Politechnika Wrocławska
Wydział Elektroniki
Grafika Komputerowa i Komunikacja Człowiek-Komputer

Sprawozdanie z projektu (Ray Tracing)

Autor:
Dariusz Tomaszewski, 235565

Poniedziałek, 17^{00}-20^{00} TN
dr inż. Marek Woda

28 stycznia 2019

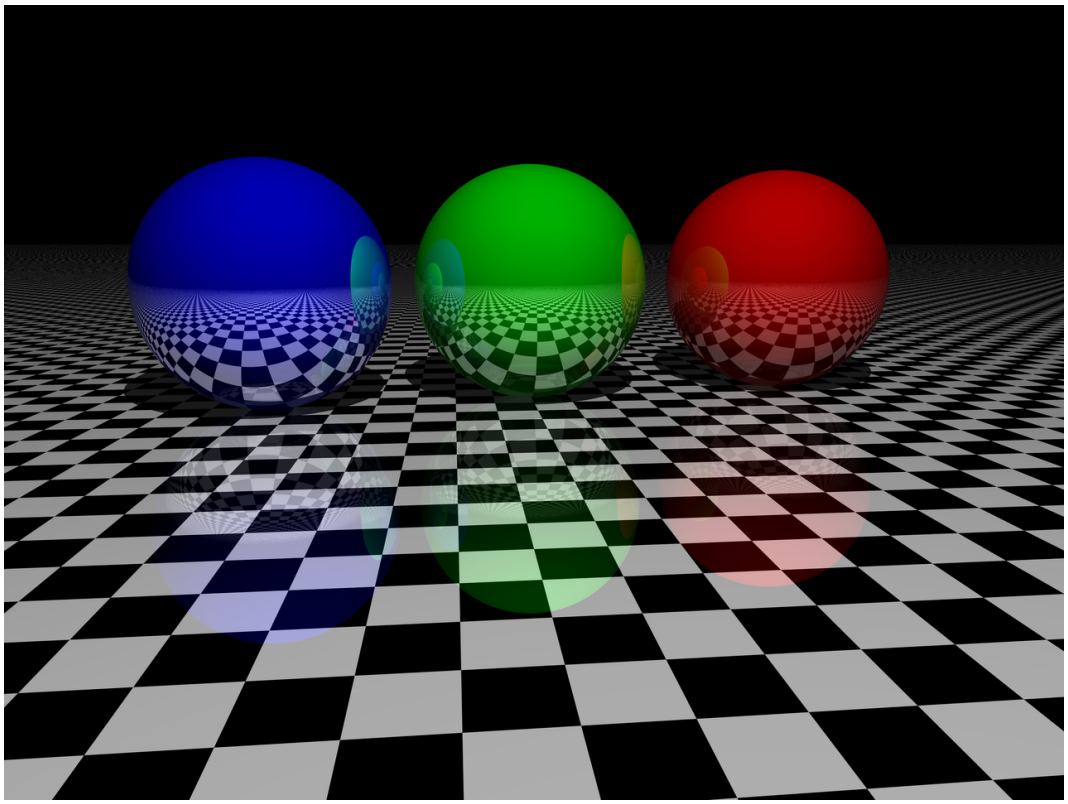
Spis treści

1 Wstęp

Celem projektu było napisanie aplikacji realizującej generowanie sceny za pomocą techniki śledzenia promieni (ang. Ray Tracing). Ray Tracing to technika pozwalająca na generowanie fotorealistycznych obrazów i scen 3D przy wykorzystaniu analizy promieni światła padających bezpośrednio na obserwatora. Jej zadaniem jest odtworzenie w uproszczeniu sposobu, w jaki ludzkie oko postrzega światło odbite od obiektów fizycznych. Zasada działania algorytmu w podstawowej wersji wygląda następująco:

1. Z punktu w którym znajduje się obserwator wyprowadzany jest promień pierwotny, który przecina rzutnię
2. Wyszukiwany jest najbliższy punkt przecięcia z obiektami znajdującymi się na scenie
3. Następnie dla każdego źródła światła zdefiniowanego na scenie wyznaczana jest jasność w tym punkcie, zgodnie z określonym modelem oświetlenia (np. Lamberta czy Phonga)

Śledzenie promieni nie jest jednak metodą idealną. Duża ilość obliczeń wymaga znacznych zasobów komputera, przez co jeszcze niedawno rzeczywiste wykorzystanie tego algorytmu nie było możliwe, poza sprzętem specjalistycznym. Generowanie pełnego obrazu wymaga ustalenia koloru osobno dla każdego z widocznych na ekranie pikseli. Dodatkowo, ze względu na zasadę działania, algorytm ten nie jest w stanie obsługiwać światła rozproszonego, symulować dyfrakcji lub rozszczepienia światła.
Wydane ostatnio karty graficzne Nvidia RTX zawierają specjalny układ, którego jedynym zadaniem jest obsługa technologii Ray Tracing w czasie rzeczywistym.



Rysunek 1: Przykład działania technologii Ray Tracing

2 Omówienie kodu

Na początku zacząłem od stworzenia klas reprezentujących obiekty na scenie oraz takich, które ułatwią późniejszą manipulację danymi. Następnie można było tworzyć funkcje oraz modyfikować istniejące celem osiągnięcia zamierzonego rezultatu. Poniżej pokazany jest kod źródłowy każdej klasy, zmiennych oraz funkcji.

2.1 Klasa Point3

Klasa *Point3* opisuje punkt w trójwymiarowej przestrzeni. Posiada ona trzy pola typu zmiennoprzecinkowego (float). Za pomocą tej klasy można definiować nie tylko punkt w przestrzeni, ale również wektor oraz kolory.

```
class Point3 {
public:
        float x, y, z;
        Point3() : x(0), y(0), z(0) {}
        Point3(float x, float y, float z) : x(x), y(y), z(z) {}
};
```

Code 1: Klasa opisująca punkt

2.2 Klasa Light

Klasa *Light* określa pojedyńcze źródło światła na scenie. Zawiera ona takie informacje jak pozycję obiektu w przestrzeni trójwymiarowej, intensywność świecenia źródła światła powodującego odbicie kierunkowe, intensywność świecenia źródła światła powodującego odbicie dyfuzyjne oraz intensywność świecenia źródła światła otoczenia.

```
class Light {
public:
        Point3 position, specular, diffuse, ambient;
        Light(Point3 position, Point3 specular, Point3 diffuse, Point3 ambient) {
                this->position = position;
                this->specular = specular;
                this->diffuse = diffuse;
                this->ambient = ambient;
        }
        Light() {
                this->position = Point3(0, 0, 0);
                this->specular = Point3(0, 0, 0);
                this->diffuse = Point3(0, 0, 0);
                this->ambient = Point3(0, 0, 0);
        }
};
```

Code 2: Klasa opisująca światło

2.3 Klasa Sphere

Klasa *Sphere* określa sferę położoną na scenie. Zawiera ona informacje o pozycji sfery na scenie, jej wielkości (promieniu), stopniu działania na nią światła odbitego, światła rozproszonego, światła otoczenia oraz współczynnik opisujący połysk powierzchni.

```
class Sphere {
public:
        Point3 position, specular, diffuse, ambient;
        float radius, specularhininess;
        Sphere(Point3 position, float radius, Point3 specular, Point3 diffuse,
        ↪  Point3 ambient, float specularhininess) {
                this->position = position;
                this->radius = radius;
                this->specular = specular;
                this->diffuse = diffuse;
                this->ambient = ambient;
                this->specularhininess = specularhininess;
        }
        Sphere() {
                this->position = Point3(0, 0, 0);
                this->radius = 0.0;
                this->specular = Point3(0, 0, 0);
                this->diffuse = Point3(0, 0, 0);
                this->ambient = Point3(0, 0, 0);
                this->specularhininess = 0.0;
        }
};
```

Code 3: Klasa opisująca sferę

2.4 Klasa SpherePoint

Klasa *SpherePoint* opisuje punkt na sferze. Instancja tej klasy tworzona jest podczas obliczania punktu przecięcia promienia. Informacje w niej zawarte są wykorzystywane podczas obliczania koloru oraz oświetlenia dla danego punktu. Klasa ta zawiera informacje o punkcie, indeks odpowiadającej sfery, indeks odpowiadającego światła oraz status czy promień ten przecina się ze sferą czy wskazuje on na źródło światła.

```
class SpherePoint {
public:
        Point3 point;
        int sphere_index;
        int light_index;
        Status status;
        SpherePoint(Point3 point, int sphere_index, int light_index, Status
        ↪  status) {
                this->point = point;
                this->sphere_index = sphere_index;
                this->light_index = light_index;
                this->status = status;
        }
        SpherePoint() {
                this->point = Point3(0.0, 0.0, 0.0);
                this->sphere_index = 0;
                this->light_index = 0;
                this->status = NO_INTERSECTION;
        }
};
```

Code 4: Klasa opisująca punkt na sferze

2.5 Zmienne globalne

Zmienna *viewportsize* określa odległość obserwatora, $viewer_vec$ określa wektor obserwacji, *maxdepth* określa maksymalną ilość odbić promienia, *light* jest tablicą zawierającą obiekty typu Light, *sphere* zawiera obiekty typu Sphere, zmienna *background* określa kolor tła, zmienna *global* określa oświetlenie globalne, a zmienne *windowsize* określają rozmiar okna.

```
std::vector<Light> light;
std::vector<Sphere> sphere;
Point3 background(0.0, 0.0, 0.0);
Point3 global(0.0, 0.0, 0.0);
Point3 viewer_vec(0.0, 0.0, 1.0);
float viewport_size = 15.0;
int window_size_x, window_size_y;
int max_depth = 2;
```

Code 5: Zmienne globalne, dane wczytywane z pliku

2.6 Funkcja dotProduct

Funkcja *dotProduct* oblicza iloczyn skalarny dwóch wektorów. Jako parametry przyjmuje dwa obiekty typu $Point3(Wektor)$ i zwraca obiekt typu $Point3(Wektor)$.

```
float dotProduct(Point3 &p1, Point3 &p2) {
        return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}
```

Code 6: Funkcja obliczająca iloczyn skalarny wektorów

2.7 Funkcja normalization

Funkcja *normalization* jako argument przyjmuje obiekt klasy $Point3(Wektor)$ i zwraca ona znormalizowany wektor jako obiekt klasy $Point3(Wektor)$.

```
Point3 normalization(Point3 &p) {
        Point3 result = p;
        float d = 0.0;
        d += result.x * result.x;
        d += result.y * result.y;
        d += result.z * result.z;
        d = sqrt(d);
        if (d > 0.0) {
                result.x /= d;
                result.y /= d;
                result.z /= d;
        }
        return result;
}
```

Code 7: Funkcja normalizująca wektor

2.8 Funkcja normal

Funkcja *normal* tworzy wektor normalny do powierzchni danej sfery w konkretnym punkcie. Jako argumenty przyjmuje punkt dla którego ma ona wyliczyć wektor normalny oraz indeks sfery. Zwraca ona obiekt klasy $Point3(Wektor)$.

```
Point3 normal(Point3 &q, int sphere_index) {
        Point3 result(q.x - sphere[sphere_index].position.x, q.y -
        ↪  sphere[sphere_index].position.y, q.z -
        ↪  sphere[sphere_index].position.z);
        return normalization(result);
}
```

Code 8: Funkcja tworząca wektor normalny do powierzchni danej sfery w punkcie q

2.9 Funkcja reflect

Funkcja $reflect$ wyznacza wektor jednostkowy opisujący kierunek kolejnego śledzonego promienia. Promień ten powstaje w wyniku odbicia promienia wychodzącego z punktu p

i biegnącego do punktu q na powierzchni obiektu. Dla wyznaczenia kierunku odbicia należy jeszcze znać wektor normalny do powierzchni w punkcie q. Wektor odbicia wyliczany jest na podstawie zasady takiej, że kąt odbicia promienia jest równy kątowi pod jakim promień pada na analizowany punkt.

```
Point3 reflect(Point3 &p, Point3 &q, Point3 &n) {
        Point3 direct_vec(p.x - q.x, p.y - q.y, p.z - q.z);
        direct_vec = normalization(direct_vec);
        float n_dot_l = dotProduct(direct_vec, n);
        Point3 result(2 * (n_dot_l)* n.x - direct_vec.x, 2 * (n_dot_l)* n.y -
        ↪  direct_vec.y, 2 * (n_dot_l)* n.z - direct_vec.z);

        if (result.x * result.x + result.y * result.y + result.z * result.z >
        ↪  1.0)
                return normalization(result);
        else return result;
}
```

Code 9: Funkcja obliczająca kierunek odbicia promienia w punkcie q

2.10 Funkcja intercest

Funkcja *intercest* wyznacza współrzędne punktu przecięcia z najbliższym obiektem sceny, znajdującym się na drodze śledzonego promienia. Argumentami funkcji jest punkt startowy będący początkiem promienia oraz kierunek promienia vec. Funkcja zwraca obiekt klasy SpherePoint która została omówiona wyżej. Funkcja ustala w jaki obiekt trafia dany promień i ustawia zmienną *status* na jedną z trzech możliwych wartości:

- *lightsource* - gdy śledzony promień trafi w źródło swiatła
- *intersection* - w przypadku przecięcia z obiektem
- *nointesection* - gdy promień nie trafia w żaden obiekt sceny.

```
SpherePoint intercest(Point3 &start, Point3 &vec) {
        SpherePoint result;
        for (int i~= 0; i~< light.size(); ++i) {
                float x = light[i].position.x - start.x;
                float y = light[i].position.y - start.y;
                float z = light[i].position.z - start.z;
                if ((x / vec.x) == (y / vec.y) && (y / vec.y) == (z / vec.z)) {
                        result.point = light[i].position;
                        result.status = LIGHT_SOURCE;
                        return result;
                }
        }
        for (int i~= 0; i~< sphere.size(); ++i) {
                float a~= vec.x * vec.x + vec.y * vec.y + vec.z * vec.z;
                float b = 2 * ((start.x - sphere[i].position.x) * vec.x +
                ↪  (start.y - sphere[i].position.y) * vec.y + (start.z -
                ↪  sphere[i].position.z) * vec.z);
```

```
16                  float c = (start.x * start.x + start.y * start.y + start.z *
                     ↪  start.z) + (sphere[i].position.x * sphere[i].position.x +
                     ↪  sphere[i].position.y * sphere[i].position.y +
                     ↪  sphere[i].position.z * sphere[i].position.z) - 2 * (start.x
                     ↪  * sphere[i].position.x + start.y * sphere[i].position.y +
                     ↪  start.z * sphere[i].position.z) - sphere[i].radius *
                     ↪  sphere[i].radius;
17                  float delta = b * b - 4 * a~* c;
18                  if (delta >= 0) {
19                          float r = (-b - sqrt(delta)) / (2 * a);
20                          if (r > 0) {
21                                  result.point.x = start.x + r * vec.x;
22                                  result.point.y = start.y + r * vec.y;
23                                  result.point.z = start.z + r * vec.z;
24                                  result.sphere_index = i;
25                                  result.status = INTERSECTION;
26                                  break;
27                          }
28                  }
29          }
30          return result;
31  }
```

Code 10: Funkcja wyznaczająca współrzędne punktu przecięcia z najbliższym obiektem sceny

2.11 Funkcja phong

Funkcja *phong* wyznacza oświetlenie lokalne dla punktu q. Oświetlenie to jest sumą oświetleń pochodzących od wszystkich źródeł, które są bezpośrednio widoczne z analizowanego punktu. Obliczenie oświetlenia polega na użyciu modelu Phonga. Argumenty jakie przyjmuje funkcja to punkt dla którego wyliczane jest oświetlenie, wektor znormalizowany do punktu q oraz indeks sfery dla której to oświetlenie jest obliczane.

```
1   Point3 phong(Point3 &q, Point3 &n, int sphere_index) {
2           Point3 color(0.0, 0.0, 0.0);
3           Point3 viewer_v(0.0, 0.0, 1.0);
4           for (int i~= 0; i~< light.size(); ++i) {
5                   Point3 light_vec(light[i].position.x - q.x, light[i].position.y -
                     ↪  q.y, light[i].position.z - q.z);
6                   light_vec = normalization(light_vec);
7                   float n_dot_l = dotProduct(light_vec, n);
8                   Point3 reflection_vec(2 * (n_dot_l)* n.x - light_vec.x, 2 *
                     ↪  (n_dot_l)* n.y - light_vec.y, 2 * (n_dot_l)* n.z -
                     ↪  light_vec.z);
9                   reflection_vec = normalization(reflection_vec);
10                  float v_dot_r = dotProduct(viewer_v, reflection_vec);
11                  if (v_dot_r < 0) v_dot_r = 0;
12                  if (n_dot_l > 0) {
```

```
13                      color.x += (sphere[sphere_index].diffuse.x *
                        ↪  light[i].diffuse.x * n_dot_l +
                        ↪  sphere[sphere_index].specular.x *
                        ↪  light[i].specular.x * pow(v_dot_r,
                        ↪  sphere[sphere_index].specularhininess) +
                        ↪  sphere[sphere_index].ambient.x * light[i].ambient.x
                        ↪  + sphere[sphere_index].ambient.x * global.x);
14                      color.y += (sphere[sphere_index].diffuse.y *
                        ↪  light[i].diffuse.y * n_dot_l +
                        ↪  sphere[sphere_index].specular.y *
                        ↪  light[i].specular.y * pow(v_dot_r,
                        ↪  sphere[sphere_index].specularhininess) +
                        ↪  sphere[sphere_index].ambient.y * light[i].ambient.y
                        ↪  + sphere[sphere_index].ambient.y * global.y);
15                      color.z += (sphere[sphere_index].diffuse.z *
                        ↪  light[i].diffuse.z * n_dot_l +
                        ↪  sphere[sphere_index].specular.z *
                        ↪  light[i].specular.z * pow(v_dot_r,
                        ↪  sphere[sphere_index].specularhininess) +
                        ↪  sphere[sphere_index].ambient.z * light[i].ambient.z
                        ↪  + sphere[sphere_index].ambient.z * global.z);
16                  }
17                  else {
18                          color.x += (sphere[sphere_index].ambient.x) * global.x;
19                          color.y += (sphere[sphere_index].ambient.y) * global.y;
20                          color.z += (sphere[sphere_index].ambient.z) * global.z;
21                  }
22          }
23          return color;
24  }
```

Code 11: Funkcja obliczająca oświetlenie punktu na powierzchni sfery używając modelu Phonga

2.12 Funkcja trace

Funkcja *trace* oblicza kolor piksela (punktu) dla promienia zaczynającego się w punkcie *start* i biegnącego w kierunku wskazywanym przez wektor *direction*. Jest to funkcja rekurencyjna. Argument *start* jest punktem początkowym promienia, *direction* wektorem opisującym kierunek biegu promienia, a *step* jest licznikiem odbić promienia. Funkcja zwraca kolor jaki obliczyła.

```
1   Point3 trace(Point3 &start, Point3 &direction, int step) {
2           Point3 normal_vec;
3           Point3 reflect_vec;
4           Point3 local(0.0, 0.0, 0.0);
5           Point3 reflected(0.0, 0.0, 0.0);
6           if (step > max_reflections) return background;
7
8           SpherePoint q = intercest(start, direction);
9           Point3 q_vec(q.point.x, q.point.y, q.point.z);
10
11          if (q.status == LIGHT_SOURCE) return light[q.light_index].specular;
12          if (q.status == NO_INTERSECTION) return background;
13
```

```
14          normal_vec = normal(q_vec, q.sphere_index);
15          reflect_vec = reflect(start, q_vec, normal_vec);
16          local = phong(q_vec, normal_vec, q.sphere_index);
17          reflected = trace(q_vec, reflect_vec, step + 1);
18          local.x += reflected.x;
19          local.y += reflected.y;
20          local.z += reflected.z;
21          return local;
22  }
```

Code 12: Funkcja obliczająca kolor piksela (punktu) na podstawie promienia opisanego w argumentach

2.13 Funkcja display

Funkcja *display* rysuje obraz oświetlonej sceny. Obraz rysowany jest piksel po pikselu na podstawie informacji zwracanych z funkcji trace.

```
1   void display() {
2           glClear(GL_COLOR_BUFFER_BIT);
3           glFlush();
4           int window_size_half_x = window_size_x / 2;
5           int window_size_half_y = window_size_y / 2;
6           for (int y = window_size_half_y; y > -window_size_half_y; --y) {
7               for (int x = -window_size_half_x; x < window_size_half_x; ++x) {
8                           float x_fl = (float)x / (window_size_x / viewport_size);
9                           float y_fl = (float)y / (window_size_y / viewport_size);
10                          Point3 starting_point(x_fl, y_fl, viewport_size);
11                          Point3 starting_directions(0.0, 0.0, -1.0);
12                          Point3 color = trace(starting_point, starting_directions,
                             ↪  0);
13                          GLubyte pixel[1][1][3];
14                          if (color.x > 1) pixel[0][0][0] = 255;
15                          else pixel[0][0][0] = color.x * 255;
16                          if (color.y > 1) pixel[0][0][1] = 255;
17                          else pixel[0][0][1] = color.y * 255;
18                          if (color.z > 1) pixel[0][0][2] = 255;
19                          else pixel[0][0][2] = color.z * 255;
20                          glRasterPos3f(x_fl, y_fl, 0);
21                          glDrawPixels(1, 1, GL_RGB, GL_UNSIGNED_BYTE, pixel);
22                  }
23                  if (!(y % 10)) glFlush();
24          }
25          glFlush();
26  }
```

Code 13: Funkcja rysująca obraz oświetlonej sceny

2.14 Funkcja readFile

Funkcja *readFile* odczytuje plik o nazwie podanej w argumencie. Najpierw sprawdza czy plik udało się otworzyć, a następnie wczytuje kolejne linie pliku dopóki nie nastąpi koniec pliku. Jeśli w konkretnej linii zostanie wykryte słowo kluczowe to następuje odczyt parametrów danego obiektu i zapisanie tego obiektu lub parametrów do zmiennych globalnych.

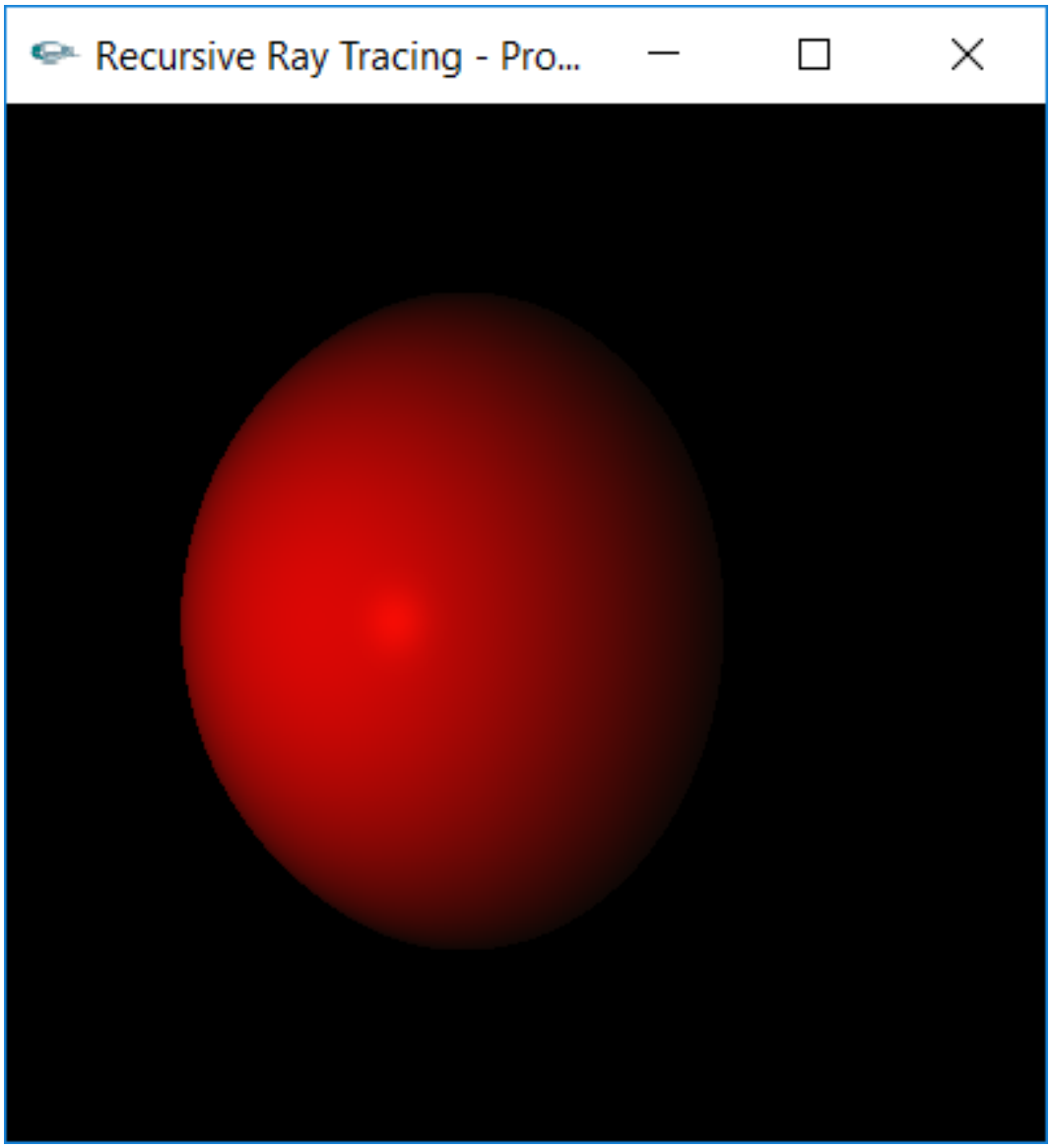
```
void readFile(std::string path) {
        std::fstream file(path, std::ios::in);
        std::string line, type;
        if (!file.is_open()) {
                std::cout << "Wczytywanie pliku " << path << " nieudane!";
        }
        while (getline(file, line)) {
                std::istringstream iss(line);
                iss >> type;
                if (!type.compare("dimensions")) {
                        iss >> window_size_x >> window_size_y;
                }
                else if (!type.compare("background"))
                        iss >> background.x >> background.y >> background.z;
                else if (!type.compare("global"))
                        iss >> global.x >> global.y >> global.z;
                else if (!type.compare("viewer"))
                        iss >> viewer_vec.x >> viewer_vec.y >> viewer_vec.z;
                else if (!type.compare("viewport_size"))
                        iss >> viewport_size;
                else if (!type.compare("max_depth"))
                        iss >> max_depth;
                else if (!type.compare("sphere")) {
                        Sphere loaded_sphere;
                        iss >> loaded_sphere.radius >> loaded_sphere.position.x
                        ↪  >> loaded_sphere.position.y >>
                        ↪  loaded_sphere.position.z;
                        iss >> loaded_sphere.specular.x >>
                        ↪  loaded_sphere.specular.y >>
                        ↪  loaded_sphere.specular.z;
                        iss >> loaded_sphere.diffuse.x >> loaded_sphere.diffuse.y
                        ↪  >> loaded_sphere.diffuse.z;
                        iss >> loaded_sphere.ambient.x >> loaded_sphere.ambient.y
                        ↪  >> loaded_sphere.ambient.z;
                        iss >> loaded_sphere.specularhininess;
                        sphere.push_back(loaded_sphere);
                }
                else if (!type.compare("source")) {
                        Light loaded_light;
                        iss >> loaded_light.position.x >> loaded_light.position.y
                        ↪  >> loaded_light.position.z;
                        iss >> loaded_light.specular.x >> loaded_light.specular.y
                        ↪  >> loaded_light.specular.z;
                        iss >> loaded_light.diffuse.x >> loaded_light.diffuse.y
                        ↪  >> loaded_light.diffuse.z;
                        iss >> loaded_light.ambient.x >> loaded_light.ambient.y
                        ↪  >> loaded_light.ambient.z;
                        light.push_back(loaded_light);
                }
        }
        file.close();
}
```

Code 14: Funkcja odczytująca plik z parametrami sceny

3 Rezultat prac

3.1 Sfera czerwona z jednym źródłem światła

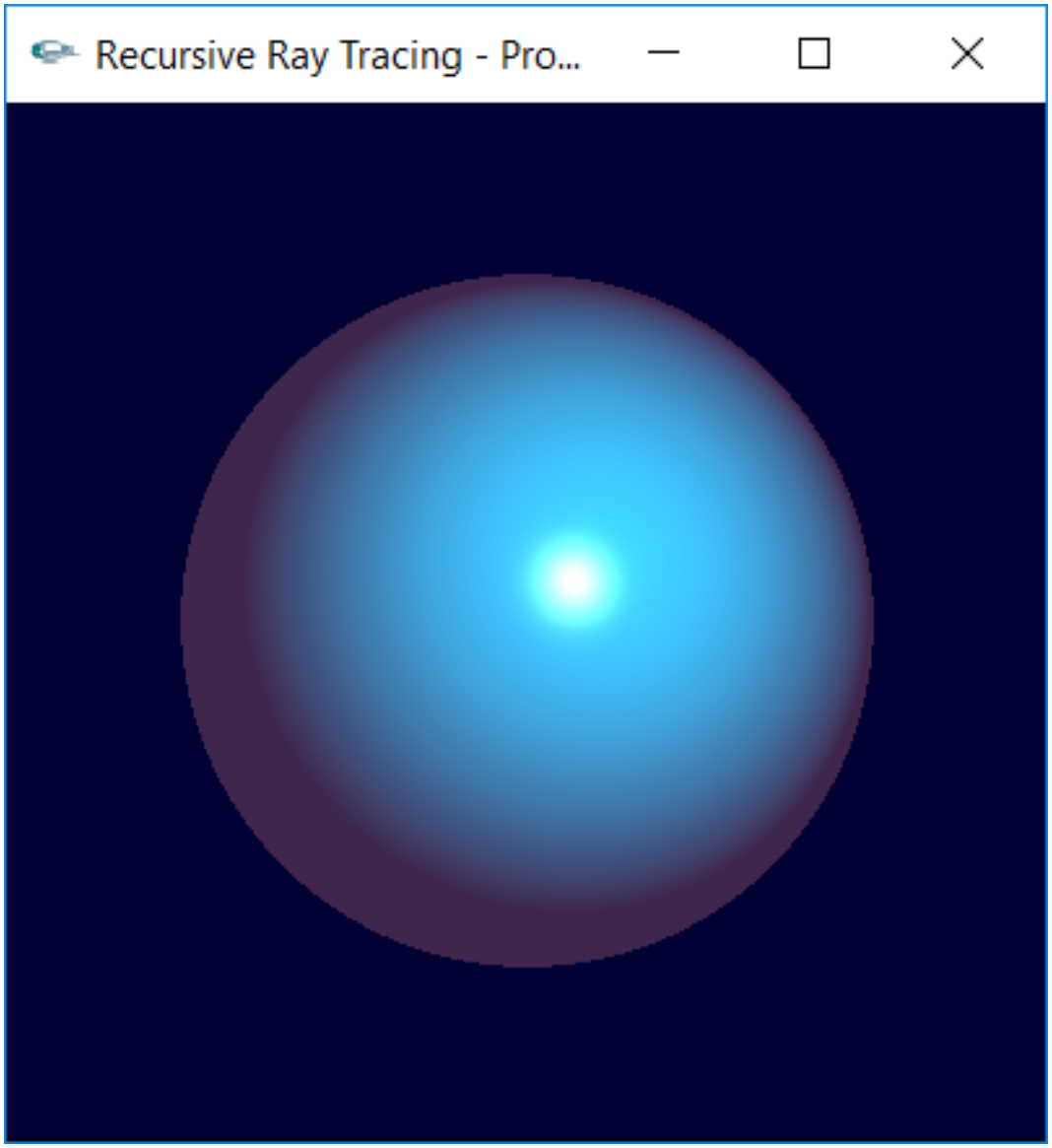
dimensions 400 400
background 0.0 0.0 0.0
global 0.0 0.0 0.0
sphere 5.0 0.0 0.0 -3.0 0.8 0.1 0.0 0.8 0.1 0.0 0.2 0.1 0.2 40
source -10.0 0.0 10.0 0.2 0.2 0.2 1.0 0.0 1.0 0.3 0.3 0.1



Rysunek 2: Sfera czerwona z jednym źródłem światła

3.2 Sfera niebieska z jednym źródłem światła

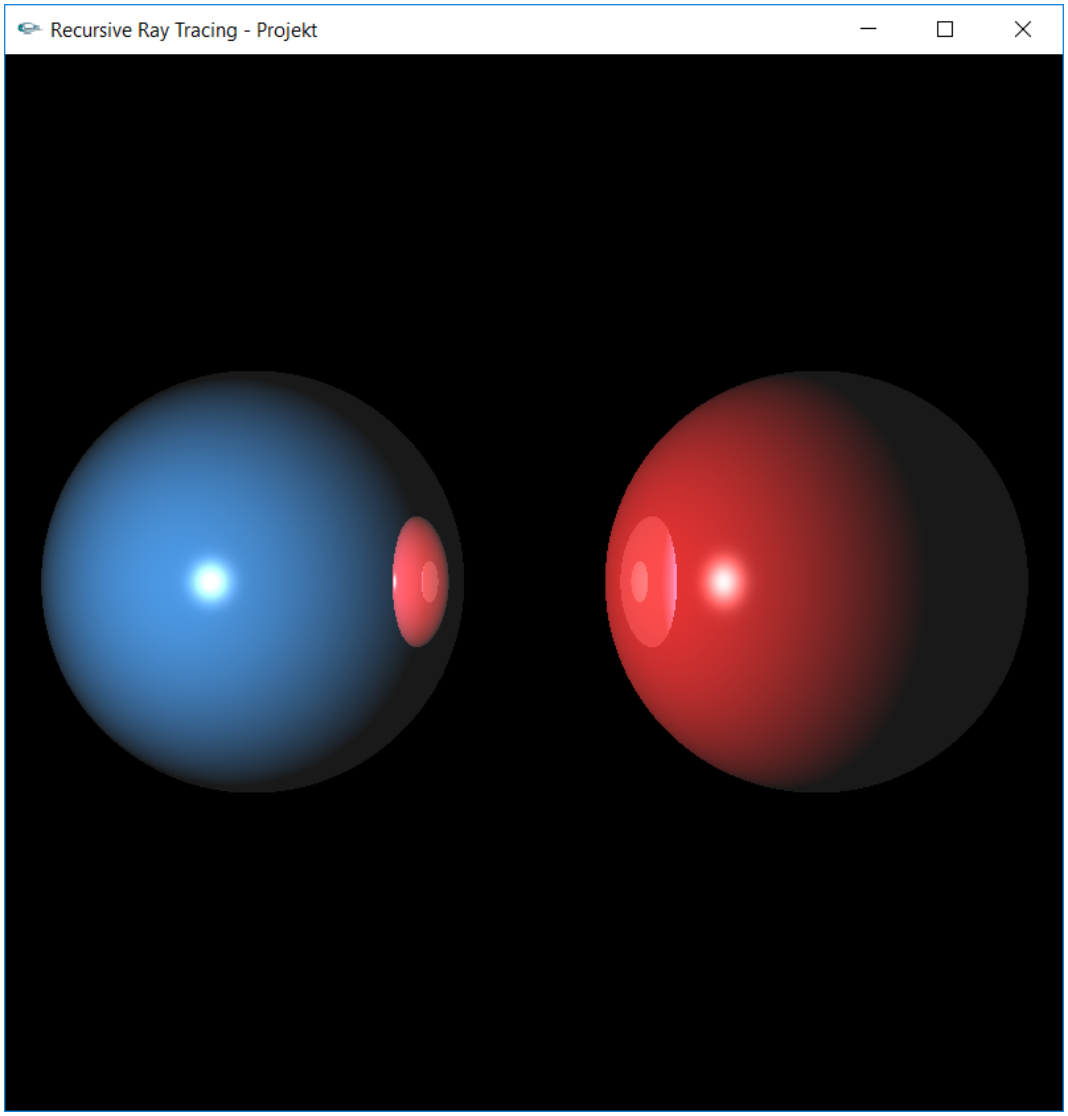
dimensions 400 400
background 0.0 0.0 0.1992
global 0.25 0.15 0.1
sphere 5.0 0.0 0.0 -8.0 0.8 0.8 0.8 0.6 0.7 0.8 1.0 1.0 1.0 30.0
source 3.0 2.5 5.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0



Rysunek 3: Sfera niebieska z jednym źródłem światła

3.3 Dwie sfery obok siebie z jednym źródłem światła

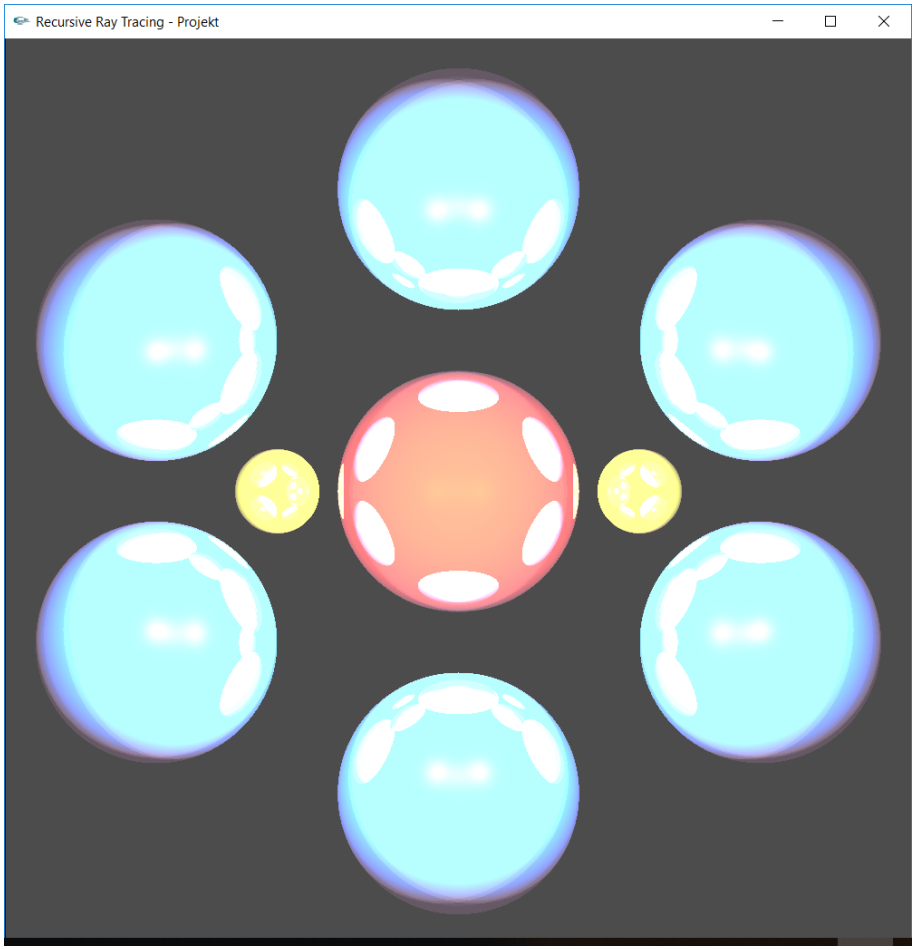
dimensions 800 800
background 0.0 0.0 0.0
global 0.1 0.1 0.1
sphere 3.0 4.0 0.0 -3.0 0.8 0.8 0.8 0.8 0.1 0.1 1.0 1.0 1.0 50
sphere 3.0 -4.0 0.0 -3.0 0.8 0.8 0.8 0.2 0.5 0.8 1.0 1.0 1.0 50
source -8.0 0.0 8.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0



Rysunek 4: Dwie sfery obok siebie z jednym źródłem światła

3.4 Scena złożona z 9 sfer i 5 źródeł światła

Przykładowa scena ze strony `http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_7_dz/`.



Rysunek 5: Obraz sceny złożonej z 9 sfer i 5 źródeł światła

4 Wnioski

Podczas wykonywania tego zadania udało mi się zapoznać z zasadą działania techniki Ray Tracing. Zauważyłem dość dużą złożoność obliczeniową dla obliczenia koloru każdego piksela przez co technika ta nie nadaje się do symulowania płynnych animacji w czasie rzeczywistym przynajmniej na poziomie programowym, ponieważ obecnie na rynku istnieją karty graficzne ze sprzętową obsługą techniki Ray Tracing w czasie rzeczywistym co pozwoli na płynne generowanie animacji z wykorzystaniem tej technologii i użycie jej np. w grach komputerowych.
Kod został od początku napisany w myśl dużej optymalizacji oraz uproszczenia niektórych operacji, czyli zostało ograniczone do absolutnego minimum zbędne kopiowanie parametrów i zmiennych, operacje potęgowania do 2 zostały zastąpione operacjami mnożenia oraz zapewniono szybszą i prostszą inicjalizacja zmiennych za pomocą konstruktorów. Generowanie sceny 400x400 jest na tyle szybkie, że zdecydowałem się generować niektóre sceny w wyższej rozdzielczości. Również nie było sensu ustawiać maksymalnej głębii na wyższą niż 2, ponieważ zwiększyłoby to ilość odbić promieni a tym samym złożoność obliczeniową, a efekt tego nie byłby już na tyle widoczny.

Literatura

[1] Metoda śledzenia promieni (Ray Tracing) - Zespół Systemów Komputerowych i Dyskretnych :
`http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_7_dz/`

[2] Śledzenie promieni - Wikipedia :
`https://pl.wikipedia.org/wiki/%C5%9Aledzenie_promieni`

[3] Jak to działa? Raytracing i techniki oświetlenia globalnego - PCLab.pl :
`https://pclab.pl/art77482.html`

[4] Metoda śledzenia promieni - Radosław Mantiuk :
`http://rmantiuk.zut.edu.pl/data/wyklad_ray_tracing.pdf`

Spis rysunków

List of Listings