Ray Tracer cz.1

Michał Chwesiuk

Zachodniopomorski Uniwersytet Technologiczny w Szczecinie
Wydział Informatyki

4 Kwiecień 2017

Plan zajęć laboratoryjnych

- Podłączanie bibliotek
  Zapis danych do pliku graficznego
- Generowanie promienia pierwotnego
  Import sceny z pliku
- Algorytm ray tracingu
  Obliczanie przecięć z kulą
- Obliczanie przecięć z trójkątem
  Korekcja Gamma
- Równanie oświetlenia
  Obliczanie wektorów normalnych
  Wyznaczanie cieni
- Rekurencyjne śledzenie promieni odbitych
  Teksturowanie kuli
- Antyaliasing

GLM

- Open**GL** **M**athematics
- Biblioteka zawierająca operacje wektorowe i macierzowe
- Często używana z połączeniem z OpenGL, użyciem przypomina GLSL
- Sama biblioteka jest zamieszczona w plikach header'owych
- http://glm.g-truc.net/0.9.8/index.html

FreeImage

- Biblioteka do obsługi obrazów, m.in. pozwala na ich zapis i odczyt
- Obsługuje takie formaty jak PNG, BMP, JPEG, TIFF i wiele innych
- Składa się z trzech plików : "freeimage.h", "freeimage.lib", i "freeimage.dll"
- http://glm.g-truc.net/0.9.8/index.html

Struktura projektu

- Zachęcam do przemyślenia struktury projektu, to znaczy odpowiedniego dołączenia plików biblioteki w odpowiedni sposób.
- Wszystkie pliki header'owe zachęcam do wstawienia do folderu "<folder projektu>/include".
- Wszystkie pliki bibliotek statycznych (pliki "lib") zachęcam do wstawienia do folderu "<folder projektu>/lib"

Struktura projektu

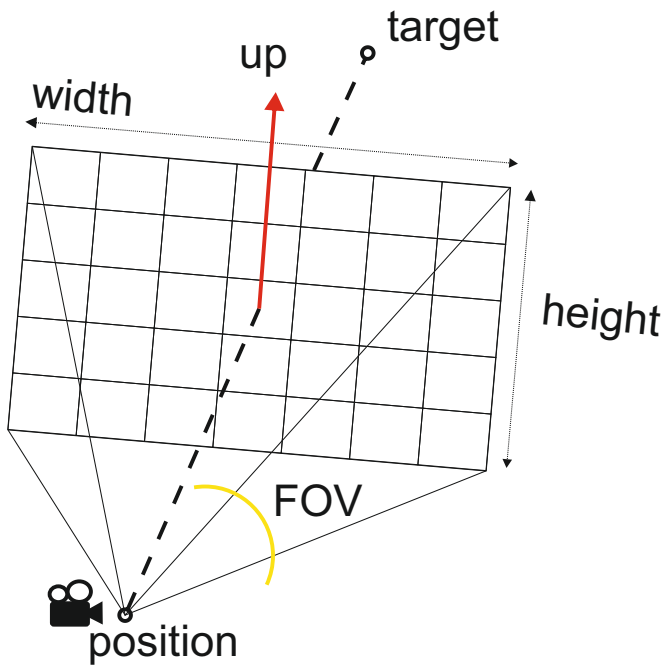
- Aby dodać do środowiska Visual Studio foldery zawierające biblioteki, należy je wskazać we właściwościach projektu
  - Folder zawierający pliki header'owe dodajemy w opcji "Project ->Properties ->Configuration Properties ->C/C++ ->General ->Additional Include Directories"
  - Folder zawierający pliki biblioteki statycznej (pliki "*.lib") dodajemy w opcji "Project ->Configuration Properties ->Linker ->General ->Additional Library Directories"
  - Ddoatkowo, każdy używany plik "*.lib"powinien zostać dodany do listy zawierającej się w (pliki "*.lib") dodajemy w opcji "Project ->Configuration Properties ->Linker ->Input ->Additional Dependencies"
- Zamiast używania ścieżki bezpośredniej ("C:/...) w praktyce korzysta się z makra $(SolutionDir), które wskazuje na folder projektu (np. $(SolutionDir)include)
- Dla ambitnych zachęcam do nauki i skorzystania z NuGet

Zapis danych do pliku graficznego

- Należy opracować klasę CBitmap, która obsługuje zapis do pliku graficznego w formie bitmapy
- Funkcje :
  - **Inicjacja bitmapy**, parametrami funkcji to rozmiary bitmapy, szerokość i wysokość (może to być także konstruktor)
  - **Edytowanie wartości piksela** na bitmapie, parametrami funkcji powinna być pozycja piksela i nowy kolor tego piksela
  - **Zapis bitmapy do pliku**, parametrem powinna być nazwa pliku, do którego bitmapa powinna być zapisana (rozszerzenie BMP)

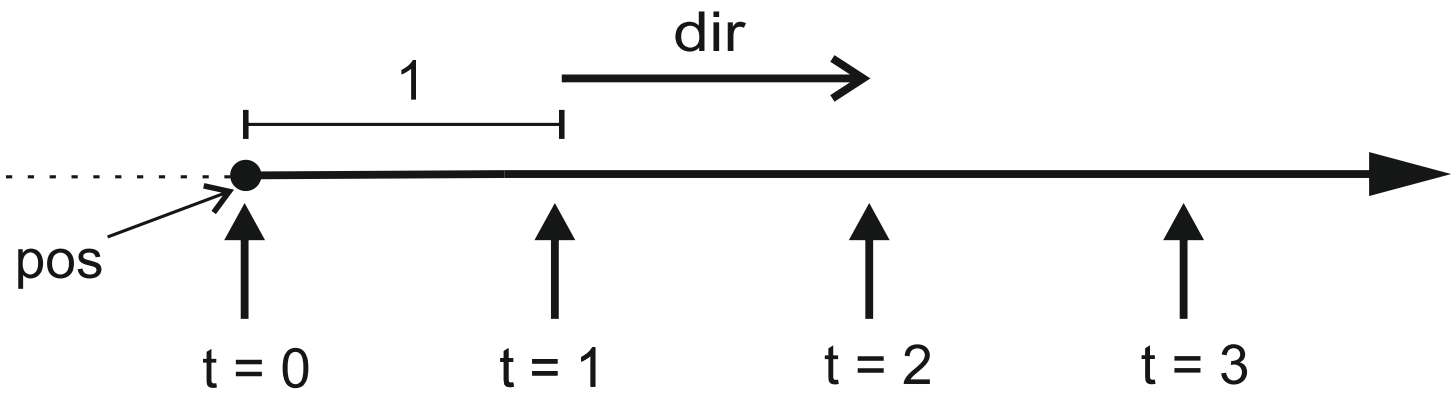
Kamera

- Atrybuty kamery :
  - Pozycja (**position**)
  - Punkt patrzenia (**target**)
  - Wektor góry (**up**)
  - Pole widzenia(Field of View - **FOV**)
  - Ilość pikseli w szerokości bitmapy (**width**)
  - Ilość pikseli w wysokości bitmapy (**height**)



Promień

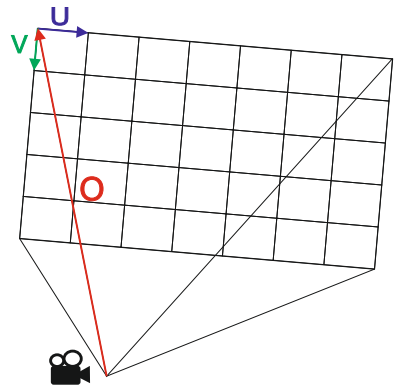
$$ray = pos + t * dir$$



- Atrybuty promienia :
  - **Pozycja**
  - **Kierunek**

Promień pierwotny

- **Promień pierwotny** jest to promień, który zaczyna się w pozycji kamery i przechodzi przez dany piksel obrazu. przez pozycje kamery i dany piksel obrazu.



$$ray.\vec{pos} = \vec{position},\ ray.\vec{dir} = \begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ u_z & v_z & o_z \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

$$i, j - numer\ kolumny, wiersza\ piksela$$

$$\vec{dir} = |\vec{target} - \vec{position}|$$

$$\vec{u} = |\vec{up} \times \vec{dir}|,\ \vec{v} = |\vec{u} \times \vec{dir}|$$

$$\vec{o} = \vec{dir} \cdot \frac{width}{2\ tan(\frac{FOV}{2})} - \frac{width}{2} \cdot \vec{u} - \frac{height}{2} \cdot \vec{v}$$

Import sceny z pliku

Przykładowy loader sceny

```
#include <fstream>
#include <sstream>
int main() {
   std::ifstream file; std::string filename = "scena.txt";
   file.open(filename.c_str(), std::ios::in); if(file.fail()) return 1;
   std::string line;
   while (getline(file, line)) {
      std::istringstream iss(line); std::string type;
      iss >>type;
      if (type.compare("cam_width") == 0) {
         int cam_width; iss >>cam_width; }
   }
   return 0; }
```

Algorytm ray tracingu

Funkcja *run*

```
for each PIXEL
{
 ENERGY = 1.0
 RAY = generate PRIMARY RAY
 COLOR = trace_ray(RAY,background_color,ENERGY)
 set COLOR in PIXEL
}
```

Algorytm ray tracingu

Funkcja *trace_ray*

```
function COLOR trace_ray(RAY, &COLOR, &ENERGY) {
  INTERSECTION = find_intersection(RAY, true)
  if(INTERSETION == null) return COLOR
  generate SHADOW_RAY(INTERSECTION POINT)
  INTERSECTION = find_intersection(SHADOW_RAY, false)
  if(INTERSECTION == null) {
    compute new_color(solve light equation)
    COLOR = COLOR + ENERGY * new_color }
  decrease ENERGY
  if(ENERGY<small value) return COLOR
  generate SECONDARY_RAY
  trace_ray(SECONDARY_RAY, COLOR, ENERGY)
  return COLOR }
```

Algorytm ray tracingu

Funkcja *find_intersection*

```
function OBJECT find_intersection(RAY, closest_intersection)
{
  INTERSECTION = null
  for each OBJECT in SCENE {
    INTERSETION = is RAY intersect OBJECT
    if(INTERSETION == true) {
      if(closest_intersection == true) {
        INTERSECTION = closest OBJECT
        continue
      } else {
        INTERSETION = OBJECT
        break } }
  return INTERSECTION }
```

Algorytm ray tracingu

Funkcja *trace_ray*

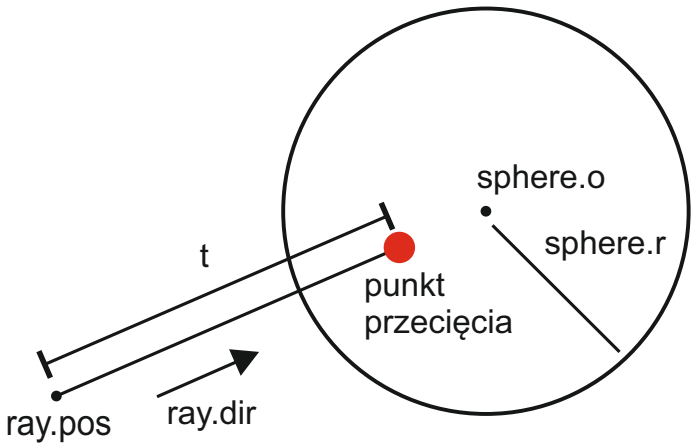
```
function COLOR trace_ray(RAY, &COLOR, &ENERGY) {
  INTERSECTION = find_intersection(RAY, true)
  if(INTERSETION == null) return COLOR
  generate SHADOW_RAY(INTERSECTION POINT)
  INTERSECTION = find_intersection(SHADOW_RAY, false)
  if(INTERSECTION == null) {
    compute new_color(solve light equation)
    COLOR = COLOR + ENERGY * new_color   }
  decrease ENERGY
  if(ENERGY<small value) return COLOR
  generate SECONDARY_RAY
  trace_ray(SECONDARY_RAY, COLOR, ENERGY)
  return COLOR }
```

Obliczanie przecięć z kulą

- **Sferę** określa się za pomocą dwóch atrybutów :
  - **Pozycja** - punkt w przestrzeni określający jej środek
  - **Promień** - odległość zbioru punktów należących do sfery od jej środka
- Aby obliczyć, czy promień przecina się ze sferą (a także punkt przecięcia) należy rozwiązać układ równań

$$\begin{cases} ||x - o||^2 = r^2 & \text{równanie sfery} \\ x = pos + t \cdot dir & \text{równanie promienia} \end{cases}$$

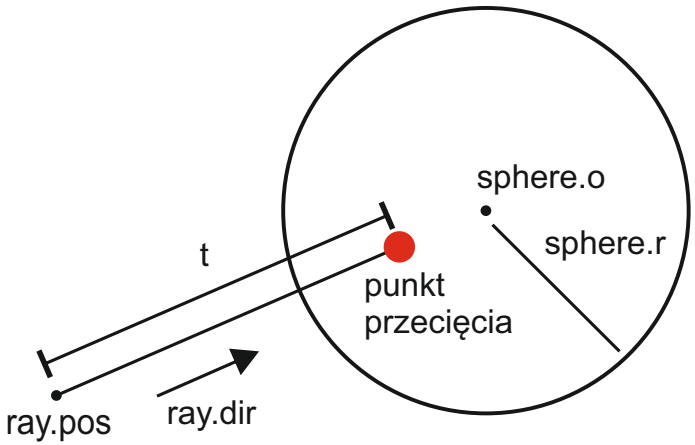
$$||pos + t \cdot dir - o||^2 = r^2$$
$$v = pos - o$$
$$||v + t \cdot dir||^2 = r^2$$
$$v^2 + 2v \cdot t \cdot dir + t^2 dir^2 = r^2$$
$$v^2 + 2v \cdot t \cdot dir + t^2 dir^2 - r^2 = 0$$



Obliczanie przecięć z kulą

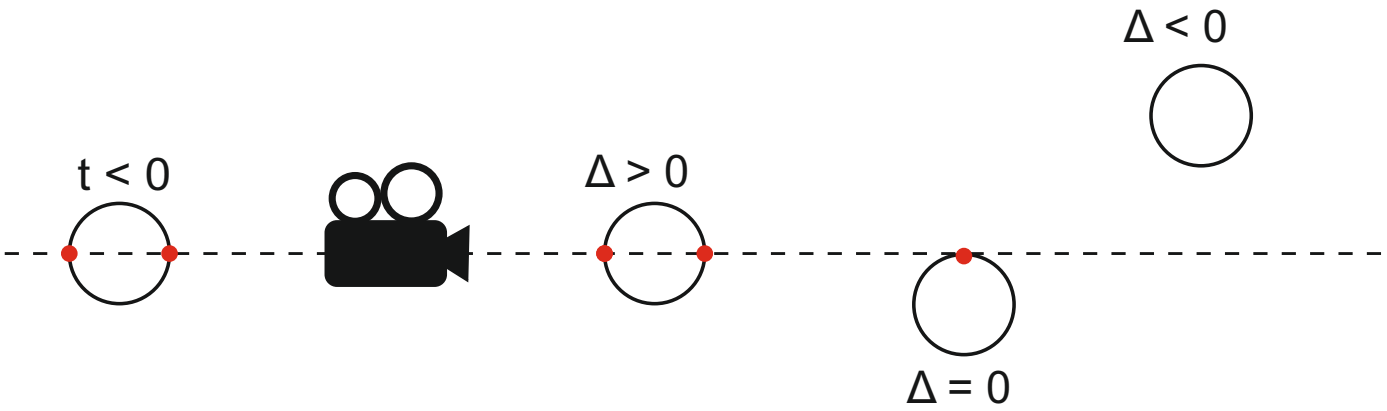
$$(dir^2)t^2 + (2v \cdot dir)t + (v^2 - r^2) = 0$$
$$A \cdot t^2 + B \cdot t + C = 0$$

$$A = dir^2$$
$$B = 2v \cdot dir$$
$$C = v^2 - r^2$$
$$\Delta = B^2 - 4 \cdot A \cdot C$$
$$t = \frac{-B \pm \sqrt{\Delta}}{2 \cdot A}$$



t - odległość od kamery do punktu przecięcia liczona wzdłuż promienia. Wybieramy najmniejszy dodatni parametr t.

Obliczanie przecięć z kulą



Wybieramy najmniejszy dodatni parametr t

Obliczanie przecięć z trójkątem

- **Trójkąt** określa się za pomocą trzech punktów, będącymi wierzchołkami
- Aby obliczyć, czy promień przecina się z trójkątem należy najpierw wyznaczyć równanie płaszczyzny, na której się on znajduje

$$Ax + By + Cz + D = 0$$

, gdzie [A B C] to wektor normalny trójkąta

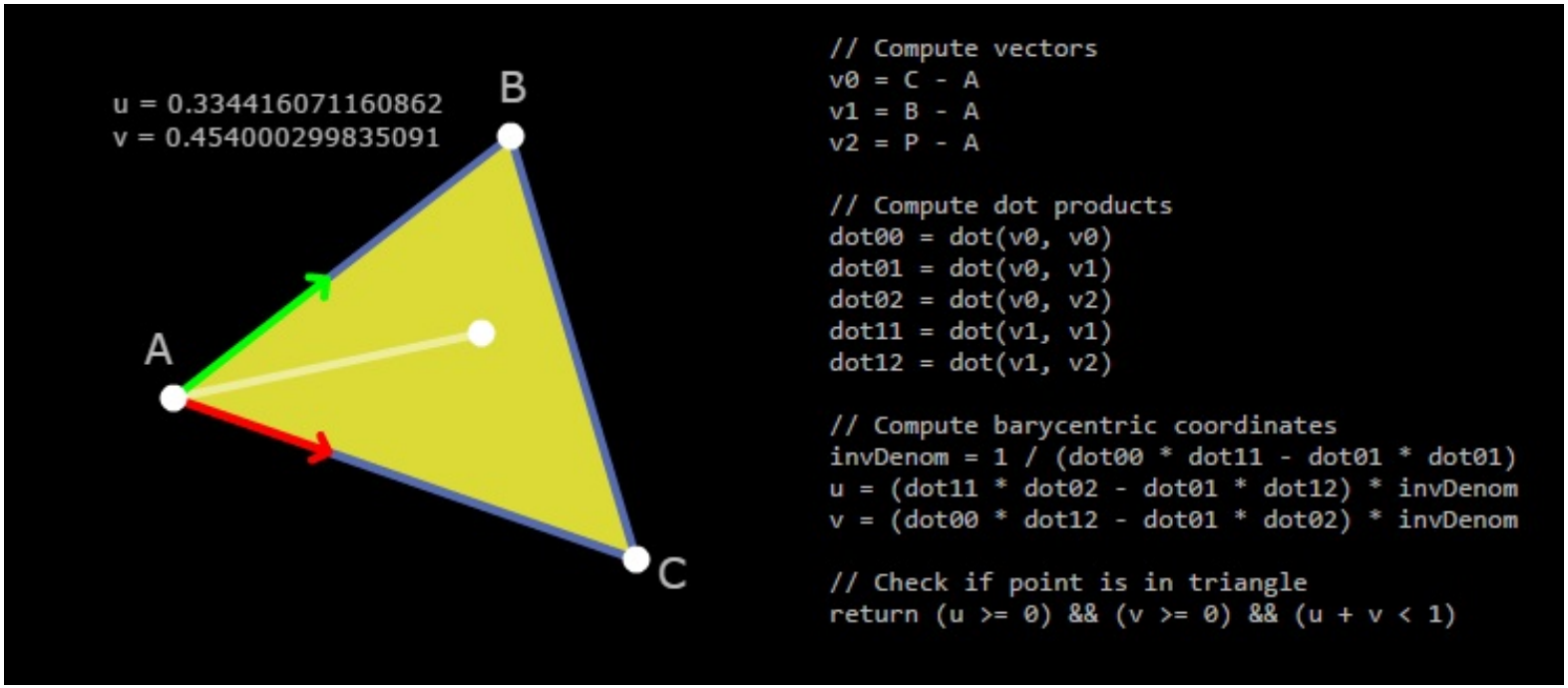
- Kolejnym krokiem jest uzyskanie punktu przecięcia promienia z tą płaszczyzną

$$t = -\frac{A * pos.x + B * pos.y + C * pos.Z + D}{A * dir.x + B * dir.y + C * dir.z}$$

$$P = pos + t * dir$$

Obliczanie przecięć z trójkątem

- Ostatnim etapem jest sprawdzenie czy punkt P leży wewnątrz trójkąta




```
// Compute vectors
v0 = C - A
v1 = B - A
v2 = P - A

// Compute dot products
dot00 = dot(v0, v0)
dot01 = dot(v0, v1)
dot02 = dot(v0, v2)
dot11 = dot(v1, v1)
dot12 = dot(v1, v2)

// Compute barycentric coordinates
invDenom = 1 / (dot00 * dot11 - dot01 * dot01)
u = (dot11 * dot02 - dot01 * dot12) * invDenom
v = (dot00 * dot12 - dot01 * dot02) * invDenom

// Check if point is in triangle
return (u >= 0) && (v >= 0) && (u + v < 1)
```

Korekcja Gamma

Korekcja gamma

for each PIXEL in IMAGE :

// osobno dla kanałów R, G i B

$if\ PIXEL.COLOR < 0.00304$:

$PIXEL.COLOR = PIXEL.COLOR * 12.92$

$else$:

$PIXEL.COLOR = 1.055 * c^{\frac{1}{2.4}} - 0.055$