POLITECHNIKA WROCŁAWSKA

Katedra Informatyki technicznej

Zakład Systemów Komputerowych i Dyskretnych

Wprowadzenie do grafiki komputerowej

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 7

TEMAT ĆWICZENIA:

Metoda śledzenia promieni (Ray Tracing) - projekt

Wykonał:	**Maciej Konefał (209885)**
Termin:	**WT/NP 15.00-18.00**
Data wykonania ćwiczenia:	**23.01.2015**
Data oddania sprawozdania:	**24.01.2015**
Ocena:	

Uwagi prowadzącego:

Spis treści

Wstęp.....2
Opis implementacji.....2
Kod źródłowy zrealizowanego programu.....3
Wnioski.....10
Materiały pomocnicze.....10

1. Wstęp

Głównym zadaniem projektowym było zaimplementowanie algorytmu rekursywnego śledzenia promieni (Recursive Ray Tracing) na złożonej scenie. W tym celu należało oświetlić kilka sfer, za pomocą wielu źródeł światła. Wprowadzenie do projektu polegało na zapoznaniu się z uproszczoną metodą śledzenia promieni (Ray Casting). Opisy w instrukcji laboratoryjnej zawierały niezbędne informacje na temat poprawnej implementacji poszczególnych funkcji programu. Ostatecznie należało przygotować scenę zgodną z efektem przedstawionym w instrukcji, uwzględniając kolejne założenia.

Mianowicie, scena zbudowana jest z dziewięciu sfer, oświetlonych kilkoma źródłami światła, natomiast przyjęty sposób rzutowania jest równoległy. Współczynniki **a**, **b**, **c** określające wpływ odległości źródła światła na oświetlenie punktu powierzchni mogą być wpisane w kodzie programu. Dodatkowo opis sceny zadany zostaje za pomocą pliku tekstowego, gdzie każda z linii zawiera słowo kluczowe i odpowiednie dane.

Co istotne, ostateczny program realizował wyznaczone cele i z sukcesem zaimplementowano algorytm rekursywnego śledzenia promieni. Kolejne kroki i uzyskany rezultat zostały przedstawione w poniższych punktach.

2. Opis implementacji

W porównaniu z uproszczoną metodą przedstawioną we wstępie, algorytm Recursive Ray Tracing ma inny zakres śledzenia promienia światła. W Ray Casting'u promień biegnący od obserwatora, przez punkt na rzutni w głąb sceny, śledzony był jedynie do pierwszego przecięcia z obiektem sceny. Z kolei implementowana w projekcie metoda śledzenia promieni, przeprowadza analizę promienia od trafienia w pierwszy obiekt sceny dalej - zależnie od ograniczenia. Zatem po przecięciu promienia z obiektem, wylicza się kierunek promienia odbitego i sprawdza, czy nie trafia on w kolejny obiekt itd.

W niniejszym rozwiązaniu najważniejszą rolę pełni rekurencyjna funkcja `Trace()`. Oblicza ona barwę piksela dla promienia zaczynającego się w zadanym punkcie p. Wspomniany punkt jest argumentem funkcji `Trace()` i biegnie w kierunku wyznaczonym przez drugi argument funkcji tj. wektor v. Ilość rekurencyjnych przebiegów określa z kolei trzeci parametr funkcji - `step`. W ciele metody początkowo weryfikowany jest zdarzenie określające przecięcie promienia z jakimkolwiek obiektem na scenie (funkcja `Intersect()` i warunek). W momencie gdy miało miejsce przecięcie, to wówczas następuje określenie wektora normalnego do powierzchni obiektu w danym punkcie (przy pomocy funkcji `Normal`()), w dalszej części wyliczenie wektora kierunku odbicia promienia (używając funkcji `Reflect`(), ostatecznie definiując oświetlenia punktu powierzchni.

Końcowy mechanizm wykonuje funkcja `Phong()`, mająca za zadanie określenie oświetlenia punktu przy użyciu modelu Phonga. W tym celu przyjmuje ona dwa argumenty, a mianowicie numer obiektu i wektor kierunkowy promienia. Na poprzednich laboratoriach zapoznano się ze wspomnianym Modelem, który służy do wyznaczania oświetlenia punktu znajdującego się na powierzchni obiektu 3-D przy czym istnieje możliwość bardzo precyzyjnego zdefiniowania pożądanego zjawiska używając trzech składowych R, G, B.

Zgodnie z ostatnim zaleceniem dotyczącym możliwości zadania sceny za pomocą wczytanego pliku, kolejne linie pliku tekstowego powinny zawierać parametry opisane w tabeli 1.

Tabela 1. Zawartość poszczególnych linii pliku opisującego zawartość sceny.

Słowo kluczowe	Opis wraz z przykładem
dimensions	Opis wymiaru sceny, szerokość wraz z wysokością. Np. dimensions 450, 450
background	Opis koloru tła oraz składowych R, G i B. Np.: background 0.3, 0.3, 0.3
global	Wartości definiujące globalne światło rozproszone, wraz ze składowymi R, G, B intensywności świecenia źródła światła. Np.: global 0.2, 0.2, 0.2
sphere	Wartości definiujące sferę. Kolejno promień, współrzędne środka oraz współczynniki materiałowe powierzchni światła otoczenia, światła rozproszonego, kierunkowego. współczynnik połysku, Np.: sphere 0.7 3.0 0.0 -5.0 0.8 0.2 0.0 0.7 1.0 0.0 0.3 0.2 0.2 40
source	Dane opisujące źródło światła. Współrzędne punktu źródła, składowe: R, G i B określające intensywności świecenia źródła dla światła otoczenia, światła rozproszonego i kierunkowego. Np.: 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

3. Kod źródłowy zrealizowanego programu

Listing 1. Kod źródłowy wykonanego projektu.

```
/*****************************************************************************/
// GRAFIKA KOMPUTEROWA I KOMUNIKACJA CZLOWIEK-KOMPUTER
//
// Maciej Konefał
// Nr indeksu: 209885
//
// Ćwiczenie 7 - Metoda śledzenia promieni (Ray Tracing)
// WT_TN_15
/*****************************************************************************/
#include <windows.h>
#include <math.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

typedef float point3[3];                // Definicja typu reprezentujacego punkt 3D

/*****************************************************************************/
// Zmienne globalne
/*****************************************************************************/
int imageSize = 400;                    // Window width&height (pix)
float viewArea = 20.0;                  // Rozmiar okna obserwatora

float lightPosition[10][3];             //// Parametry zrodla swiatla
float lightSpecular[10][3];
```

```
float lightDiffuse[10][3];
float lightAmbient[10][3];

float sphereRadius[10];                 //// Parametry sfery, ktora jest rysowana
float sphereDiffuse[10][3];
float sphereAmbient[10][3];
float spherePosition[10][3];
float sphereSpecular[10][3];
float sphereSpecularhininess[10];

point3 globalAmbient;                   // Parametry swiatla rozproszonego
                                        //// Parametry sledzonego promienia:
point3 startingPoint;                   // punkt, generujacy promien wychodzacy
point3 startingDir = { 0.0, 0.0, -1.0 }; // wektor okreslajacy kierunek promienia

point3 normalVector;                    // Wektor normalny do powierzchni
point3 reflectionVector;                // Wektor odbijajacy
                                        //// Dodatkowe zmienne pomocnicze:
point3 intersPoint;                     // Wspolrzedne punktu przeciecia sfery i promienia (x,y,z)
point3 intersColor;                     //Skladowe koloru dla oswietlonego pkt na powierzchni sfery
point3 color;
point3 backColor;                       // Barwa tla wczytana z pliku
GLubyte pixel[1][1][3];                 // Skladowe koloru rysowanego piksela

int number = 1,
spheres = 0,                            // Liczba sfer wczytana z pliku
lights = 0;                             // Liczba zrodel swiatla wczytana z pliku
int    limit = 50;                      // Graniczna ilosc iteracji

/*****************************************************************************/
// Deklaracje funkcji używanych w programie
/*****************************************************************************/
void ReadSceneFromFile(string fileName); // Funkcja wczytuje zadane informacje z pliku

int Intersect(point3 p, point3 v);      // Funkcja okreslajaca wspolrzedne punkt przeciecia
promienia oraz obiektu

void Phong(int nr, float *viewerVec);   // Funkcja wyznacza oswietlenie punktu na powierzchni sfery
zgodnie z modelem Phonga

void Trace(point3 p, point3 v, int step);       // Funkcja wyznacza kolor piksela dla promienia
zaczynajacego sie w punkcie p i biegnacego w kierunku wskazywanym przez wektor v

void Normalization(point3 p);           // Funkcja wykonujaca normalizacje wektora

float Scalar(point3 p1, point3 p2);     // Funkcja obliczajaca iloczyn skalarny dwoch wektorow

void Normal(int nr);                    // Funkcja wyznaczajaca wektor normalny w punkcie

void Reflect(point3 v);                 // Funkcja oblicza kierunek odbicia promienia w punkcie

void Myinit(void);                      // Funkcja inicjalizujaca, okreslajaca sposob rzutowania
void RenderScene(void);                 // Funkcja rysujaca obraz oswietlonej sceny
/*****************************************************************************/
// Funkcja glowna
/*****************************************************************************/
void main(void)
{
        string fileName = "";
        cout << " ____________________________________________________ " << endl;
        cout << "| GRAFIKA KOMPUTEROWA I KOMUNIKACJA CZLOWIEK-KOMPUTER |" << endl;
        cout << "|____________________________________________________|" << endl;
        cout << "|                                                    |" << endl;
        cout << "| PROJEKT: Metoda sledzenia promieni (Ray Tracing)   |" << endl;
        cout << "| Maciej Konefal 209885                              |" << endl;
        cout << "|____________________________________________________|" << endl << endl;
```

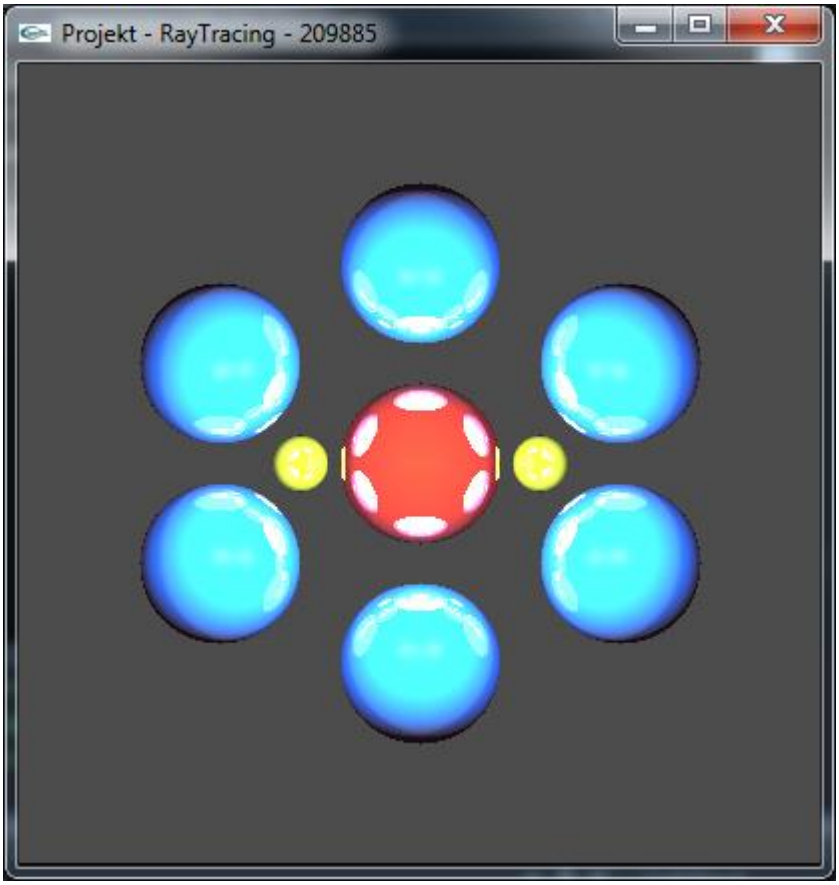
```
        cout << "  Podaj nazwe pliku z opisem sceny (+txt): ";
        cin >> fileName;

    ReadSceneFromFile(fileName);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(imageSize, imageSize);
    glutCreateWindow("Projekt - RayTracing");
    Myinit();
    glutDisplayFunc(RenderScene);
    glutMainLoop();
}

/*********************************************************************************/
// Funkcja wczytuje zadane informacje z pliku
void ReadSceneFromFile(string fileName)
{
        string buffer = "";
        float value = 0.0;
        int i = 0;

        fstream file(fileName.c_str(), ios::in);
        if (!file.is_open())
        {
                cout << "Brak pliku ze scena (" << fileName << ")" << endl;
                system("PAUSE");
                exit(0);
        }
        while (!file.eof())
        {
                file >> buffer;

                if (buffer == "dimensions")                     //Rozmiar okna
                {
                        file >> value;
                        imageSize = value;
                }
                if (buffer == "background")                     //Kolor tla
                {
                        for (i = 0; i<3; i++) {
                                file >> value;
                                backColor[i] = value;
                        }
                }
                if (buffer == "global")                         //Parametry swiatla rozproszonego
                {
                        for (i = 0; i<3; i++) {
                                file >> value;
                                globalAmbient[i] = value;
                        }
                }
                while (buffer == "sphere" && !file.eof())       //Parametry rysowanej sfery
                {
                        file >> value;
                        sphereRadius[spheres] = value;
                        for (i = 0; i<3; i++) {
                                file >> value;
                                spherePosition[spheres][i] = value;
                        }
                        for (i = 0; i<3; i++) {
                                file >> value;
                                sphereSpecular[spheres][i] = value;
                        }
                        for (i = 0; i<3; i++) {
                                file >> value;
                                sphereDiffuse[spheres][i] = value;
```

```
                }
                for (i = 0; i<3; i++) {
                        file >> value;
                        sphereAmbient[spheres][i] = value;
                }
                file >> value;
                sphereSpecularhininess[spheres] = value;
                spheres++;
                file >> buffer;
        }
        while (buffer == "source" && !file.eof())       //Parametry zrodel swiatla
        {
                for (i = 0; i<3; i++) {
                        file >> value;
                        lightPosition[lights][i] = value;
                }
                for (i = 0; i<3; i++) {
                        file >> value;
                        lightSpecular[lights][i] = value;
                }
                for (i = 0; i<3; i++) {
                        file >> value;
                        lightDiffuse[lights][i] = value;
                }
                for (i = 0; i<3; i++) {
                        file >> value;
                        lightAmbient[lights][i] = value;
                }
                lights++;
                file >> buffer;
        }
    }
    file.close();
}

/*****************************************************************************************/
// Funkcja okreslajaca wspolrzedne punktu przeciecia promienia oraz obiektu
int Intersect(point3 p, point3 v) {
    float r, a, b, c, d;
    float length = 1000000000000;
    int status = -1;

    for (int i = 0; i<spheres; i++) {
        a = v[0] * v[0] + v[1] * v[1] + v[2] * v[2];
        b = 2 * (v[0] * (p[0] - spherePosition[i][0]) + v[1] * (p[1] - spherePosition[i][1]) +
v[2] * (p[2] - spherePosition[i][2]));
        c = p[0] * p[0] + p[1] * p[1] + p[2] * p[2] - 2 * (spherePosition[i][0] * p[0] +
spherePosition[i][1] * p[1] + spherePosition[i][2] * p[2]) + spherePosition[i][0] *
spherePosition[i][0] + spherePosition[i][1] * spherePosition[i][1] + spherePosition[i][2] *
spherePosition[i][2] - sphereRadius[i] * sphereRadius[i];
        d = b*b - 4 * a*c;
        if (d >= 0) {
                r = (-b - sqrt(d)) / (2 * a);

                if (r > 0 && r < length) {
                        intersPoint[0] = p[0] + r*v[0];
                        intersPoint[1] = p[1] + r*v[1];
                        intersPoint[2] = p[2] + r*v[2];
                        length = sqrt((intersPoint[0] - p[0])*(intersPoint[0] - p[0]) +
(intersPoint[1] - p[1])*(intersPoint[1] - p[1]) + (intersPoint[2] - p[2])*(intersPoint[2] - p[2]));
                        status = i;
                }
        }
    }
    return status;
}
```

```
/*****************************************************************************/
// Funkcja wyznacza kolor piksela dla promienia zaczynajacego sie w punkcie p i biegnacego w
kierunku wskazywanym przez wektor v
void Trace(point3 p, point3 v, int step)
{
	if (step > limit)
		return;

	number = Intersect(p, v);
	if (number >= 0) {
		Normal(number);
		Reflect(v);
		Phong(number, v);
		color[0] += intersColor[0];
		color[1] += intersColor[1];
		color[2] += intersColor[2];
		Trace(intersPoint, reflectionVector, step + 1);
	}
	else
		return;
}

/*****************************************************************************/
// Funkcja wyznacza oswietlenie punktu na powierzchni sfery zgodnie z modelem Phonga
void Phong(int nr, point3 viewerVec)
{
	point3 reflectionVec;					// wektor kierunku odbicia swiatla
	point3 lightVec;					// wektor wskazujacy zrodlo swiatla
	float n_dot_l, v_dot_r;					// dodatkowe zmienne pomocnicze

	point3 viewer = { -viewerVec[0], -viewerVec[1], -viewerVec[2] };

	intersColor[0] = 0;
	intersColor[1] = 0;
	intersColor[2] = 0;

	for (int i = 0; i<lights; i++) {
		lightVec[0] = lightPosition[i][0] - intersPoint[0]; // wektor wskazujacy kierunek
zrodla swiatla
		lightVec[1] = lightPosition[i][1] - intersPoint[1];
		lightVec[2] = lightPosition[i][2] - intersPoint[2];

		Normalization(lightVec);   // Normalizacja wektora kierunku swiecenia zrodla swiatla

		n_dot_l = Scalar(lightVec, normalVector);

		reflectionVec[0] = 2 * (n_dot_l)*normalVector[0] - lightVec[0];
		reflectionVec[1] = 2 * (n_dot_l)*normalVector[1] - lightVec[1];
		reflectionVec[2] = 2 * (n_dot_l)*normalVector[2] - lightVec[2];

		// obliczenie wektora kieruneku swiatla odbitego od punktu na powierzchni sfery

		Normalization(reflectionVec);     // normalizacja wektora kierunku swiatla odbitego

		v_dot_r = Scalar(reflectionVec, viewer);

		if (v_dot_r < 0)                  // obserwator nie widzi oswietlanego punktu
			v_dot_r = 0;

		// sprawdz czy punkt na powierzchni sfery jest oswietlany przez zrodlo swiatla

		if (n_dot_l > 0)                  // Punkt oswietlany, za pomoca modelu Phonga
		{
```

```
                    float x = sqrt((lightPosition[i][0] - intersPoint[0])*(lightPosition[i][0] -
intersPoint[0]) + (lightPosition[i][1] - intersPoint[1])*(lightPosition[i][1] - intersPoint[1]) +
(lightPosition[i][2] - intersPoint[2])*(lightPosition[i][2] - intersPoint[2]));
                    intersColor[0] += (1.0 / (1.0 + 0.01*x + 0.001*x*x))*(sphereDiffuse[nr][0] *
lightDiffuse[i][0] * n_dot_l + sphereSpecular[nr][0] * lightSpecular[i][0] * pow(double(v_dot_r),
(double)sphereSpecularhinineess[nr])) + sphereAmbient[nr][0] * lightAmbient[i][0];
                    intersColor[1] += (1.0 / (1.0 + 0.01*x + 0.001*x*x))*(sphereDiffuse[nr][1] *
lightDiffuse[i][1] * n_dot_l + sphereSpecular[nr][1] * lightSpecular[i][1] * pow(double(v_dot_r),
(double)sphereSpecularhinineess[nr])) + sphereAmbient[nr][1] * lightAmbient[i][1];
                    intersColor[2] += (1.0 / (1.0 + 0.01*x + 0.001*x*x))*(sphereDiffuse[nr][2] *
lightDiffuse[i][2] * n_dot_l + sphereSpecular[nr][2] * lightSpecular[i][2] * pow(double(v_dot_r),
(double)sphereSpecularhinineess[nr])) + sphereAmbient[nr][2] * lightAmbient[i][2];
            }
            else
            // punkt nie jest oswietlany, uwzgledniane jedynie swiatlo rozproszone
            intersColor[0] += sphereAmbient[nr][0] * globalAmbient[0];
            intersColor[1] += sphereAmbient[nr][1] * globalAmbient[1];
            intersColor[2] += sphereAmbient[nr][2] * globalAmbient[2];
      }
}

/*****************************************************************************/
// Funkcja wyznaczajaca wektor normalny w punkcie
void Normalization(point3 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 obliczajaca iloczyn skalarny dwoch wektorow
float Scalar(point3 p1, point3 p2)
{
      float res = p1[0] * p2[0] + p1[1] * p2[1] + p1[2] * p2[2];
      return res;
}

/*****************************************************************************/
// Funkcja oblicza kierunek odbicia promienia w punkcie
void Reflect(point3 v) {
      float  n_dot_i;
      float   invert[3] = { -v[0], -v[1], -v[2] };

      Normalization(invert);

      n_dot_i = Scalar(invert, normalVector);
      reflectionVector[0] = 2 * (n_dot_i)*normalVector[0] - invert[0];
      reflectionVector[1] = 2 * (n_dot_i)*normalVector[1] - invert[1];
      reflectionVector[2] = 2 * (n_dot_i)*normalVector[2] - invert[2];

      Normalization(reflectionVector);
}
```

```
/*************************************************************************************/
// Funkcja wyznaczajaca wektor normalny w punkcie
void Normal(int nr)
{
	normalVector[0] = intersPoint[0] - spherePosition[nr][0];
	normalVector[1] = intersPoint[1] - spherePosition[nr][1];
	normalVector[2] = intersPoint[2] - spherePosition[nr][2];

	Normalization(normalVector);
}


/*************************************************************************************/
// Funkcja inicjalizujaca, okreslajaca sposob rzutowania
void Myinit(void)
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-viewArea / 2, viewArea / 2, -viewArea / 2, viewArea / 2, -viewArea / 2, viewArea /
2);
	glMatrixMode(GL_MODELVIEW);
}

/*************************************************************************************/
// Funkcja rysujaca obraz oswietlonej sceny
void RenderScene(void)
{
	int  x, y;				// Calkowitoliczbowe wspolrzedne rysowanego piksela
	float x_fl, y_fl;			// Zmiennoprzecinkowe wspolrzedne rysowanego piksela
	int imageSize_2;			// Polowa rozmiaru obrazu w pikselach

	imageSize_2 = imageSize / 2;		// Wyznaczenie polowy rozmiaru obrazu w pikselach
	glClear(GL_COLOR_BUFFER_BIT);
	glFlush();

	for (y = imageSize_2; y > -imageSize_2; y--)	// Rysowanie - przekatna w prawo i w dol
	{
		for (x = -imageSize_2; x < imageSize_2; x++)
		{
			x_fl = (float)x / (imageSize / viewArea);
			y_fl = (float)y / (imageSize / viewArea);

			startingPoint[0] = x_fl;	//// Wspolrzedne (x, y) w pikselach na pozycje
			startingPoint[1] = y_fl;	// Zmiennoprzecinkowa w oknie obserwatora
			startingPoint[2] = viewArea;

			color[0] = 0.0;		// Wyznaczenie poczatku sledzonego promienia rys. piksela
			color[1] = 0.0;
			color[2] = 0.0;

			Trace(startingPoint, startingDir, 1);		// Wyznaczenie koloru piksela
			if (color[0] == 0.0) color[0] = backColor[0];
			if (color[1] == 0.0) color[1] = backColor[1];
			if (color[2] == 0.0) color[2] = backColor[2];

			// Wykonanie konwersji wartosci wyliczonego oswietlenia na liczby od 0 do 255
			color[0] > 1 ? pixel[0][0][0] = 255 : pixel[0][0][0] = color[0] * 255;
			color[1] > 1 ? pixel[0][0][1] = 255 : pixel[0][0][1] = color[1] * 255;
			color[2] > 1 ? pixel[0][0][2] = 255 : pixel[0][0][2] = color[2] * 255;

			glRasterPos3f(x_fl, y_fl, 0);	// INC pozycji rastrowej dla rys. piksela
			glDrawPixels(1, 1, GL_RGB, GL_UNSIGNED_BYTE, pixel);  //Rys next pix na ekranie
		}
	}
	glFlush();
}
```



Rysunek 1. Efekt końcowy programu wykorzystującego algorytm Recursive Ray Tracing.

4. Wnioski

Podczas niniejszego ćwiczenia zapoznano się z zaawansowanymi możliwościami oświetlania obiektów, oferowanymi przez bibliotekę OpenGL. Dzięki zastosowaniu algorytmu Rekursywnego Śledzenia Promieni oraz prostych brył 3D, udało się sprawić aby na sferach pojawiły się odbite źródła światła. W tym celu pomocne okazały się materiały zawarte w instrukcji laboratoryjnej, zawierające niezbędne metody oraz wstęp w postaci prostszego algorytmu. Co istotne, ostateczny program realizował wyznaczone cele oraz z sukcesem zaimplementowano algorytm Recursive Ray Tracing.

Podczas implementacji zauważono jak istotną rolę odgrywa ograniczenie wspomnianego mechanizmu. W przypadku zbyt dużej ilości rekurencyjnych wywołań funkcji, czas renderowania sceny znacząco się wydłuża, co zmusza do rozsądnego dobrania wspomnianego ograniczenia.

Dodatkową zaletą laboratorium była możliwość pracy z algorytmami, które znajdują zastosowanie w realiach, bowiem aplikacje już z przed kilkunastu lat z powodzeniem wykorzystywały swoje implementacje zrealizowanych tu rozwiązań. Zastosowanie tego typu algorytmów w sposób znaczący wpływa na atrakcyjność samej aplikacji i zbliża efekt końcowy do pierwowzorów inspirowanych obserwacją zjawisk w naszym otoczeniu

Zrealizowana metoda śledzenia promieni ma swoje silne matematyczne podparcie i pozwala na stworzenie realistycznego oświetlenia. Z pewnością jedną z wad algorytmu Ray Tracing może być jego złożoność obliczeniowa. Wygenerowanie stosunkowo prostej sceny zajmuje programowi kilka sekund. Ostatecznie bez większych problemów wykonano zadanie projektowe a efekty przedstawia rysunek 1.

5. Materiały pomocnicze

1. Instrukcja laboratoryjna http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_7_dz/