Politechnika Łódzka

Instytut Informatyki

Instrukcja do laboratorium

Fotorealistyczna Grafika Komputerowa

Laboratorium II

Kamera, próbkowanie, obraz, natężenie

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

dr inż. Piotr Napieralski
mgr inż. Krzysztof Guzek

Łódź, 28.02.2012

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: napieral@ics.p.lodz.pl

1. Wstęp

Wykonanie ćwiczenia z poprzednich zajęć pozwoliło na zdefiniowanie podstawowych i bazowych elementów systemu renderującego. Dzisiejsze zajęcia pozwolą na otrzymanie pierwszego obrazu. By móc wygenerować obraz należy zdefiniować odpowiednią strukturę definiującą sam obraz jak i kamerę. Dodatkowo definiując kamerę możemy określić sposoby próbkowania i antyaliasingu.

2. Kolor i obraz

Kiedy policzymy przecięcie promienia z najbliższym obiektem, powinniśmy znaleźć kolor piksela przez który promień przeszedł i zapisać znalezioną wartość w tablicy pikseli (obrazie). Dlatego też dwa kolejne elementy składające się na nasz system to klasa odpowiedzialna za kolor i obraz.

2.1 Kolor a natężenie światła

Będzie nam potrzebne natężenie światła otoczenia (ang. ambient), które jest stałe w całym środowisku, nie posiada jednoznacznie określonej pozycji i jest równe I_a. Światłość (natężenie światła) jest to wielkość fizyczna mierzona stosunkiem strumienia świetlnego emitowanego przez źródło światła. Jak na razie nie będziemy brali pod uwagę odbić. Zajmiemy się sytuacją, gdy nie mamy żadnych konkretnych źródeł światła i zakładamy ze we wszystkich kierunkach rozchodzi się tak samo (ambient). Dzięki natężeniu światła będziemy mogli obliczyć kolor obiektów naszego wirtualnego świata. Natężenie składa się z trzech składowych (R (czerwony) G (zielony) B (niebieski)) powinno być dodatnie lub 0. W przypadku wartości ujemnej, zamieniamy jego wartość na 0. Składowe przyjmować będą wartości od 0 do 1 i określać będą kolejne składowe koloru.

Prócz samej definicji potrzebne będą metody pozwalające na dokonywanie pewnych operacji między kolejnymi natężeniami jak i natężeniem i skalarem. Podstawowe metody mające znaleźć się w naszej klasie to dodawanie natężeń oraz dzielenie przez skalar.

Poniżej znajduje się przykładowa definicja natężenia w C++

```
class LightIntensity
{
      protected:
      double r, g, b;

      public:
     LightIntensity(double R, double G, double B){ r=R; g=G; b=B;}
     LightIntensity(double R, double G){ r=R; g=G; b=0.0;}
     LightIntensity(double R){ r=R; g=b=0.0;}
```

```
        LightIntensity(){r=g=b=0.0;}

        double gRed()   {return r;}
        double gGreen() {return g;}
        double gBlue()  {return b;}

        void operator()(float R, float G, float B) { r=R; g=G; b=B;}
        void operator()(float R, float G) { r=R; g=G; b=0.0;}
        void operator()(float R) { r=R; g=b=0.0;}

        void add(double R, double G, double B);

        LightIntensity operator+(LightIntensity &li);
         LightIntensity operator-(LightIntensity &li);
         LightIntensity operator/(float num);
         void operator+=(LightIntensity &li);
         LightIntensity operator-=(LightIntensity &li);
         LightIntensity operator*=(float num);
         LightIntensity operator/=(float num);
    friend LightIntensity operator*(float num, LightIntensity &li);
    friend LightIntensity operator*(LightIntensity &li, float num);
    friend std::ostream& operator<<(std::ostream& str,
LightIntensity &li);
};
```

2.2 Obraz

Obraz to nic innego jak tablica o rozmiarach [X*Y] w której umieścimy obliczony kolor. W przypadku obrazu możemy wykorzystać gotowe metody lub biblioteki. Dla osób piszących w javie proponuję wykorzystać możliwości java.awt.Color, dla C++ zamieszczam plik do pobrania: Bitmapa[1].rar Trzy składowe koloru ustawione na 0 to kolor czarny na 1 to kolor biały. Najczęściej stosowany jest 24-bitowy zapis kolorów, w którym każda z barw jest zapisana przy pomocy składowych, które przyjmują wartość z zakresu 0-255. W modelu RGB 0 oznacza kolor czarny, natomiast 255 kolor biały.

Po obliczeniu natężenia wynikowego powinna istnieć możliwość przekonwertowania wartości z zakresu $I \in [0;1)$ na $i \in \{0,1,...255\}$. Należy tu wykonać proste rzutowanie $i = \mathrm{int}(256 * I)$. Format pliku jak i sposób wyświetlenia obrazu końcowego jest dowolny, zarówno może być wyświetlony bezpośrednio na ekran jak i zapisany do pliku graficznego. Propozycje poniżej dotyczą zapisu do pliku formacie bitmapy.

Dla eksperymentatorów z C# proponuję wykorzystanie biblioteki System.Drawing:

[1] http://ics.p.lodz.pl/~napieral/fotorealistyczna/Bitmapa.rar

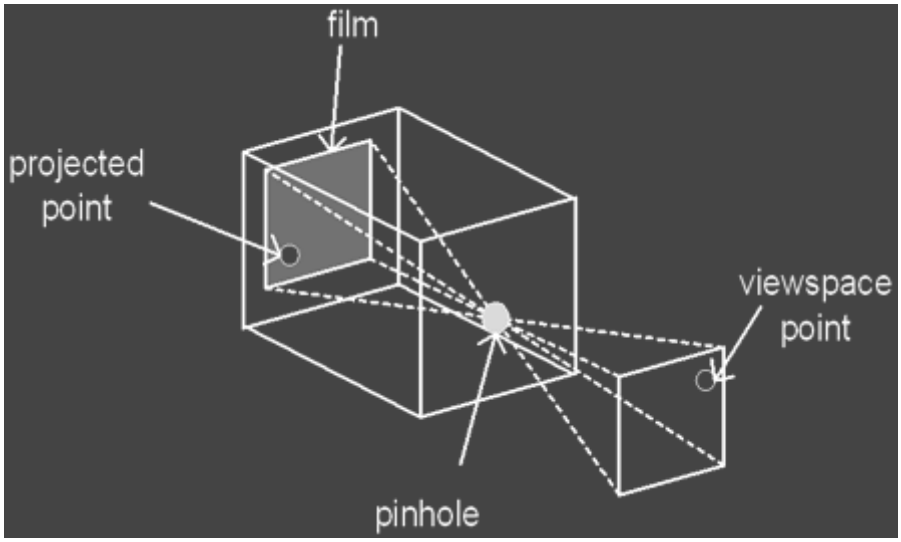
```
using System.Drawing;

//a tam mamy:
obraz = new Bitmap(xsize, ysize,
System.Drawing.Imaging.PixelFormat.Format32bppArgb);

//Należy teraz obliczyć kolor na podstawie natężenia:

public void setPixel(int x, int y, Intensity pixel)
{
int red, green, blue;
//konwersja na 0-255
red = (int)(pixel.getRed() * 255);
green = (int)(pixel.getGreen() * 255);
blue = (int)(pixel.getBlue() * 255);
obraz.SetPixel(x, y, Color.FromArgb(red, green, blue));
}
```

3. Kamera

„Promień świetlny poruszający się (w dowolnym ośrodku) od punktu A do punktu B przebywa zawsze lokalnie ekstremalną drogę optyczną, czyli taką, na której przebycie potrzeba czasu najkrótszego, bądź najdłuższego z możliwych.[2].”

Proste urządzenie rejestrujące obraz możemy stworzyć za pomocą zwykłego kartonowego pudełka i błony światłoczułej. Tak zbudowane urządzenie nosi nazwę camera obscura. Konstrukcja tego urządzenia polega na pomalowaniu wewnątrz pudełka czarną farbą, wycięciu niewielkiego otworu z jednej strony i umieszczeniu błony światłoczułej po przeciwnej stronie wyciętego otworu (Rys.1). Światło wpadające przez wycięty otwór w naświetli błonę filmową, tworząc obraz z odbitego światła od przedmiotów znajdujących się przed naszym prymitywnym aparatem fotograficznym. Pierwsze aparaty fotograficzne opierały się na budowie przedstawionego urządzenia. Prosta konstrukcja niesie ze sobą niedogodności związane z długim czasem naświetlania. W przypadku zwiększenia otworu czas naświetlania zostanie radykalnie skrócony, lecz powstały obraz będzie dość mocno rozmazany.

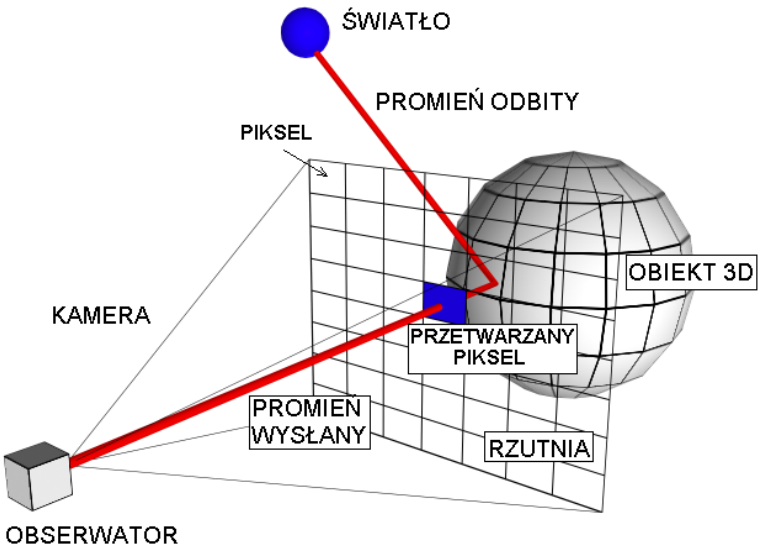
[2] Zasada Fermata



Rys.1 Zasada działania camera obscura

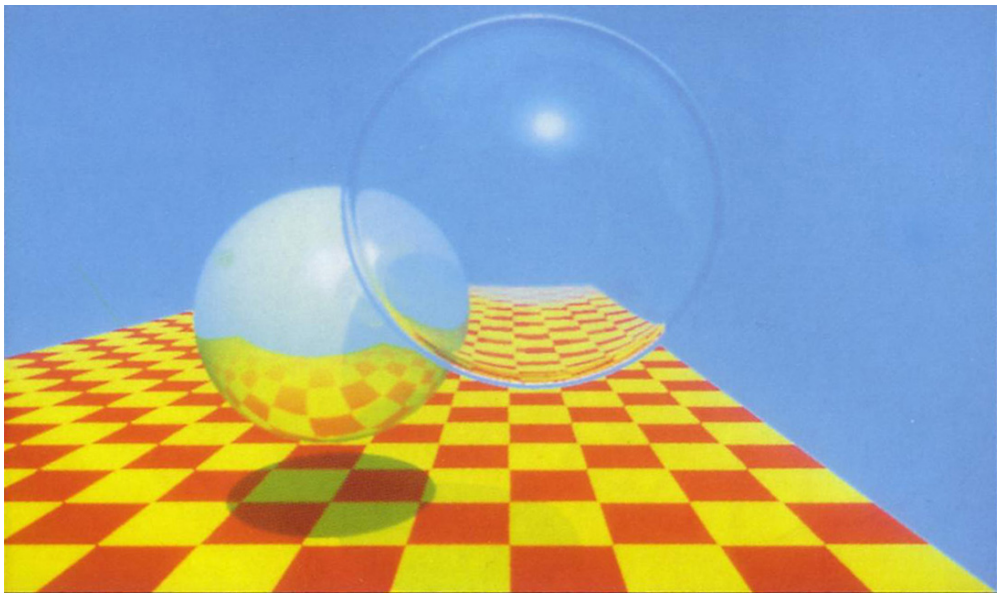
Współczesne aparaty fotograficzne wyposażone zostały w optykę oraz przesłonę umożliwiającą zmianę wielkości otworu przez którego wpadać będzie światło. Punkt naświetlony na błonie filmowej, powstaje na skutek rzutowania stożka widzenia. W przyszłości zasymulujemy tego rodzaju kamerę z użyciem aproksymacji bardzo małej soczewki. Ostatecznie dodamy do implementacji kamery efekt występujący w rzeczywistych aparatach związany z rozmyciem obiektu w ruchu, z angielskiego motion blur.

W przypadku wirtualnej kamery możemy założyć że materiał światłoczuły znajduje się przed soczewką a nie jak ma to miejsce w rzeczywistym urządzeniu za otworem wpuszczającym światło. Tego rodzaju założenie przyjmowane jest we wszystkich sposobach wizualizacji w grafice komputerowej. By tego dokonać należy zdefiniować przestrzeń projekcji. Promień wyemitowany przez źródło przebywa pewną drogę trafiając do obserwatora i tworząc obraz końcowy zgodnie z wcześniej przyjętymi zasadami. Prawdopodobieństwo trafienia w źródło światła jest bardzo niewielkie, dlatego też przyjmuje się że promień wychodzi od obserwatora i eksploruje scenę do momentu trafienia w źródło światła. W rzeczywistości nie każdy wyemitowany foton dociera do ludzkiego oka, możemy jednak skorzystać z takiego uproszczenia w celu wyznaczenia ścieżki przemieszczania się światła. Pierwszym algorytmem wykorzystującym tą zasadę był Ray Casting [9]. Metoda została wykorzystana przez Turnera Whitteda do stworzenia raytracingu (RT). Algorytm opracowany został w roku 1980[8]. Danymi wyjściowymi dla metody RT jest położenie obserwatora, płaszczyzny rzutowania, opis sceny, na którą składają się obiekty geometryczne i materiały, z których są zbudowane, jak i źródła światła (Rys.2). Celem całego procesu jest obliczenie koloru każdego piksela płaszczyzny rzutowania.



Rys.2 Zasada działania metody raytracing[3]

Rozwiązanie tego typu może uwzględniać jedynie najprostsze odbicia i załamania światła oraz bezpośrednie oświetlenie (Rys.3). Bezpośrednim oświetleniem nazywamy oświetlenie pochodzące od źródła.



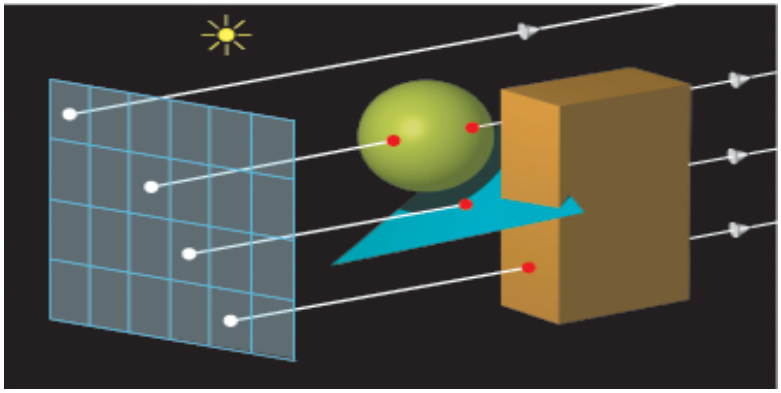
Rys.3 Pierwszy raytracingowy obraz z 1980 wyrenderowany przez Whitteda [8]

3.1 Kamera ortogonalna

Aby wygenerować pierwszy obraz wykonamy rendering w rzucie ortogonalnym (równoległym). W rzutowaniu równoległym bryła widzenia ma kształt prostopadłościenny (Rys.4). Takie rzutowanie jest odpowiednie dla zobrazowania obiektów, dla których chcemy uniknąć zniekształcenia perspektywą. Rzutowanie takie jest przydatne w

[3] Źródło wikipedia [3in]

programach CAD i rysunkach architektonicznych, w których chcemy zachować właściwe rozmiary i proporcje obiektów.



Rys.4 Rzut ortogonalny

Wykonajmy eksperyment wysyłając promień ze środka każdego piksela płaszczyzny rzutowania w kierunku sceny. Płaszczyzna powinna być ustawiona na środku układu współrzędnych. By zlokalizować środek piksela należy wykonać prostą operację:

```
szerokoscPixela = 2.0f / ROZDZIELCZOSCPIONOWAOBRAZU;
wysokoscPixela = 2.0f / ROZDZIELCZOSCPOZIOMAOBRAZU;
```

Posiadając wielkość piksela możemy znaleźć jego środek w układzie współrzędnym sceny:

```
srodekX = -1.0f + (x + 0.5f) * szerokoscPixela;
srodekY = 1.0f - (y + 0.5f) * wysokoscPixela;
```

Algorytm renderingu możemy napisać:

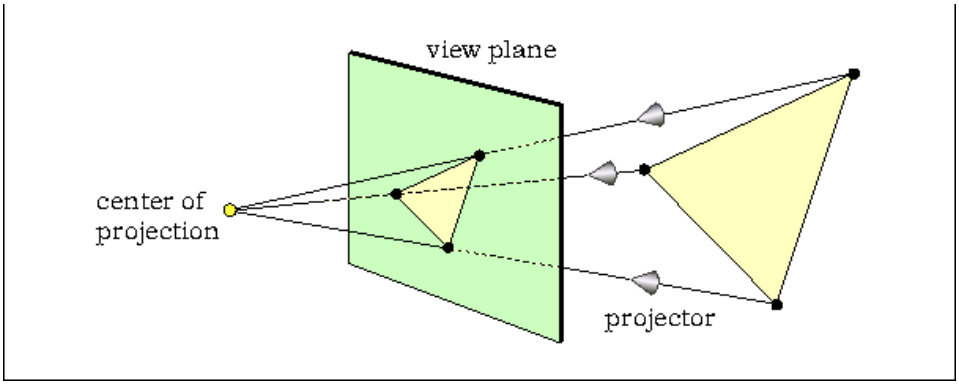
```
for (int i = 0; i < img.Width; i++)
{
      for (int j = 0; j < img.Height; j++)
      {
      srodekX = -1.0f + (i + 0.5f) * widthPixel;
      srodekY = 1.0f - (j + 0.5f) * heightPixel;
      Ray ray = new Ray(new Point(0, 0, 1), new Point(srodekX, srodekY, 0));
      Point intersetion = sfera.intersect(ray);
      if (intersetion != null)
      {
      img.setPixel(i, j, kolor);
      }
      else img.setPixel(i, j, kolortla);
      }
}
img.Bitmap.Save("obrazkoncowy.jpg");
```

Pierwszy obraz powinien zawierać sferę w wybranym kolorze (Rys.5)

Rys.5 Pierwszy wygenerowany obraz

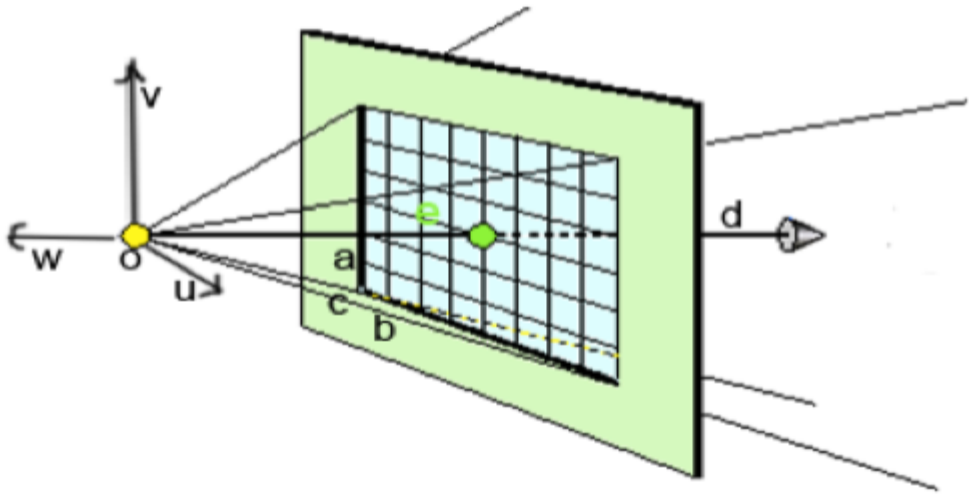
3.2 Kamera perspektywiczna

W rzutowaniu perspektywicznym otrzymujemy efekt zmniejszania się obiektów wraz ze wzrostem ich odległości od obserwatora. Szerokość tylnej płaszczyzny bryły widzenia jest różna od szerokości płaszczyzny przedniej a bryła widzenia ma kształt ostrosłupa. Zauważmy że odległość płaszczyzny rzutowania s jest z góry narzucona; jeśli odległość będzie większa powstały obraz będzie większy i taki też zostanie wygenerowany po skończeniu obliczeń. Wiele systemów renderujących przyjmuje odległość s =, dla uproszczenia obliczeń, my założymy możliwość zmiany parametru s dla przyszłych rozważań nad zmianą ogniskowej. W celu implementacji kamery, ustawiamy płaszczyznę rzutowania, wyrównaną do naszego układu współrzędnych.



Rys.6 Rzut perspektywiczny dla trójkąta

System rzutowania ma określony środek e, który zdefiniowany jest w środku układu współrzędnych uvw i ustawiony w bazie uvw (układzie współrzędnym płaszczyzny rzutowania). W układzie współrzędnych uvw, rogi na przekątnych płaszczyzny rzutowania zdefiniowane są w (u_0,v_0) oraz (u_1,v_1).



Rys.7 Układ współrzędnych rzutowania

Punkt c jest to lewy dolny róg płaszczyzny, wektory a oraz b rozciągają się wzdłuż dolnej i lewej strony płaszczyzny. Są one zdefiniowane jako:

$$a = (u_1 - u_0)\mathbf{u}$$
$$b = (v_1 - v_0)\mathbf{v}$$
$$c = u_0\mathbf{u} + v_0\mathbf{v} - s\mathbf{w}.$$

Zauważmy że płaszczyzna nie musi być wyśrodkowana w kierunku patrzenia (dlatego też nie jest wymagane by u_0=-u_1). Płaszczyzna skierowana jest w przeciwnym kierunku do w, z racji na założenia prawo-skrętnego układu współrzędnych, oraz umiejscowieniu u po prawej stronie płaszczyzny. W celu oddzielenia układu współrzędnych pikseli i kamery, zakładamy że układ współrzędnych okna $[0,1]^2$, gdzie współrzędne są (a,b).
Punkt s na ekranie obliczany jest na podstawie:

$$s = \mathbf{c} + a\mathbf{a} + b\mathbf{b}.$$

Promień z punktu e do s ma więc wartość:

$$\mathbf{e} + t(s - \mathbf{e}).$$

Przechodząc z układu współrzędnych pikseli (x,y) do (a,b), rzutujemy [-0.5, n_x-0.5] na [0,1] oraz [-0.5, n_y-0.5] na [0,1]:

$$a = \frac{x + 0.5}{n_x},$$
$$b = \frac{y + 0.5}{n_y}.$$

Jeśli chcemy zdefiniować widok zorientowany po osiach od pewnego określonego początku, możemy wysyłać promienie według układu współrzędnych xyz. Często istnieje jednakże potrzeba ustawienia położenia i orientacji widoku. Istnieje wiele sposobów realizacji tego zadania, poniżej zostanie opisana jedna z nich.

Istnieją zdefiniowane następujące zmienne:

- e, położenie obserwatora w układzie xyz;
- g, kierunek patrzenia w układzie xyz (jest to także normalna do płaszczyzny rzutowania, dlatego też nazywana jest często jako normalna do płaszczyzny rzutowania);
- v_{up}, wektor skierowany do góry kierunku obserwacji (jest to dowolny wektor przyczepiony na wektorze kierunku patrzenia g, określa on orientację położenia kamery, niejako czubek głowy obserwatora);
- s, odległość od obserwatora e do płaszczyzny rzutowania;
- u_0, v_0, u_1, v_1, w układzie współrzędnych uv rogów płaszczyzny rzutowania.

Wyżej wymienione dane pozwalają na ustalenie uvw frame, gdzie e jest początkiem układu współrzędnych a wektorami bazowymi będą:

$$\mathbf{w} = -\frac{g}{\|g\|},$$

$$\mathbf{u} = -\frac{\mathbf{v}_{up} \times \mathbf{w}}{\|\mathbf{v}_{up} \times \mathbf{w}\|},$$

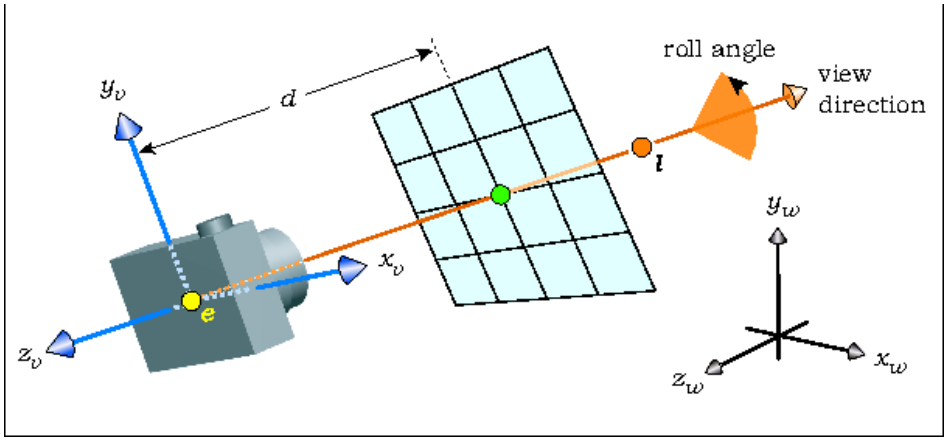
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

Zauważmy że jest to uogólniony system obserwacji, dlatego też istotny jest dobór odpowiednich parametrów. Jeśli chcemy wyrównać płaszczyznę rzutowania tak by jej środek był w miejscu gdzie wychodzi wektor obserwacji, należy upewnić się że: u_0=-u_1 oraz v_0=-u_1. Jeśli chcemy by piksele były kwadratowe należy się upewnić że:

$$\frac{u_1 - u_0}{v_1 - v_0} = \frac{n_x}{n_y}.$$

Zauważmy że s jest niezależne póty dopóki pozostaje dodatnie, gdy jest większe okno zostaje zwiększone, czyli wysokość i szerokość końcowego obrazka. Współczynnik v_1-v_0 do 2s jest tangensem połowy „poziomego pola widzenia”. Jest to jeden ze sposobów ustalania parametrów widzenia.

Najbardziej przyzwyczajeni jesteśmy do definiowania parametrów kamery za pomocą określenia jej pozycji, przedniej i tylniej płaszczyzny obcinania, wektora kierunku obserwacji oraz wektorów określających stopnie swobody.



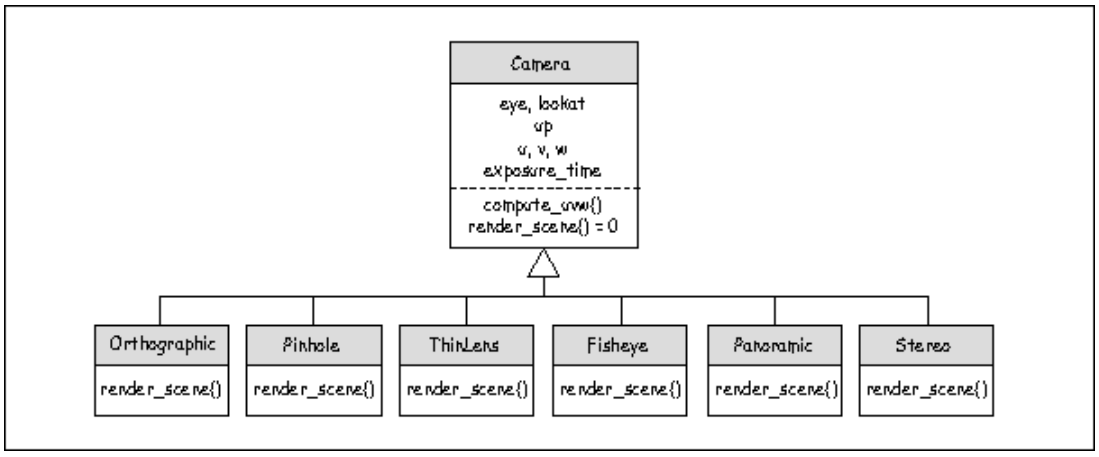
Rys.8 Elementy wirtualnej kamery

Poniższy przykład dotyczyć będzie najprostszej z możliwych definicji kamery. Chcąc zdefiniować w programie zarówno rzut równoległy jak i perspektywiczny należało by zdefiniować abstrakcyjną klasę kamery z której dziedziczyła by kamera perspektywiczna i ortogonalna. Osoby które nie przewidują istnienia kamery ortogonalnej mogą ograniczyć się do najprostszej definicji kamery:

```
public class Camera
{
      private Point position;

      public Point Position
      {
            get { return position; }
            set { position = value; }
      }
      private Vector target;

      public Vector Target
      {
            get { return target; }
            set { target = value; }
      }
      private Vector up;

      public Vector Up
      {
            get { return up; }
            set { up = value; }
      }

      private float nearPlane;

      public float NearPlane
      {
            get { return nearPlane; }
            set { nearPlane = value; }
      }
      private float farPlane;

      public float FarPlane
      {
```

```
        get { return farPlane; }
        set { farPlane = value; }
    }
    private float fov;

    public float Fov
    {
        get { return fov; }
        set { fov = value; }
    }

    public Camera()
    {
        this.position = new Point(0, 0, 0);
        this.target = new Vector(0, 0, 1);
        this.nearPlane = 1;
        this.farPlane = 1000;
        this.up = new Vector(0, 1, 0);
    }

    public Camera(Point position, Vector target)
    {
        this.position = position;
        this.target = target;
        this.nearPlane = 1;
        this.farPlane = 1000;
        this.up = new Vector(0, 1, 0);
    }
```

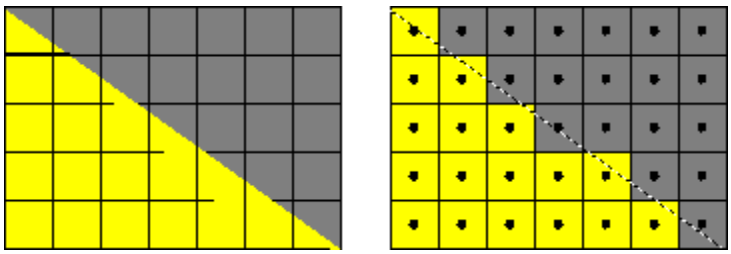
Przewidując możliwość istnienia wielu kamer można nadać jej charakterystyczną nazwę. Podkreślam że przedstawiona definicja jest dla najprostszego modelu kamery. Bardzo częstą praktyką jest definiowanie wewnątrz kamery metody renderingu. Poniższy schemat przedstawia definicję zaawansowanej klasy kamery (Rys.9).



Rys.9 Schemat definicji kamery.

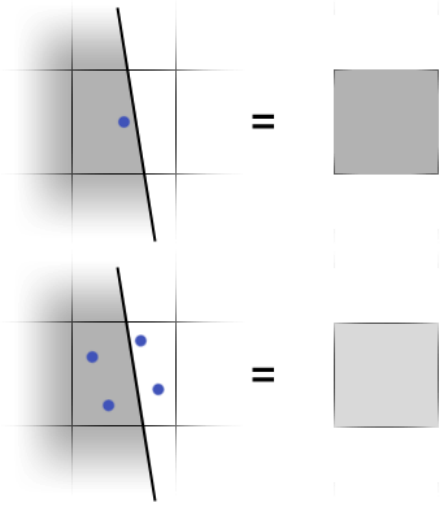
4. Próbkowanie

Pierwszy uzyskany obraz, zawierał sferę wyrenderowaną za pomocą ortogonalnego rzutu. Piksel wysyłany był przez środek piksela. Wcześniejszy przypadek pokazał nam jak stworzyć prosty raytracer z jedną próbką na piksel. Obraz uzyskiwany był przez wysyłanie ze środka każdego kolejnego piksela równolegle promieni. Tego rodzaju rozwiązanie jest szybkie a co za tym idzie dobre dla interaktywnych aplikacji. Nie jest jednak wystarczające dla obrazów wysokiej jakości. Wynikowy kolor piksela będzie średnią ważoną punktów z obrębu całej powierzchni piksela. Postrzępione krawędzie obiektów, pojawiające się w przypadku wysłania jednego promienia na piksel jest rodzajem aliasingu i nosi nazwę „jaggies".



Rys.10 Jak powstaje aliasing

Wyeliminowanie powstałego szumu polega na wysłaniu szeregu promieni na piksel (próbek) i obliczenie koloru wynikowego piksela przez średnią arytmetyczną. W celu obliczenia średniej koloru piksela, należy ustalić dwuwymiarowy układ współrzędnych oparty na pikselach (płaszczyźnie rzutowania).



Rys.11 Obliczanie wartości piksela dla jednej i czterech próbek

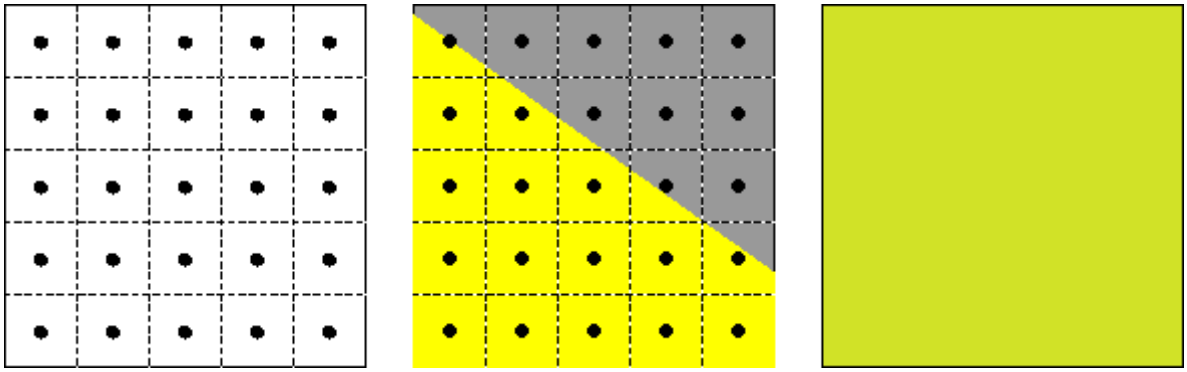
4.1 Metody próbkowania

W celu obliczenia estymatora średniej koloru w około piksela, potrzebny jest algorytm wyznaczania losowych punktów na powierzchni piksela. Istnieje wiele metod do wyznaczania próbek dla piksela. Częsty algorytm polega na wybieraniu próbek $(x,y) \in [0,1] \times [0,1]$ następnie dodawany jest (i-0,5,j-0,5) do wartości (x,y) w celu przesunięcia w około obszaru piksela. Najprościej jest użyć prosty generator losowy do wyboru punktów wewnątrz jednostkowego kwadratu. Następnie punkty mogą zostać zrzutowane na bieżący piksel. Istnieje ryzyko zgromadzenia próbek w jednym obszarze piksela. Daje to ryzyko wyboru nieprawidłowego estymatora do obliczenia średniego koloru piksela, grupując próbki w jednym obszarze piksela. Powinniśmy dążyć do rozsiania próbek po całym obszarze piksela. By zapobiec temu możemy wybrać próbki znajdujące się na przecięciach siatki rozmieszczonej na obszarze piksela. Dodatkowym zyskiem będzie zredukowanie obliczeń z racji na generowanie jednokrotnie ścieżki próbkowania. Stworzenie ścieżki próbkowania (rozkładu próbek) będzie taka sama dla każdego piksela. Regularne próbkowanie niesie jednak ze sobą kolejne zagrożenie wystąpienia aliasingu z racji na powiązanie próbek pomiędzy pikselami. Istnieje wiele sposobów rozkładu próbkowania. Kilka z nich zostanie teraz omówione.

Większość współczesnych sposobów próbkowania polega na wyborze próbek przez podział jednostkowego kwadratu na siatkę n - podobszarów i losowy wybór w każdym z obszarów. Metoda znana jest jako jittered lub stratified sampling. Wybierając jedną z metod rozmieściliśmy próbki w sposób niezależny od rozkładu pikseli. Istotne przy próbkowaniu jest dobre rozmieszczenie próbek tak by się nie pokrywały. Dobór odpowiedniej metody próbkowania to ważny przedmiot badań metod próbkowania punktowego. Dobór odpowiedniego rozkładu próbek przekłada się bezpośrednio na jakość końcowego obrazu. Wyjątek stanowi przypadek bardzo dużej ilości próbek, który eliminuje problem aliasingu ze względu na dużą ilość czynników wpływających na ostateczną wartość koloru piksela.

4.1.1 Regularne próbkowanie

Podstawowym sposobem próbkowania jest próbkowanie regularne polegające na ustalonym podziale każdego piksela na regularną siatkę przez którą wysyłane są promienie w głąb sceny.
Poniższy przykład ilustruje podział piksela na regularną siatkę 5x5, kolor wynikowy zostanie obliczony przez średnią wartość wszystkich wysłanych próbek (Rys.12).



Rys.12 Rozkład próbek na regularnej siatce

Podczas wysyłania promieni, może zostać policzony kolor dla każdego piksela, końcowa wartość dla piksela może zostaje policzona na końcu jako średnia. Należy tu podkreślić że piksel musi mieć określoną wartość koloru. Na powyższym rysunku cały kwadrat reprezentuje piksel a czarny punkt miejsce przejścia promienia.

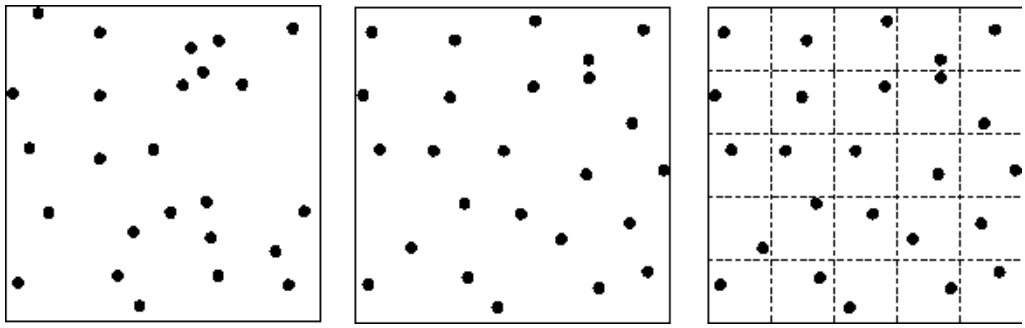
4.1.2 Losowe próbkowanie

Losowy rozkład próbek na powierzchni piksela, pozwala na znaczne zredukowanie aliasngu przy mniejszej ilości promieni. Rozkład próbek nie jest generowany na regularnej siatce. Poniżej znajduje się kod próbkowania losowego powierzchni piksela.

```
for (int p=0; p< vp.num_samples; p++)
{
pp.x=vp.s(c-0.5*vp.hres+rand_float());
pp.y=vp.s(r-0.5*vp.vres+rand_float());
ray.o=Point3D(pp.x,pp.y,zw);
pixel_color+=tracer_ptr->trace_ray(ray);
}
```

4.1.3 Losowe nieregularne próbkowanie (Jittered)

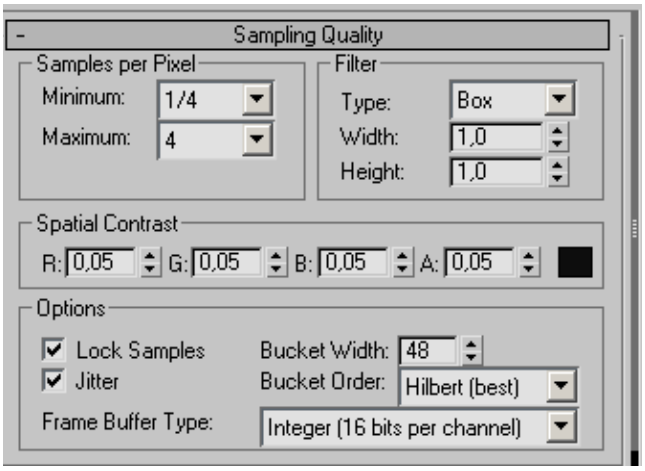
Losowe próbkowanie niesie ze sobą pewne niedogodności, związane z możliwością skupienia się próbek w jednym obszarze, pomijając istotną część piksela. Dobrym rozwiązaniem wydaje się wmuszenie obszarów losowego próbkowania. Piksel dzielony jest na siatkę jednostkową nxn na której generowane będą losowe próbki.



Rys.13 (a) Rozkład 25 próbek (b) Jittered z 25 próbkami (c) Jittered z siatką podziału

4.1.4 Próbkowanie adaptacyjne

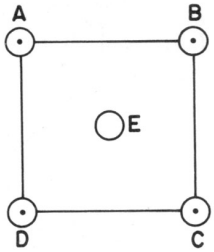
Próbkowanie adaptacyjne jest bardzo popularne w wielu współczesnych systemach renderujących (np. Mental Ray). Idea adaptacyjnego próbkowania polega na dostosowywaniu miejsca i ilości próbek do aktualnie renderowanej sceny w celu otrzymania jak „najlepszego obrazu" przy jak najmniejszej ilości próbek. Poniżej znajduje się okno z opcjami próbkowania dla Mental Ray (Rys.14). Zwróćmy uwagę że przy określaniu próbek na piksel określona jest, nie tyle liczba próbek co ich minimalna i maksymalna wartość.



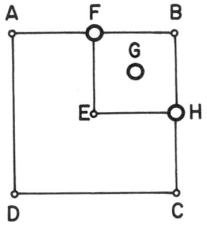
Rys.14 Opcje próbkowania w Mental Ray

Renderer w pierwszym przebiegu wybiera jak najmniejszą liczbę próbek (określoną w minimum) i porównuje obliczone wartości w próbkach, jeśli różnica jest większa niż określona w „spatial kontrast" wybierane są kolejne próbki według algorytmu próbkowania adaptacyjnego. Wartość maksymalna kończy przebieg nawet jeśli warunek końcowy nie został osiągnięty, zabezpiecza to przed zapętleniem się rekurencji.

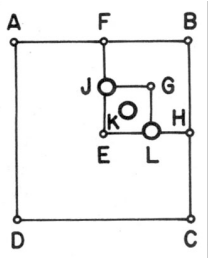
Algorytm działa na zasadzie próbkowania kwinkunksa (z każdego rogu i środka piksela). Przebieg zaczyna się od wysłania promieni z każdego rogu (A,B,D,C)



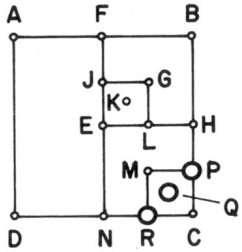
Jeżeli natężenia obliczone w rogach będą się różnić, piksel zostanie podzielony na cztery podobszary.



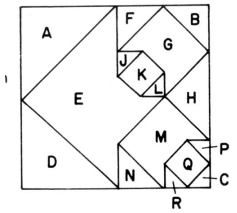
Każdy z nowych obszarów, ma już obliczoną wartość w dwóch rogach, należy wysłać promienie według ustalonego szablonu z brakujących rogów i środka (F,G,H)



Proces zostanie powtórzony (J,K,L) do znalezienia obszaru o zbliżonym natężeniu lub do zadanej wcześniej wielkości.



Po wykonaniu podziału na całej przestrzeni piksela, otrzymamy obszary i zakres wpływu próbki na dany obszar:



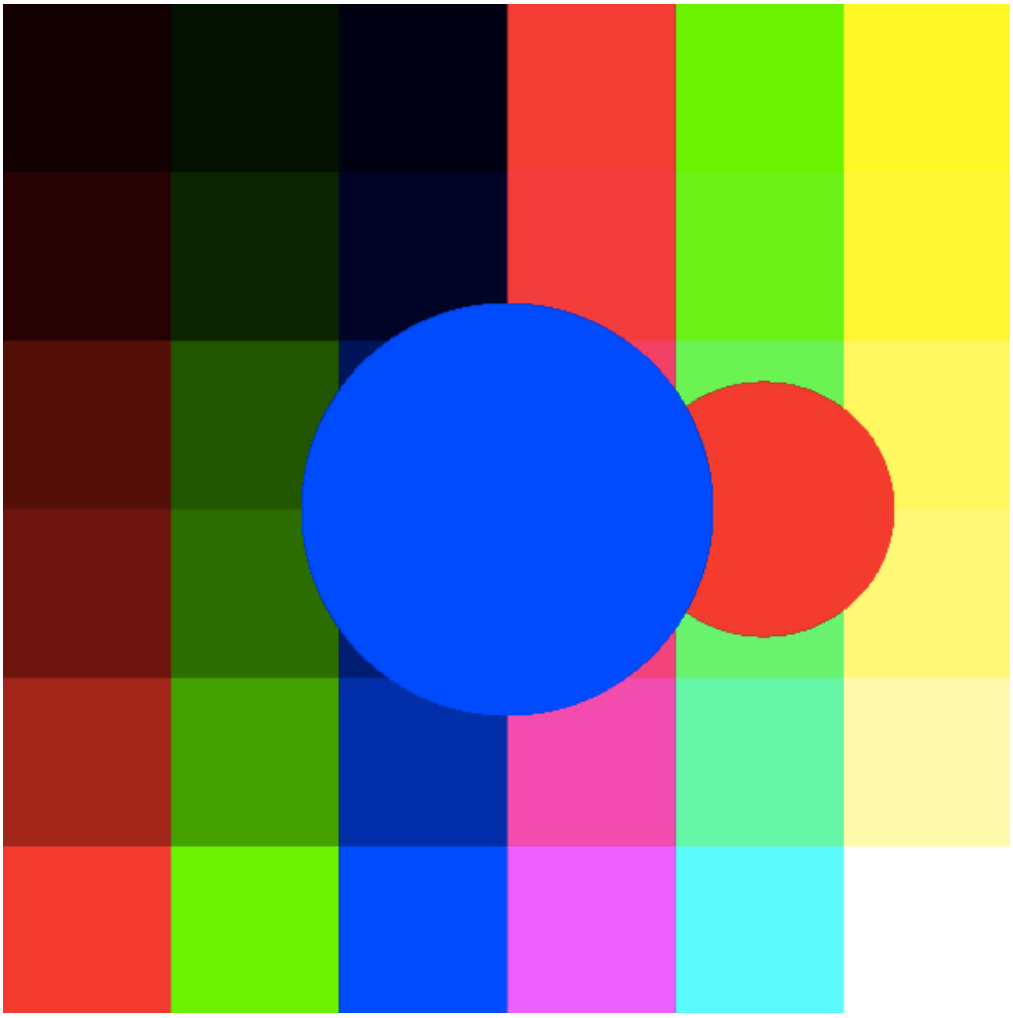
Policzenie wartości koloru piksela, powinno zajść z uwzględnieniem odpowiednich proporcji. We wskazanym przykładzie będzie to:

$$piksel\_color(i) = \frac{1}{4}\left(\frac{A+E}{2} + \frac{D+E}{2} + \frac{1}{4}\left[\frac{F+G}{2} + \frac{B+G}{2} + \frac{H+G}{2} \right.\right.$$
$$\left. + \frac{1}{4}\left\{ \frac{J+K}{2} + \frac{G+K}{2} + \frac{L+K}{2} + \frac{E+K}{2} \right\}\right] + \frac{1}{4}\left[\frac{E+M}{2} + \frac{H+M}{2} + \frac{N+M}{2} \right.$$
$$\left.\left. + \frac{1}{4}\left\{ \frac{M+Q}{2} + \frac{P+Q}{2} + \frac{C+Q}{2} + \frac{R+Q}{2} \right\}\right] \right)$$

Próbkowanie adaptacyjne sprawdza się bardzo dobrze w obszarach gdzie występują krawędzie obiektów. Częstym problemem jest zatrzymanie próbkowania za wcześnie, co oczywiście może doprowadzić do znacznego aliasingu.

5. Zadanie

Proszę zaimplementować klasy obrazu, natężenia i kamery (ortogonalną oraz perpsektywiczną), a następnie wyrenderować obraz, zawierający dwie kule (Rys.15). wykorzystując rzut perspektywiczny i rzut ortogonalny. Należy zaimplementować podaną metodą antyaliasingu adaptacyjnego lub własną, zaproponowaną metodę.



Rys.15 Docelowy wyrenderowany obraz.

6. Bibliografia

[1] **Ashdown Ian, P. Eng., LC, FIES.** Photometry and Radiometry A Tour Guide for Computer Graphics Enthusiasts. : John Wiley & Sons in 1994

[2] **Woźniak Władysław Artur.** Radiometria i Fotometria. : Instytut Fizyki Politechniki Wrocławskiej 2004

[3] **Wynn Chris.** An Introduction to BRDF-Based Lighting : NVIDIA Corporation 2000

[4] **Rusinkiewicz Szymon.** A Survey of BRDF Representation for Computer Graphics : CS348c, Winter 1997

[5] **Glassner, Andrew S.** Space Subdivision for Fast Ray Tracing : IEEE Computer Graphics & Applications, March 1988, volume 8, number 2, pp. 60-70

[6] **Marlon John.** Focus On Photon Mapping : Premier Press 2003 ISBN 1-1-59200-008-8

[7] **Shirley Peter, R. Keith Morley.** Realistic Ray Tracing: Second Edition : AK Peters; 2nd edition (July 2003) ISBN-13: 978-1568811987

[8] **Whitted Turner.** An improved illumination model for shaded display : Communications of the ACM archive. Volume 23 , Issue 6 (June 1980)

[9] **Appel Arthur.** The notion of quantitative invisibility and the machine rendering of solids : Proceedings of ACM National Conference 1967

[10] **Sunday Dan.** Intersections of Rays, Segments,Planes and Triangles in 3D : softSurfer 2006

[11] **Maciej Falski** Przegląd modeli oświetlenia w grafice komputerowej: Praca magisterska Uniwersytet Wrocławski Wydział Matematyki i Informatyki, Instytut Informatyki 2004

[12] **Christophe Schlick.** An inexpensive BRDF model for physically-based rendering : Computer Graphics Forum 1994

[1in] **Dr inż. Władysław Artur Woźniak.** Strona informacyjna Instytut Fizyki. http://www.if.pwr.wroc.pl/~wozniak/

[2in] **The Australian National UniversityFaculty of Engineering and Information Technology (FEIT) .** Global Illumination ModelsPhysically Based Illumination, Ray Tracing and Radiosity : http://escience.anu.edu.au/lecture/cg/GlobalIllumination/printNotes.en.html

[4in] **Softsurfer.** List of Algorithm Titles in the softSurfer Archive. http://softsurfer.com/algorithm_archive.htm

[5in] **Henrik Wann Jensen.** Strona informacyjna Henrika Wann Jensena. http://www.gk.dtu.dk/~hwj

[6in] **Jiajun Zhu** CS 645 Computer Graphics http://www.cs.virginia.edu/~jz8p/