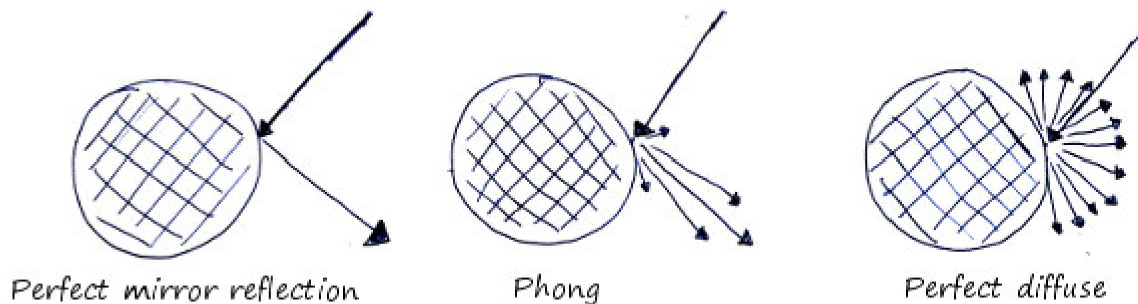
Raytracing: krok po kroku

cz. 6 - model Phonga

I. Co/Dlaczego?

Model oświetlenia Phonga, którym będziemy się zajmować w tej części, został przedstawiony przez Phong Bui-Tuonga w jego rozprawie doktorskiej w roku 1973.
Model ten przyjmuje, że obiekt jest pokryty cienką przezroczystą warstwą, na której zachodzi odbicie lustrzane, natomiast tuż pod tą warstwą następuję odbicie rozproszone, które zabarwia światło na odpowiedni kolor. Mimo że model ten nie jest dokładny fizycznie, bardzo dobrze przybliża wygląd obiektów takich jak błyszczące plastiki i lakierowane ściany.

Materiały Phonga różnią się od materiałów perfekcyjnie rozpraszających tym, że więcej światła jest odbijanego w kierunku lustrzanego odbicia niż w pozostałych kierunkach. Przedstawia to, umieszczany już wcześniej, rysunek:



II. Model Phonga

Oświetlenie dzieli sie na niezależne (*viewer - independent*) i zależne (*viewer - dependent*) od obserwatora. Stworzony przez nas wcześniej model perfekcyjnie rozpraszającego materiału jest doskonałym przykładem oświetlenia niezależnego od obserwatora - jeśli popatrzymy na obiekt z innego punktu (nie zmieniając oświetlenia), każdy punkt tego obiektu zachowa swój poprzedni kolor.

Model Phonga czyni obiekty lśniącymi pozwalając im, do pewnego stopnia, odbijać światło - co jest widoczne jako jasne rozbłyski (*specular highlights*) na powierzchni.
Jest przykładem oświetlenia zależnego od obserwatora - przy przemieszczaniu kamery rozbłyski wydają się przemieszczać.

Rozbłysk jest zależny od kąta pomiędzy promieniami odbitymi od źródła światła, a kierunkiem do obserwatora. Najsilniejszy rozbłysk występuje w miejscu gdzie kąt ten jest równy zero (promień odbity od powierzchni leci 'prosto do oka'), ale szybko słabnie wraz z jego wzrostem.

specular highlight na powierzchni sfery.

Tak więc wiemy od czego zależy jasnośc rozbłysków na powierzchni materiału, ale dalej nie wiemy w jaki sposób je symulować. Zacznijmy od tego jakich danych potrzebujemy podczas cieniowania:

- Pozycja światła
- Pozycja punktu w który trafił promień
- Normalna cieniowanej powierzchni
- Pozycja obserwatora

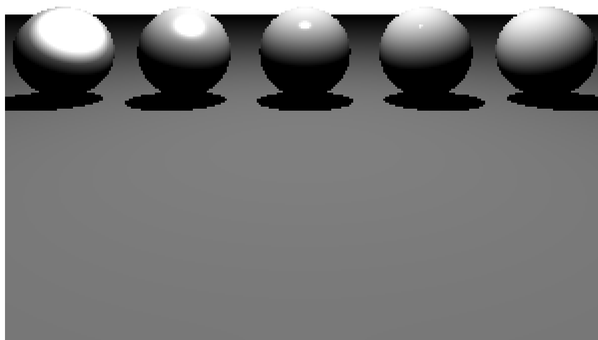
I tyle nam wystarczy. Jak widać pojawił się nowy, w porównaniu do światła rozproszonego punkt - jest to pozycja obserwatora.

Cieniując, zaczniemy podobnie jak w przypadku modelu Lamberta - obliczymy kierunek wpadającego światła i sprawdzimy jaki kąt (cosinus kąta) tworzy z normalną powierzchni. Jeśli ujemny - możemy od razu zwrócić kolor czarny i nie martwić się dalej.
Następnie dalej nic nowego - liczymy współczynnik światła rozproszonego: `light.Color * materialColor * diffuseFactor`.
Ale tym razem to nie koniec. Ostatni krok wygląda tak: `result += materialColor * phongFactor`.

Tajemniczy *phongFactor* jest głównym bohaterem tej części. Jest, podniesionym do pewnej potęgi, *cosinusem kąta* zawartego pomiędzy kierunkiem *odbitego światła* a kierunkiem *do obserwatora*:

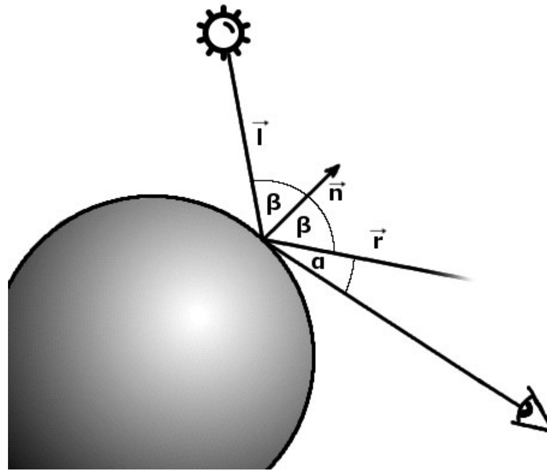
$$phong = \cos^{\exp} \alpha$$

Wspomniana potęga (oznaczona we wzorze jako exp) jest większym od zera parametrem zwanym wykładnikiem rozbłysku (specular exponent) kontrolującym jego wygląd:



ta sama kula renderowana z exp = 1, 10, 100, 1000, 10000

Kierunek do obserwatora już znamy (pozycja obserwatora - pozycja punktu trafienia) - jak obliczyć kierunek odbitego światła? Okazuje się że wystarczy odbić kierunek wpadającego światła (który znamy) od normalnej powierzchni (którą również znamy).



Tak więc zostaje pytanie - jak odbić wektor?

III. Odbijanie wektora

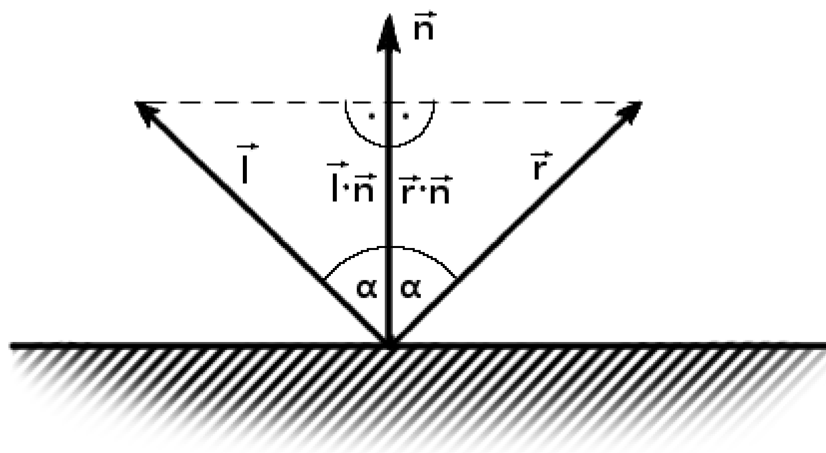
W tym podrozdziale będzie trochę matematyki na wektorach - jeśli kogoś to przeraża, można przeskoczyć od razu na koniec gdzie jest podana gotowa funkcja.
Tak więc chcemy odbić wektor wpadającego światła (l) od normalnej (n) żeby otrzymać wektor światła odbitego (r).

Zauważmy najpierw że odbijane i odbity wektor leżą wszystkie na jednej płaszczyźnie - wynika z tego że możemy zapisać każdy z nich za pomocą kombinacji liniowej pozostałych dwóch. Zapiszmy więc:

$$r = al + bn$$

Gdzie a i b to pewne liczby. Do ich wyznaczenia potrzebujemy jeszcze dwóch równań. Pierwsze równanie możemy ułożyć zauważając że zasadniczą cechą odbicia jest to że kąt pomiędzy wektorami r i n jest równy kątowi pomiędzy l i n. Wynika z tego że równe są ich projekcje (wykonywane za pomocą dot productu) na wektor n.

$$r \cdot n = l \cdot n$$



Po wykonaniu dot productu po obydwóch stronach podstawowego równania otrzymujemy:

$$r \cdot n = al \cdot n + bn \cdot n$$

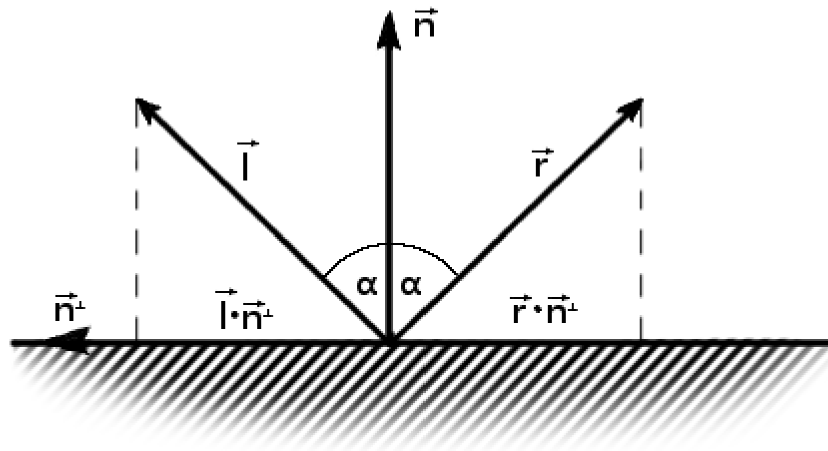
$$l \cdot n = al \cdot n + bn \cdot n$$

$$r \cdot n - al \cdot n = bn \cdot n$$

$$(1-a)l \cdot n = b$$

Możemy również wykonać projekcję wektorów r i l na wektor leżący na płaszczyźnie od której odbijamy (będący prostopadły do n).

$$r \cdot n^{\perp} = -l \cdot n^{\perp}$$



Znowu podstawiając do obydwóch stron równania:

$$r \cdot n^{\perp} = al \cdot n^{\perp} + bn \cdot n^{\perp}$$
$$-l \cdot n^{\perp} = al \cdot n^{\perp} + bn \cdot n^{\perp}$$
$$l \cdot n^{\perp} + al \cdot n^{\perp} = 0$$
$$(1+a)l \cdot n^{\perp} = 0$$
$$a = -1$$

Łącząc wyprowadzone wcześniej równania, otrzymujemy:

$$r = -l + 2(n \cdot l)n$$

Teraz możemy to zapisać w obiecanej funkcji (należącej do klasy Vector3):

```
public static Vector3 Reflect(Vector3 vec, Vector3 normal)
{
    double dot = normal.Dot(vec);
    return normal * dot * 2 - vec;
}
```

Przy okazji, jeszcze jedna uwaga - będziemy za chwilę potrzebować funkcji do odwracania zwrotu wektora (zamienianie wszystkich jego współrzędnych na liczby przeciwne) którą jakimś sposobem wcześniej pominęliśmy. Musimy więc do wektora dodać taki oto kod:

```
public static Vector3 operator -(Vector3 vec)
{
    return new Vector3(-vec.X, -vec.Y, -vec.Z);
}
```

IV. Implementacja

Wiemy wszystko czego potrzebujemy, możemy przystąpić do zamieniania tego w kod.

Najpierw stwórzmy szkielet klasy materiału, pobierający potrzebne parametry:

```
class Phong : IMaterial
{
    ColorRgb materialColor;
    double diffuseCoeff;
    double specular;
    double specularExponent;

    public Phong(ColorRgb materialColor,
        double diffuse,
        double specular,
        double specularExponent)
    {
        this.materialColor = materialColor;
        this.diffuseCoeff = diffuse;
        this.specular = specular;
        this.specularExponent = specularExponent;
    }

    public ColorRgb Radiance(PointLight light, HitInfo hit)
    {
        throw new System.NotImplementedException();
    }
}
```

Parametry jakie przyjmuje nasz materiał to (oczywiście) kolor, współczynnik rozproszenia (na wypadek gdybyśmy chcieli uczynić go mniejszym niż 1 czyniąc światło rozproszone słabszym) współczynnik rozbłysku (specular - kontrolujący, zaskakująco, jasność rozbłysku) i wykładnik rozbłysku (ostrość odbić światła).

Materiały będą niestety przyjmowały z czasem coraz więcej parametrów, i trudno coś na to poradzić (opcja `udostępnij bezparametrowy konstruktor i settery pól` ma swoje zalety, ale również wady - materiał bez ustawionego chociaż jednego pola to nieprawidłowy materiał, a takie błędy może być ciężko wykryć).

Zaczniemy podobnie jak przy świetle rozproszonym:

```
Vector3 inDirection = (light.Position - hit.HitPoint).Normalized;
double diffuseFactor = inDirection.Dot(hit.Normal);

if (diffuseFactor < 0) { return ColorRgb.Black; }

ColorRgb result = light.Color * materialColor * diffuseFactor * diffuseCoeff;
```

Dalej musimy wyliczyć współczynnik phonga i zmodyfikować kolor w oparciu o niego:

```
double phongFactor = PhongFactor(inDirection, hit.Normal, -hit.Ray.Direction);

if (phongFactor != 0)
{ result += materialColor * specular * phongFactor; }
```

Jak wyliczyć współczynnik już wiemy, przytaczając podaną wcześniej definicję:

Współczynnik Phonga jest, podniesionym do pewnej potęgi, cosinusem kąta zawartego pomiędzy kierunkiem odbitego światła a kierunkiem do obserwatora

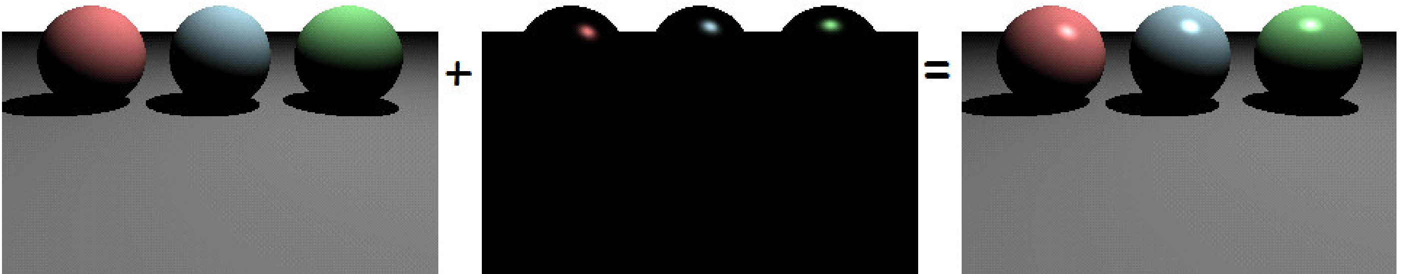
```
double PhongFactor(Vector3 inDirection, Vector3 normal, Vector3 toCameraDirection)
{
    Vector3 reflected = Vector3.Reflect(inDirection, normal);
    double cosAngle = reflected.Dot(toCameraDirection);

    if (cosAngle <= 0) { return 0; }

    return Math.Pow(cosAngle, specularExponent);
}
```

Gotowe (trzeba tylko pamiętać o zwróceniu zera jeśli cosinus kąta jest mniejszy od zera)!
Łącząc w jedną całość, otrzymujemy taką oto funkcję radiancji:

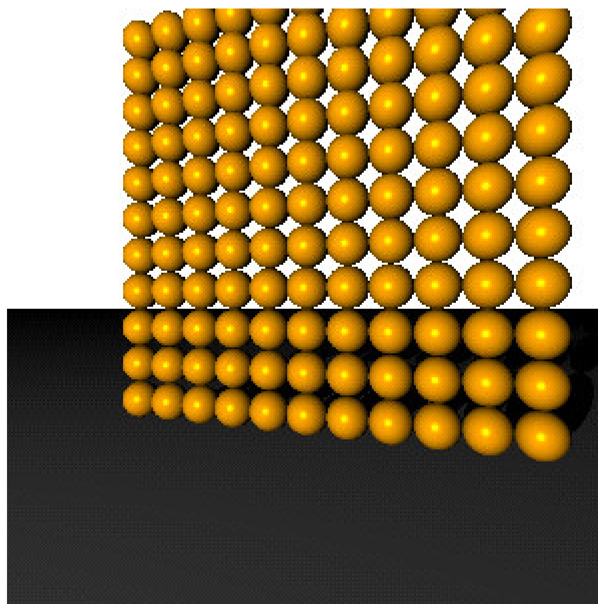
```
public ColorRgb Radiance(PointLight light, HitInfo hit)
{
    Vector3 inDirection = (light.Position - hit.HitPoint).Normalized;
    double diffuseFactor = inDirection.Dot(hit.Normal);

    if (diffuseFactor < 0) { return ColorRgb.Black; }

    ColorRgb result = light.Color * materialColor * diffuseFactor * diffuseCoeff;
    double phongFactor = PhongFactor(inDirection, hit.Normal, -hit.Ray.Direction);

    if (phongFactor != 0)
    { result += materialColor * specular * phongFactor; }

    return result;
}
```

V. Wyniki

Jak wygląda efekt naszej pracy? Znacznie lepiej niż poprzednio (jak na jedną, dość prosta funkcję). Na początku rozłóżmy światło na czynniki pierwsze:

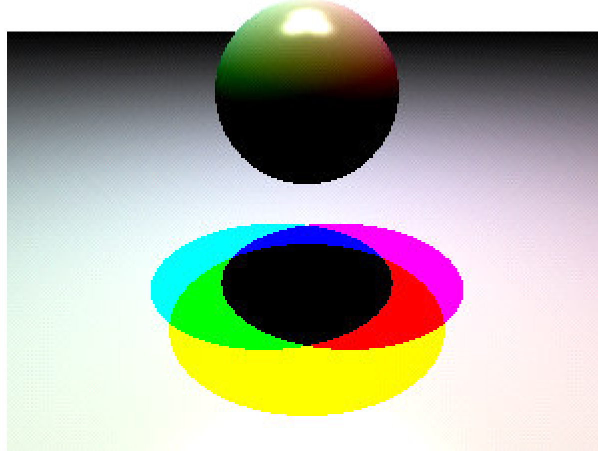


Nie musimy się wcale ograniczać do trzech obiektów na scenie:



Ciekawie wygląda również obrazek pokazujący działanie światła i cienia, który właściwie powinien się znaleźć w

poprzedniej części, ale nie wyglądałby tak przyjemnie (czerwone, zielone i niebieskie światło):



Na koniec, obowiązkowy obrazek naszych klasycznych Trzech Kul Na Płaszczyźnie. Podmieńmy materiały przypisywane obiektom (dodatkowo kolory zostały zmienione na mniej bijące w oczy - mają one też te zaletę że nie są monochromatyczne co przyda się przy odbiciach):

```
IMaterial redMat = new Phong(Color.LightCoral, 0.8, 1, 30);
IMaterial greenMat = new Phong(Color.LightGreen, 0.8, 1, 30);
IMaterial blueMat = new Phong(Color.LightBlue, 0.8, 1, 30);
IMaterial grayMat = new Phong(Color.Gray, 0.8, 1, 30);
```

I w taki oto sposób otrzymujemy: