Raytracing: krok po kroku

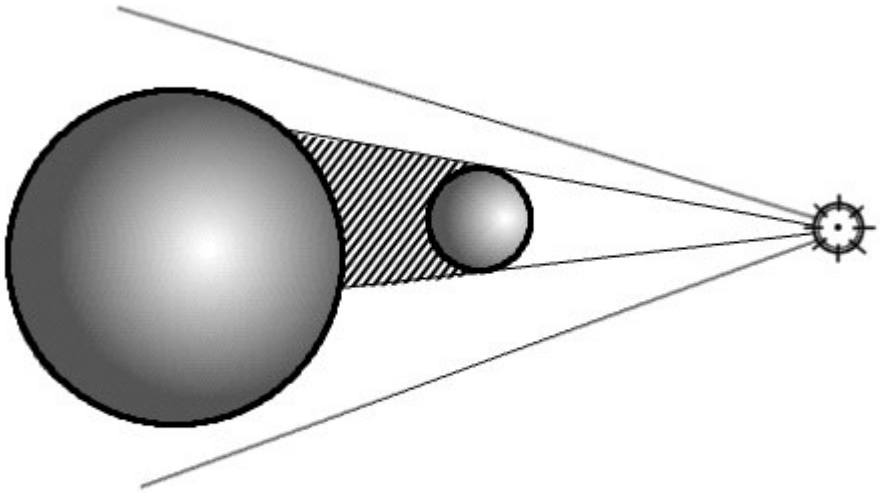
cz. 5 - cień

I. Co/Dlaczego?

Skoro mamy już ładne światło, trochę głupio nie mieć ciemności. Na szczęście dodanie rzucania cieni (przez obiekty dowolnych kształtów!) będzie dużo prostsze niż oświetlenia.
Jeśli masz jakieś wcześniejsze doświadczenie z implementowaniem dymanicznych cieni w grafice trójwymiarowej pewnie myślisz że przesadzam z tą prostotą. Ale nie, nie przesadzam - nie będzie lightmap, shadowmap, volumetric shadows... niczego nie będzie. Właściwie to ta część tak pod względem teoretycznym i jeśli chodzi o kod będzie wręcz banalna - jeśli ktoś nie wierzy zapraszam do czytania.

II. Cień

Najpierw może trochę teorii. Czym jest cień? Cień jest brakiem światła. To było proste pytanie.
No dobrze, wypada trochę rozwinąć. No wiec jak pamiętamy, obecnie w każdym punkcie sprawdzamy wkład wszystkich świateł. Jest tutaj jednak pewna pułapka - nie do każdego punktu dochodzi przecież światło z każdego źródła... Co więc z tym możemy zrobić...? Hmm... Pomyślmy... Możemy sprawdzić czy pomiędzy światłem i cieniowanym punktem znajduje się jakaś przeszkoda!



W rzeczywistości cienie są trójwymiarowe, ale będziemy rozważać je w 2D ponieważ to tam ma miejsce całe cieniowanie.

III. Implementacja

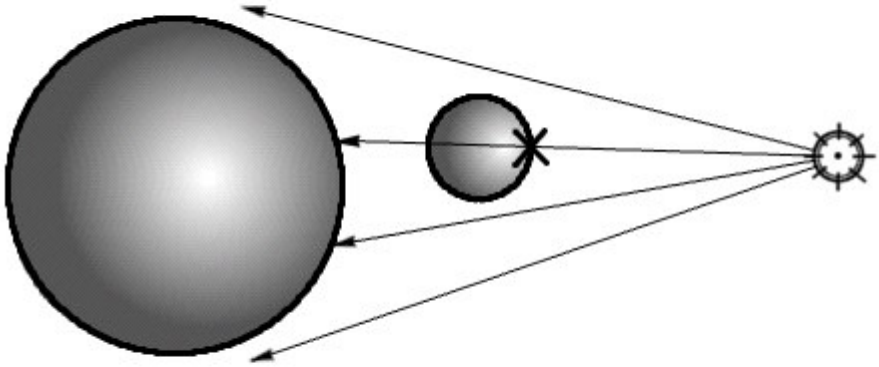
Szybko poszła teoria, ale co robić, trzeba zacząć pisać.
Czyli to czego potrzebujemy to sposób na sprawdzenie czy między dowolnymi dwoma punktami (u nas będzie to cieniowany punkt i światło) znajduje się przeszkoda. Jest to wbrew pozorom bardzo proste. A może wiesz jak to zrobić?

Spróbuj się zastanowić...

Wiesz już?

No więc podpowiedź. Mamy do tego gotową metodę.

No dobrze, jeśli się jeszcze nie domyśliłeś to znaczy pewnie że nawet się nie starałeś. W takim razie zrobię to za Ciebie - metoda TraceRay w klasie World robi dokładnie to czego potrzebujemy. Po prostu wypuszczamy promień ze światła w kierunku punktu (albo odwrotnie, to nie ma znaczenia) i sprawdzamy czy pierwszą rzeczą na jaką trafia jest cieniowany obiekt.



Szukanie obiektu najbardziej pasuje do klasy World więc tam właśnie umieścimy nasz kod:

```
public bool AnyObstacleBetween(Vector3 pointA, Vector3 pointB)
{
    // odległość od cieniowanego punktu do światła
    Vector3 vectorAB = pointB - pointA;
    double distAB = vectorAB.Length;
    double currDistance = Ray.Huge;

    // promień (półprosta) z cieniowanego punktu w kierunku światła
    Ray ray = new Ray(pointA, vectorAB);
```

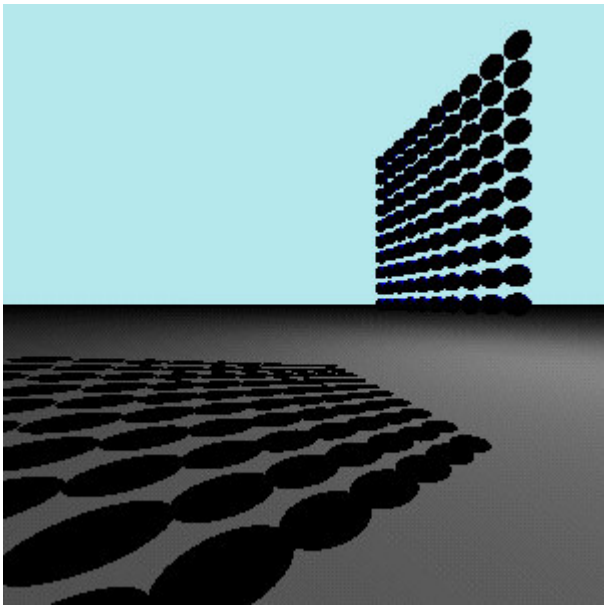
```
    Vector3 ignored = default(Vector3);
    foreach (var obj in objects)
    {
        // jeśli jakiś obiekt jest na drodze promienia oraz trafienie
        // nastąpiło bliżej niż odległość punktu do światła,
        // obiekt jest w cieniu
        if (obj.HitTest(ray, ref currDistance, ref ignored) && currDistance < distAB)
        { return true; }
    }

    // obiekt nie jest w cieniu
    return false;
}
```

Hmm, proste. Teraz wystarczy tego użyć w funkcji sumującej światło (w klasie Raytracer)

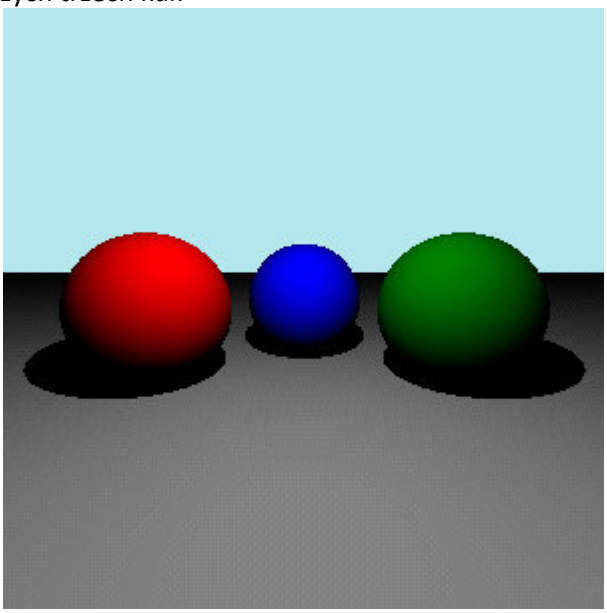
```
public ColorRgb ShadeRay(World world, Ray ray)
{
    HitInfo info = world.TraceRay(ray);

    if (info.HitObject == null) { return world.BackgroundColor; }

    ColorRgb finalColor = ColorRgb.Black;
    IMaterial material = info.HitObject.Material;

    foreach (var light in world.Lights)
    {
        if (world.AnyObstacleBetween(info.HitPoint, light.Position)) { continue; }

        finalColor += material.Radiance(light, info);
    }

    return finalColor;
}
```

Gotowe - to było proste!

Możemy zastosować ten sposób do renderowania cieni na dowolnie skompilkowanych scenach:



A oto jak zmienia wię wygląd naszych trzech kul:



Przynajmniej widać już że kule nie latają w przestrzeni tylko stoją na naszej płaszczyźnie.
W następnej części zajmiemy się porządniejszymi materiałami dla naszych obiektów i zamienimy ten papier w coś trochę błyszczącego.