

r = r1;

else

r = r2;

if (r>0) {

tmp[0] = p[0] + r\*v[0];

tmp[1] = p[1] + r\*v[1];

tmp[2] = p[2] + r\*v[2];

dl = sqrt((tmp[0] - p[0])\*(tmp[0] - p[0]) + (tmp[1] - p[1])\*(tmp[1] - p[1]) + (tmp[2] - p[2])\*(tmp[2] - p[2]));

if (r < min || min == 0)

{

min = dl; status = i;

inter[0] = tmp[0];

inter[1] = tmp[1];

inter[2] = tmp[2];

}

}

}

}

}

return status;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja śledząca promień

int Trace(float \*p, float \*v, int step)

{

if (step > iter)

return 0;

typ = Intercest(p, v, typ);

if (typ >= 0)

{

Normal(typ, normal\_vector);

Reflect(normal\_vector, v);

Phong(typ, v, normal\_vector);

kon[0] += inters\_c[0];

kon[1] += inters\_c[1];

kon[2] += inters\_c[2];

Trace(inter, reflection, step + 1);

}

if (typ < 0)

{

return 0;

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja rysująca obraz oświetlonej sceny

void Display(void)

{

int x, y; // pozycja rysowanego piksela "całkowitoliczbowa"

float x\_fl, y\_fl; // pozycja rysowanego piksela "zmiennoprzecinkowa"

int im\_size\_x\_2 = im\_size\_x / 2; // połowa rozmiaru obrazu w pikselach

int im\_size\_y\_2 = im\_size\_y / 2;

if (im\_size\_x > im\_size\_y)

viewport\_size\_x = viewport\_size\_y \* im\_size\_x / im\_size\_y;

else if (im\_size\_x < im\_size\_y)

viewport\_size\_y = viewport\_size\_x \* im\_size\_y / im\_size\_x;

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

// rysowanie pikseli od lewego górnego narożnika do prawego dolnego narożnika

for (y = im\_size\_y\_2; y > -im\_size\_y\_2; y--)

{

for (x = -im\_size\_x\_2; x < im\_size\_x\_2; x++)

{

x\_fl = (float)x / (im\_size\_x / viewport\_size\_x);

y\_fl = (float)y / (im\_size\_y / viewport\_size\_y);

// przeliczenie pozycji(x,y) w pikselach na pozycję

// "zmiennoprzecinkową" w oknie obserwatora

starting\_point[0] = x\_fl;

starting\_point[1] = y\_fl;

starting\_point[2] = viewport\_size\_x;

// wyznaczenie początku śledzonego promienia dla rysowanego piksela

kon[0] = 0.0; kon[1] = 0.0; kon[2] = 0.0;

Trace(starting\_point, starting\_directions, 1);

if (kon[0] == 0.0) kon[0] = background[0];

if (kon[1] == 0.0) kon[1] = background[1];

if (kon[2] == 0.0) kon[2] = background[2];

// obliczenie punktu przecięcia ze sferą

// konwersja wartości wyliczonego oświetlenia na liczby od 0 do 255

if (kon[0] > 1) // składowa czerwona R

pixel[0][0][0] = 255;

else

pixel[0][0][0] = kon[0] \* 255;

if (kon[1] > 1) // składowa zielona G

pixel[0][0][1] = 255;

else

pixel[0][0][1] = kon[1] \* 255;

if (kon[2] > 1) // składowa niebieska B

pixel[0][0][2] = 255;

else

pixel[0][0][2] = kon[2] \* 255;

glRasterPos3f(x\_fl, y\_fl, 0);

// inkrementacja pozycji rastrowej dla rysowania piksela

glDrawPixels(1, 1, GL\_RGB, GL\_UNSIGNED\_BYTE, pixel);

// Narysowanie kolejnego piksela na ekranie

}

}

glFlush();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Zmiana rozmiaru okna

void ChangeSize(GLsizei horizontal, GLsizei vertical)

{

GLfloat AspectRatio; // proporcja wymiarów okna

if (vertical == 0) // Zabezpieczenie przed dzieleniem przez 0

vertical = 1;

glViewport(0, 0, horizontal, vertical); // Ustawienie wielkoœciokna okna widoku (viewport)

glMatrixMode(GL\_PROJECTION); // Przełączenie macierzy bieżącej na macierz projekcji

glLoadIdentity(); // Czyszcenie macierzy bieżącej

AspectRatio = (GLfloat)horizontal / (GLfloat)vertical; // Wyznaczenie współczynnika proporcji okna

switch (displayMode) // zastosowanie wybranego przez użytkownika trybu wyświetlania

{

case 1:

if (horizontal <= vertical)

glOrtho(-viewport\_size / 2, viewport\_size / 2, -viewport\_size / AspectRatio / 2, viewport\_size / AspectRatio / 2, viewport\_size, -viewport\_size);

else

glOrtho(-viewport\_size\*AspectRatio / 2, viewport\_size\*AspectRatio / 2, -viewport\_size / 2, viewport\_size / 2, viewport\_size, -viewport\_size);

break;

case 2:

glOrtho(-viewport\_size / 2, viewport\_size / 2, -viewport\_size / 2, viewport\_size / 2, -viewport\_size / 2, viewport\_size / 2);

break;

}

glMatrixMode(GL\_MODELVIEW); // Przełączenie macierzy bieżącej na macierz widoku modelu

glLoadIdentity(); // Czyszczenie macierzy bieżącej

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Obsługa klawiszy

void Keys(unsigned char key, int x, int y)

{

// dla klawiszy 1 i 2 zmiana trybu wyœwietlania, gdy jest inny ni¿ wciœniêty

if ((key == '1' || key == '2') && key - '0' != displayMode)

{

displayMode = key - '0'; //zmiana trybu wyświetlania

ChangeSize(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT)); // wywołanie funkcji zmieniającej proporcje sceny

Display(); // przerysowanie sceny

}

else if (key == '+')

{

iter++; // zwiększenie liczby iteracji

Display(); // przerysowanie sceny

}

else if (key == '-' && iter > 0)

{

iter--; // zmniejszenie liczby iteracji

Display(); // przerysowanie sceny

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Odczytywanie danych z pliku

void pobierzDane(string name)

{

if (name.length() == 0)

name = "scene.txt";

fstream plik(name, ios::in);

if (!plik)

cerr << "brak pliku 'scene.txt'";

cout << "Ray Tracing\nAdrian Frydmański (209865)\n----Window:--------------------------------------------------------------------\n";

char buf[20];

float liczba;

while (!plik.eof())

{

plik >> buf;

if (buf[0] == 'd') // dimensions

{

plik >> liczba;

im\_size\_x = liczba;

plik >> liczba;

im\_size\_y = liczba;

cout << "Dimensions = (" << im\_size\_x << "," << im\_size\_y << ")\nBackground = (";

}

else if (buf[0] == 'b') // background

{

for (int i = 0; i < 3; i++)

{

plik >> liczba;

background[i] = liczba;

cout << background[i] << ",";

}

cout << ")\nGlobal = (";

}

else if (buf[0] == 'g') // globat

{

for (int i = 0; i < 3; i++)

{

plik >> liczba;

globallight[i] = liczba;

cout << globallight[i] << ",";

}

cout << ")\n-------------------------------------------------------------------------------\n"

" r-radius, p-position, s-specular, d-diffuse, a-ambient, ss-specularshininess\n"

"----Objects:-------------------------------------------------------------------\n";

}

else if (buf[0] == 's')

if (buf[1] == 'p') // sfera

{

sphere s;

plik >> liczba;

s.radius = liczba; // radius

cout << "Sphere " << spheres.size() + 1 << ": r=" << s.radius << " p=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.position.push\_back(liczba); // position

cout << s.position[i];

if (i < 2)

cout << ",";

}

cout << ") s=(";

for (int i = 0; i<3; i++)

{

plik >> liczba;

s.specular.push\_back(liczba); // specular

cout << s.specular[i];

if (i < 2)

cout << ",";

}

cout << ") d=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.diffuse.push\_back(liczba); // diffuse

cout << s.diffuse[i];

if (i < 2)

cout << ",";

}

cout << ") a=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.ambient.push\_back(liczba); // ambient

cout << s.ambient[i];

if (i < 2)

cout << ",";

}

plik >> liczba;

s.specularshininess = liczba; // specularshininess

cout << ") ss=" << s.specularshininess << "\n";

spheres.push\_back(s); // dodaj sferę do vectora

}

else if (buf[1] == 'o')

{

source s;

cout << "Source " << sources.size()+1 << ": p = (";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.position.push\_back(liczba); // position

cout << s.position[i];

if (i < 2)

cout << ",";

}

cout << ") s=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.specular.push\_back(liczba); // specular

cout << s.specular[i];

if (i < 2)

cout << ",";

}

cout << ") d=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.diffuse.push\_back(liczba); // diffuse

cout << s.diffuse[i];

if (i < 2)

cout << ",";

}

cout << ") a=(";

for (int i = 0; i < 3; i++)

{

plik >> liczba;

s.ambient.push\_back(liczba); // ambient

cout << s.ambient[i];

if (i < 2)

cout << ",";

}

cout << ")\n";

sources.push\_back(s); // dodaj źródło światła do vectora

}

}

plik.close();

cout << "-------------------------------------------------------------------------------\n";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Główna funkcja

void main(void)

{

setlocale(LC\_ALL, "");

pobierzDane("");

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);

glutInitWindowSize(im\_size\_x, im\_size\_y);

glutCreateWindow("Ray Tracing - 209865");

glutDisplayFunc(Display);

glutKeyboardFunc(Keys);

glutReshapeFunc(ChangeSize);

glutMainLoop();

}

# Działanie

Program wczytuje odpowiednio przygotowany plik. W sprawozdaniu wyniki działania programu będą opierać się o następujące dane wejściowe w pliku scene.txt:

dimensions 800 800

background 0.1 0.1 0.1

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

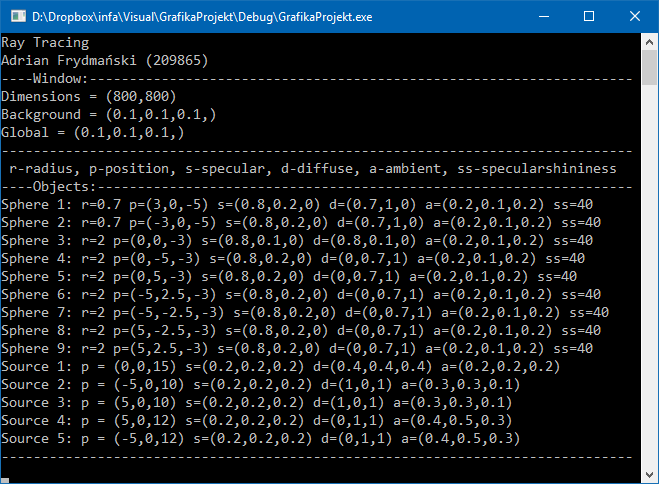
source 5.0 0.0 10.0 0.2 0.2 0.2 1.0 0.0 1.0 0.3 0.3 0.1

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

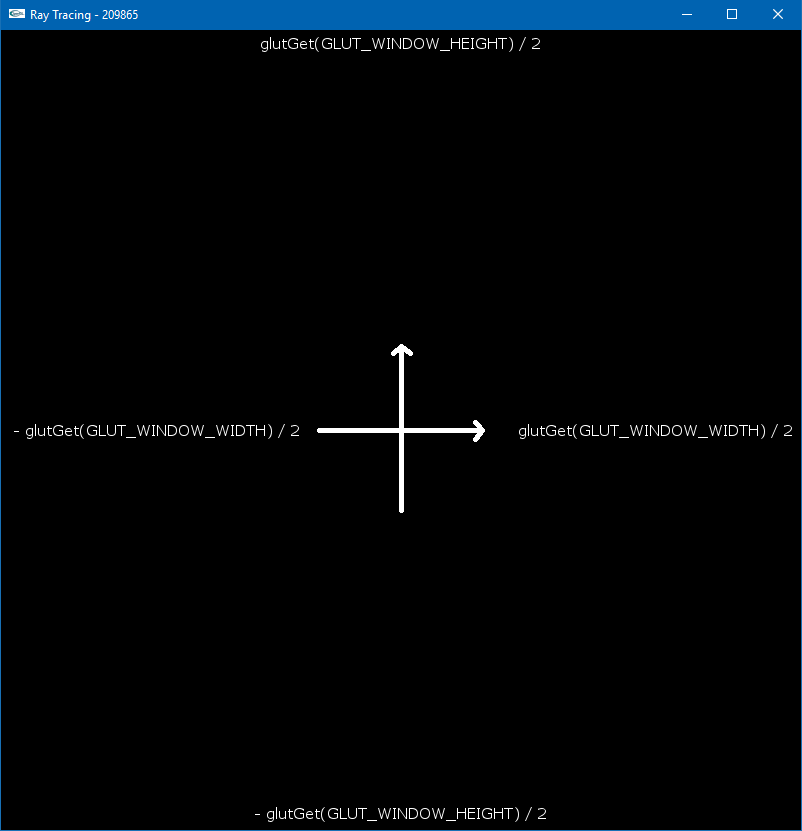
Na początku zostają podane wymiary okna, szerokość i wysokość, kolor tła i kolor globalnego oświetlenia. Następnie, w zależności od pierwszego wyrazu linii, odczytywane są dane sfery (promień, współrzędne środka, współczynniki materiałowe powierzchni dla światła otoczenia, światła rozproszonego i kierunkowego oraz współczynnik połysku), lub źródła światła (współrzędne punktu, w którym umieszczone jest źródło, składowe kolorów określające intensywności świecenia źródła dla światła otoczenia, światła rozproszonego i kierunkowego).

Po uruchomieniu pliki zostają wczytane:



Rysunek 1 Konsolowe okno programu

Po wczytaniu danych rysowanie nadzoruje funkcja Display. Piksele rysowane są od lewego górnego do prawego dolnego narożnika. Granice pętli od –x/2 i y/2 do x/2 i –y/2 sprawiają, że środek wyświetlanego obrazu jest w środku okna, jak widać na poniższym rysunku:



Rysunek 2 Granice pętli

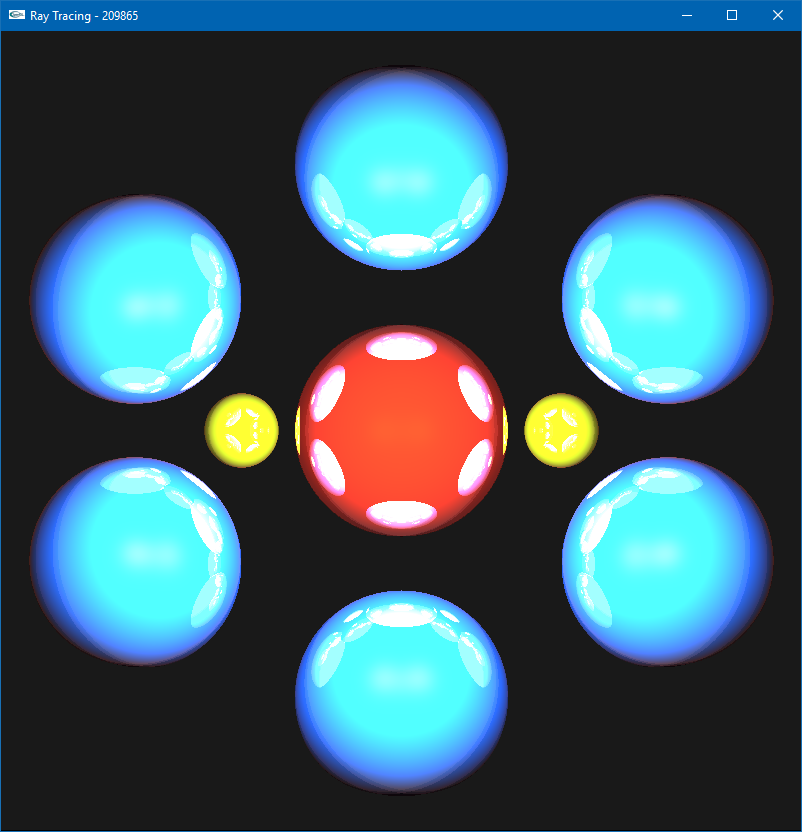
Cały algorytm rysowania jest opisany na poniższym schemacie:



Rysunek 3 Kroki algorytmu rysującego obraz metodą śledzenia promieni

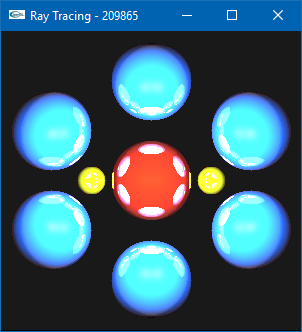
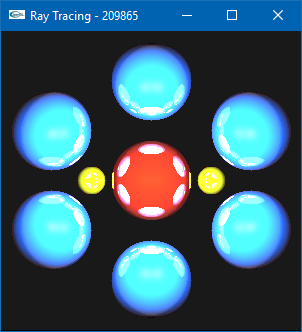
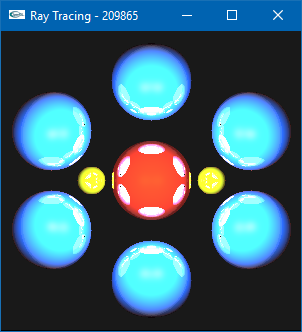
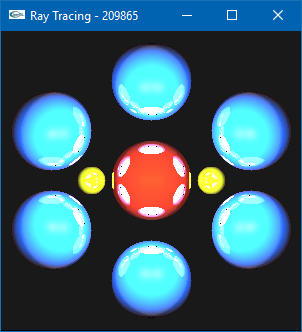
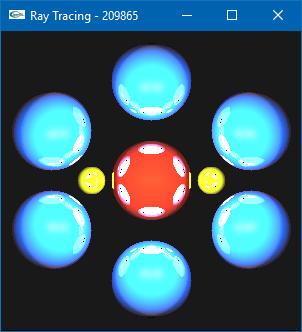
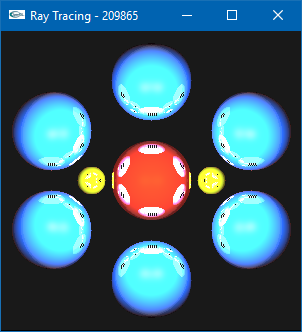
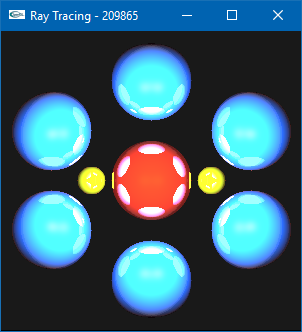
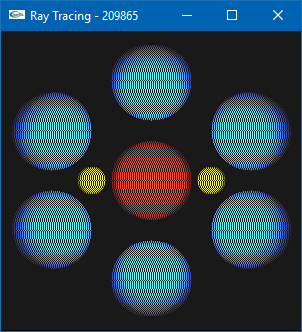
Program w funkcji zwrotnej dla klawiszy pozwala na zmianę proporcji obrazu z kwadratu na rozciągnięcie do granic okna (klawisze 1 i 2). Ponadto możliwa jest zmiana liczby iteracji w śledzeniu promienia (klawisze + i -).

Zrenderowany obraz o wymiarach 800x800 wygląda następująco:



Rysunek 4 Obraz stworzony na podstawie wczytanego pliku

Liczba iteracji w śledzeniu promienia ma wpływ na otrzymany obraz, co ilustrują poniższe przykłady.



Rysunek 5 Renderowany obraz 300x300 w zależności od liczby iteracji (od 1 do 8)

Dla zbyt „płytkich” rekurencji – ze zbyt małym ograniczeniem iteracji – promień nie trafia na kolejny obiekt i śledzenie się kończy. Widać wtedy prześwitujące w dziwny sposób tło zamiast oświetlonej powierzchni obiektu. Już dla dwóch iteracji efekt ten maleje i jest znikomy. Dla 8 iteracji obraz można uznać za ładny.

Jak łatwo się domyślić, wraz ze wzrostem liczby iteracji, liniowo rośnie czas wykonywania się algorytmu.

# Podsumowanie

Ćwiczenie pokazało, w jaki sposób tworzony jest obraz w metodzie śledzenia promieni i trudności z nim związane.