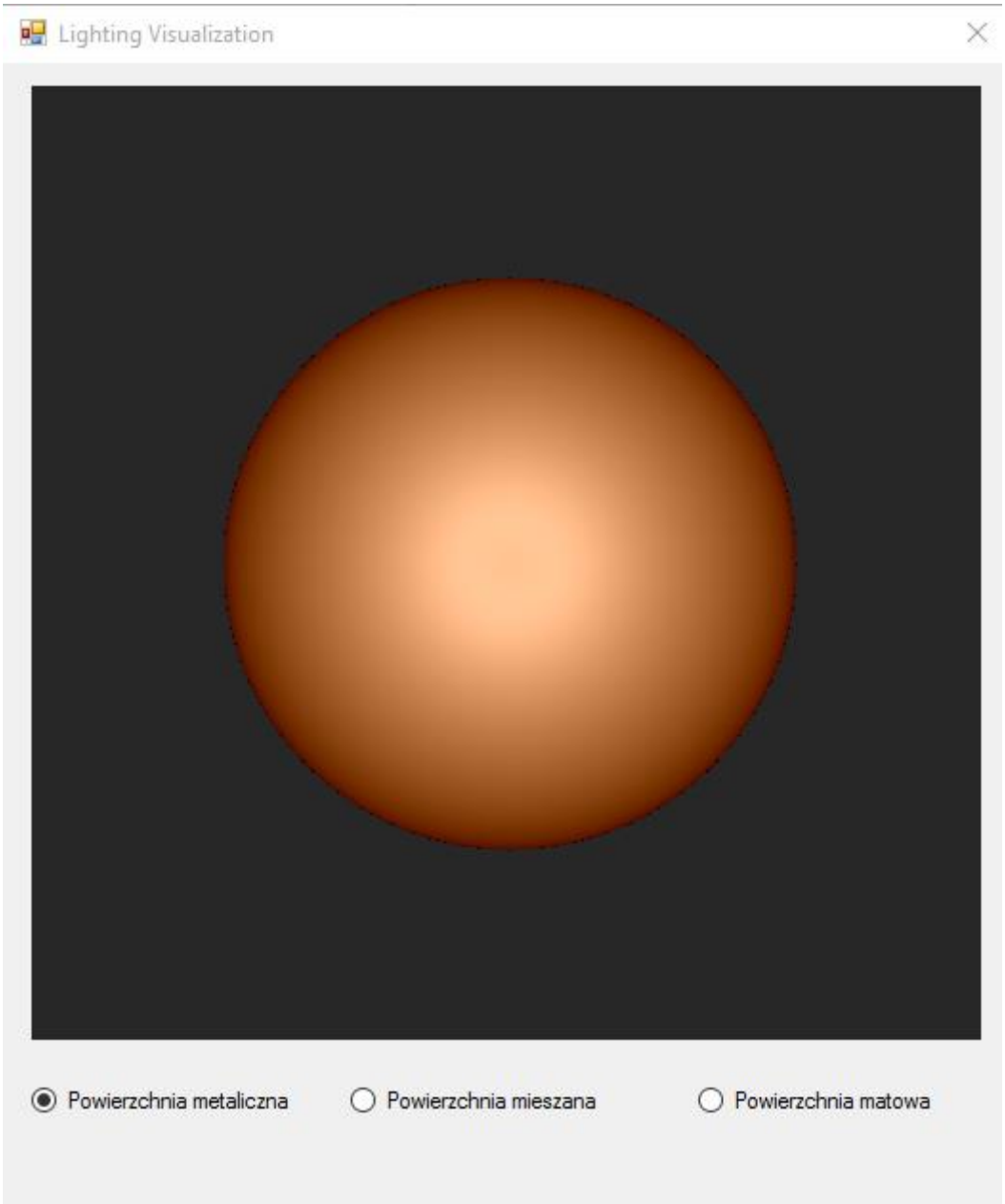

1. Opis projektu

Celem trzeciego projektu było napisanie programu symulującego wizualizację oświetlenia. Obliczenia zawarte w programie powinny prezentować odpowiednio odbicie światła na różnych powierzchniach oraz jego cieniowanie.

2. Wybór modelu

W dostępnej literaturze opisanych jest wiele modeli odbicia światła, najstarszym z praktycznie wykorzystywanych jest **model Phonga** (1975). Model ten jest modelem eksperymentalnym, nieuzasadnionym fizycznie i niespełniającym zasad zachowania energii.

Mimo to jest, chyba, najczęściej stosowanym modelem odbicia w grafice komputerowej, gdyż pozwala szybko uzyskać rysunki o wystarczająco realistycznych barwach.

$$I = I_a * k_a + f_{att} * I_p * k_d * \left(\vec{N} \circ \vec{L}\right) + f_{att} * I_p * k_s * \cos^n(\alpha)$$

I – wynikowe natężenie światła
I_a – natężenie światła w otoczeniu obiektu
I_p – natężenie światła punktowego
k_a – współczynnik odbicia światła otoczenia (tła)
k_s – współczynnik odbicia światła kierunkowego
k_d – współczynnik odbicia światła rozproszonego
f_{att} – współczynnik tłumienia źródła z odległością
n – współczynnik gładkości powierzchni

Dodatkowo użyte zostało również **cieniowanie Phonga**, jako pierwszy element wyznaczania barwy każdego piksela. W pierwszym etapie wyznaczamy wektor normalny w wierzchołku. W drugim etapie wyznaczamy interpolowany wektor normalny dla każdego piksela (to znaczy dla punktu powierzchni odpowiadającego pikselowi). Następnie wyznaczamy barwę piksela, na podstawie interpolowanego wektora normalnego korzystając z wybranego modelu odbicia światła.

3. Opis realizacji

Program został napisany w języku C# z wykorzystaniem silnika graficznego Windows Presentation Foundation (WPF).

Jako model danych posłużyły następujące klasy:

- **Point3D** – klasa reprezentująca punkt w przestrzeni 3D. Posiada punkty X,Y,Z.
- **Vector** - klasa reprezentująca wektor. Posiada trzy współrzędne, X,Y,Z. Dodatkowo klasa udostępnia metody:
    - **Normalize** – normalizacja wektora
    - **Multiply** – mnożenie wektora przez wektor lub wektora przez liczbę
- **Surface** – klasa reprezentująca powierzchnie. Posiada trzy właściwości kluczowe dla powierzchni w tym projekcie, tj.:
    - **Ks –** współczynnik odbicia światła kierunkowego
    - **Kd –** współczynnik odbicia światła rozproszonego
    - **N –** współczynnik gładkości powierzchni (im większy współczynnik powierzchni, tym lepsze właściwości odbicia kierunkowego)

Logika związana z obliczaniem cieniowania oraz odbicia mieści się w klasie **PhongOperator.** Klasa posiada kilka stałych niezbędnych do obliczeń:

- **Ia** – natężenie światła w otoczeniu obiektu
- **Ip** – natężenie światła punktowego
- **Ka** – współczynnik odbicia światła otoczenia (tła)

Klasa posiada również następujące metody:

- **PhongAlgorithm** – główna metoda w klasie, która odpowiada za odpowiednie przeliczanie kolorów wszystkich pixeli. W jej środku korzystamy z metod pomocniczych opisanych poniżej. Na początku w metodzie wykonujemy cieniowanie, następnie odbicie i obliczanie nowych kolorów.
- **CalculateLightReflection**– metoda odpowiadająca za obliczanie odbicia zgodnie z modelem Phonga,

```
        private double CalculateLightReflection(Surface surface, double scalar, double
cosAlpha, Point3D point)
        {
            return Ia * Ka
                    + Fatt(point) * Ip * surface.Kd * scalar
                    + Fatt(point) * Ip * surface.Ks * Math.Pow(cosAlpha, surface.N);
               }
```

- **Fatt** –pozwala na obliczenie współczynnika źródła światła z odległością. Strumień światła pochodzący z punktowego źródła światła maleje z kwadratem odległości jaką przebywa, stąd metoda ma postać:

```
        private double Fatt(Point3D p)
        {
            var distance = Math.Pow(p.X + Source.X, 2) + Math.Pow(p.Y + Source.Y, 2) +
Math.Pow(p.Z + Source.Z, 2);
            return 1.0 / Math.Sqrt(distance);
               }
```

- **CalculateCosAlpha** – opisuje odbicie kierunkowe,

```
        private static double CalculateCosAlpha(Vector v, Vector b)
        {
            var distance = Math.Sqrt(v.X * v.X + v.Y * v.Y + v.Z * v.Z) *
Math.Sqrt(b.X * b.X + b.Y * b.Y + b.Z * b.Z);
            if (Scalar(v, b) > 0) return 0;
            return Scalar(v, b) / distance;
        }
```

- **MoveRight, MoveLeft, Forward, Backward, Up, Down –** metody umożliwiające poruszanie źródłem światła,
- **Prywatne metody pomocnicze ComputeZ, ComputeVector oraz Scalar**, w których umiejscowiona jest logika odpowiedzialna za operacje na wektorach.

GUI oraz obsługa przycisków umiejscowiona jest w klasie **LighitingVisualization.** Jest to klasa, podzielona na część Designer, dzięki której widzimy projektowane okno oraz code-behind odpowiedzialny za logikę wyświetlania.

4. Podsumowanie

Wszystkie funkcjonalności zostały zrealizowane, program umożliwia poruszanie źródłem światła i odpowiednio modeluje cienie i nowe barwy obiektu.

Największym problemem, jaki napotkałem był problem wydajnościowy. Przeliczanie kolorów piksel po pikselu i operacje na bitmapach w języku C# są bardzo kosztowne pamięciowo.