Grafika komputerowa i komunikacja człowiek-komputer

Patryk Jurkiewicz 263896

Ćwiczenie 5 – oświetlanie scen 3-D

1. Cel ćwiczenia

   Celem niniejszego ćwiczenia jest eksploracja technik oświetlania obiektów trójwymiarowych przy wykorzystaniu biblioteki OpenGL wraz z rozszerzeniem GLUT. Poprzez modyfikację programu z poprzednich ćwiczeń, skupiamy się na ilustrowaniu potencjalności oświetlania scen 3D, prezentując proces definiowania materiału, określania źródła światła oraz dostosowywania ich parametrów. Zadaniem jest nie tylko wprowadzenie jednego źródła światła na scenę, ale również zoptymalizowanie oświetlenia poprzez dodanie wektorów normalnych do modelu obiektu, co umożliwia bardziej realistyczną reprezentację interakcji światła z powierzchnią. Cel ćwiczenia obejmuje także rozwinięcie umiejętności programistycznych w zakresie manipulacji parametrami oświetlenia oraz implementację funkcji sterowania położeniem źródeł światła przy użyciu myszy, co pozwala na uzyskanie dynamicznych i interaktywnych efektów wizualnych na obiektach 3D.

2. Oświetlanie scen 3D

   Oświetlanie scen 3D stanowi kluczowy element w grafice komputerowej, mający na celu nadanie trójwymiarowym obiektom realizmu i głębi. W ramach tego ćwiczenia, korzystając z modelu oświetlenia Phonga (Rys 1.), który uwzględnia składowe światła rozproszonego, kierunkowego i otoczenia, eksplorujemy techniki generowania realistycznych efektów oświetlenia na powierzchniach obiektów. Wzory matematyczne modelu Phonga pozwalają na obliczenia intensywności światła w punktach powierzchni, bazując na wektorach normalnych, kierunku źródła światła, kierunku obserwacji oraz współczynnikach materiału. Oświetlanie scen 3D staje się nie tylko narzędziem wizualizacji, ale także umożliwia eksperymentowanie z parametrami oświetlenia, co wpływa na końcowy efekt wizualny i percepcję trójwymiarowej sceny. W zastosowaniach praktycznych oświetlanie to odgrywa ważną rolę w symulacjach graficznych, grach komputerowych oraz wizualizacjach architektonicznych, przyczyniając się do zwiększenia realizmu i immersji oglądającego.

$$I_R = k_{aR} \cdot I_{aR} + \frac{1}{(a + bd_l + cd_l^2)} \left(k_{dR} \cdot I_{dR} \cdot (\overline{N} \cdot \overline{L}) + k_{sR} \cdot I_{sR} (\overline{R} \cdot \overline{V})^n \right)$$

$$I_G = k_{aG} \cdot I_{aG} + \frac{1}{(a + bd_l + cd_l^2)} \left(k_{dG} \cdot I_{dG} \cdot (\overline{N} \cdot \overline{L}) + k_{sG} \cdot I_{sG} (\overline{R} \cdot \overline{V})^n \right)$$

$$I_B = k_{aB} \cdot I_{aB} + \frac{1}{(a + bd_l + cd_l^2)} \left(k_{dB} \cdot I_{dB} \cdot (\overline{N} \cdot \overline{L}) + k_{sB} \cdot I_{sB} (\overline{R} \cdot \overline{V})^n \right)$$

Rys 1.

3. Wektory normalne i geometria obiektów

Wektor normalny oraz geometria obiektów pełnią kluczową rolę w procesie oświetlania scen 3D. Wektor normalny wskazuje kierunek prostopadły do powierzchni obiektu w danym punkcie. W tym ćwiczeniu, dodanie wektorów normalnych do modelu obiektu, takiego jak jajko, stanowi duży krok w poprawie jakości oświetlenia. Wektory te są obliczane na podstawie zależności parametrycznych opisujących geometrię obiektu. W przypadku modelu jajka, zbudowanego na podstawie równań parametrycznych, operacje różniczkowania pozwalają na uzyskanie wzorów do obliczenia wektorów normalnych. Po ich obliczeniu, ważne jest znormalizowanie tych wektorów, czyli doprowadzenie ich do jednostkowej długości, co gwarantuje poprawność obliczeń oświetlenia.

Geometria obiektów, zwłaszcza w kontekście modeli parametrycznych, wpływa bezpośrednio na efekty oświetlenia. W przypadku modelu jajka, zbudowanego z trójkątów powstałych w wyniku triangulacji parametrycznej powierzchni, brak informacji o wektorach normalnych w modelu może prowadzić do utraty detalizacji i realistyczności oświetlenia. Dlatego właśnie, oprócz współrzędnych wierzchołków, konieczne jest dodanie informacji o wektorach normalnych, co pozwala systemowi oświetlania uwzględniać kształt obiektu podczas procesu renderowania.

Wprowadzenie wektorów normalnych i uwzględnienie geometrii obiektów to kluczowe elementy, które umożliwiają osiągnięcie bardziej realistycznego oświetlenia w grafice 3D, co przekłada się na finalny efekt wizualny obserwowanej sceny.

$$x_u = \frac{\partial x(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \cos(\pi v)$$

$$x_v = \frac{\partial x(u,v)}{\partial v} = \pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \sin(\pi v)$$

$$y_u = \frac{\partial y(u,v)}{\partial u} = 640u^3 - 960u^2 + 320u$$

$$y_v = \frac{\partial y(u,v)}{\partial v} = 0$$

$$z_u = \frac{\partial z(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \sin(\pi v)$$

$$z_v = \frac{\partial z(u,v)}{\partial v} = -\pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \cos(\pi v)$$

Rys 2. Wzory wektora normalnego

4. Kod

```cpp
#define _USE_MATH_DEFINES
#include <gl/gl.h>
#include <gl/glut.h>
#include <cmath>
#include <iostream>
#include <stdlib.h>
#include <time.h>

// Poniżej znajduje się deklaracja przestrzeni nazw 'std', co oznacza, że korzystamy z przestrzeni nazw standardowej
using namespace std;

// Deklaracja typu 'point3' jako tablicy trójwymiarowej o elementach typu 'float'
typedef float point3[3];

// Liczba punktów wykorzystywana do generowania powierzchni jajka
int N;

// Stała definiująca promień powierzchni jajka
static GLfloat R = 10.0;

// Pozycja obserwatora w przestrzeni trójwymiarowej
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };

// Punkt obserwacji, służący do ustalenia kierunku widzenia
static GLfloat punkt_obserwacji[] = { 0.0, 0.0, 0.0 };

// Kąty obrotu dla całej sceny
static GLfloat theta_x = 0.0;
static GLfloat theta_y = 0.0;

// Kąty obrotu dla dwóch źródeł światła
static GLfloat theta_x1 = 0.0;
static GLfloat theta_y1 = M_PI;
static GLfloat theta_x2 = 0.0;
static GLfloat theta_y2 = -M_PI;

// Zmienna przechowująca przelicznik do konwersji ruchu myszy na obrót sceny
static GLfloat pix2angle;

// Status myszy (przesunięcie lewego lub prawego przycisku)
static GLint status = 0;

// Pozycja myszy podczas poprzedniego naciśnięcia lewego przycisku
static int x_pos_old = 0;
static int delta_x = 0;
static int y_pos_old = 0;
static int delta_y = 0;
static int z_pos_old = 0;
static int delta_z = 0;
```

W tym fragmencie kodu używane są dyrektywy preprocesora, takie jak #define _USE_MATH_DEFINES, aby umożliwić korzystanie z matematycznych definicji stałych, takich jak π (pi). Dołączone nagłówki, takie jak gl.h, glut.h, cmath, iostream, stdlib.h oraz time.h, dostarczają niezbędne funkcje i deklaracje używane w programie. Zdefiniowano także pewne zmienne globalne, takie jak liczba punktów N, promień powierzchni jajka R, pozycje obserwatora i punktu obserwacji w trójwymiarowej przestrzeni, kąty obrotu dla sceny oraz dla dwóch źródeł światła, a także zmienne związane z ruchem myszy. Te elementy są kluczowe dla generowania trójwymiarowej grafiki jajka oraz obsługi interakcji myszy w programie.

```
// Funkcje definiujące wektory normalne do powierzchni jajka
float wektorNormXU(float u, float v) {
    return (-450 * u * u * u * u + 900 * u * u * u - 810 * u * u + 360 * u - 45) * cos(M_PI * v);
}

float wektorNormXV(float u, float v) {
    return M_PI * (90 * u * u * u * u * u - 225 * u * u * u * u + 270 * u * u * u - 180 * u * u + 45 * u) * sin(M_PI * v);
}

float wektorNormYU(float u, float v) {
    return 640 * u * u * u - 960 * u * u + 320 * u;
}

float wektorNormYV(float u, float v) {
    return 0;
}

float wektorNormZU(float u, float v) {
    return (-450 * u * u * u * u + 900 * u * u * u - 810 * u * u + 360 * u - 45) * sin(M_PI * v);
}

float wektorNormZV(float u, float v) {
    return -M_PI * (90 * u * u * u * u * u - 225 * u * u * u * u + 270 * u * u * u - 180 * u * u + 45 * u) * cos(M_PI * v);
}

// Funkcje definiujące wektory normalne do powierzchni jajka na podstawie wcześniej zdefiniowanych funkcji
float wektorNormX(float u, float v) {
    return wektorNormYU(u, v) * wektorNormZV(u, v) - wektorNormZU(u, v) * wektorNormYV(u, v);
}

float wektorNormY(float u, float v) {
    return wektorNormZU(u, v) * wektorNormXV(u, v) - wektorNormXU(u, v) * wektorNormZV(u, v);
}

float wektorNormZ(float u, float v) {
    return wektorNormXU(u, v) * wektorNormYV(u, v) - wektorNormYU(u, v) * wektorNormXV(u, v);
}

// Funkcje definiujące współrzędne obserwatora w przestrzeni trójwymiarowej
float observerXS(float R, float azymut, float elewacja) {
    return R * cos(azymut) * (float)cos(elewacja);
}

float observerYS(float R, float elewacja) {
    return R * sin(elewacja);
}

float observerZS(float R, float azymut, float elewacja) {
    return R * sin(azymut) * (float)cos(elewacja);
}
```

W tym kodzie znajdują się funkcje matematyczne służące do obliczania wektorów normalnych do powierzchni jajka w trójwymiarowej przestrzeni. Powierzchnia jajka jest opisana za pomocą dwóch parametrów (u, v), które są używane w funkcjach do obliczeń. Każda z funkcji reprezentuje składnik wektora normalnego odpowiadający jednej z współrzędnych (X, Y, Z). Przykładowo, funkcje wektorNormXU, wektorNormXV, wektorNormYU, itd., obliczają składniki wektora normalnego w kierunku osi X, Y, Z w zależności od parametrów (u, v). Dodatkowo, zdefiniowane są funkcje wektorNormX, wektorNormY, wektorNormZ, które korzystają z wcześniej zdefiniowanych funkcji składników wektora normalnego, aby obliczyć pełny wektor normalny dla danej parametrycznej powierzchni jajka. Ponadto, funkcje observerXS, observerYS, observerZS odpowiadają za obliczanie współrzędnych obserwatora w trójwymiarowej przestrzeni na podstawie kątów azymutu i elewacji, co jest istotne w kontekście ustawień kamery. Te funkcje matematyczne są później wykorzystywane w rysowaniu trójwymiarowej grafiki jajka oraz w obszarze manipulacji oświetleniem, reagując na interakcje użytkownika poprzez klawiaturę i mysz. Wspomniane funkcje odgrywają kluczową rolę w procesie generowania realistycznego oświetlenia i perspektywy wizualizacji powierzchni jajka.

```
103  // Funkcja obsługująca zdarzenia myszy
104  void Mouse(int btn, int state, int x, int y)
105  {
106      // Jeśli lewy przycisk myszy został naciśnięty
107      if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
108      {
109          // Zapisujemy aktualne pozycje myszy
110          x_pos_old = x;
111          y_pos_old = y;
112          // Ustawiamy status na 4, co oznacza manipulację światłem 1
113          status = 4;
114      }
115      // Jeśli prawy przycisk myszy został naciśnięty
116      else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
117      {
118          // Zapisujemy aktualne pozycje myszy
119          x_pos_old = x;
120          y_pos_old = y;
121          // Ustawiamy status na 3, co oznacza manipulację światłem 2
122          status = 3;
123      }
124      // Jeśli nie został naciśnięty ani lewy, ani prawy przycisk myszy
125      else
126          // Ustawiamy status na 0, co oznacza brak aktywnej manipulacji
127          status = 0;
128  }
129
130  // Funkcja obsługująca ruch myszy
131  void Motion(GLsizei x, GLsizei y)
132  {
133      // Obliczamy zmiany pozycji myszy
134      delta_x = x - x_pos_old;
135      x_pos_old = x;
136      delta_y = y - y_pos_old;
137      y_pos_old = y;
138      // Informujemy, że należy przerysować scenę
139      glutPostRedisplay();
140  }
```

Ten fragment kodu zajmuje się obsługą zdarzeń myszy w kontekście interakcji użytkownika z trójwymiarową sceną. Funkcja Mouse reaguje na naciśnięcie przycisków myszy, a w zależności od tego, czy został naciśnięty lewy czy prawy przycisk, ustawia odpowiedni status manipulacji. Jeżeli lewy przycisk myszy jest naciśnięty, status ustawiany jest na 4, co sygnalizuje manipulację światłem 1. Natomiast w przypadku naciśnięcia prawego przycisku status ustawiany jest na 3, co oznacza manipulację światłem 2. Jeśli nie został naciśnięty ani lewy, ani prawy przycisk myszy, status ustawiany jest na 0, co oznacza brak aktywnej manipulacji. Z kolei funkcja Motion obsługuje ruch myszy. Oblicza zmiany pozycji myszy i informuje, że scena musi zostać przerysowana (glutPostRedisplay). Te dwie funkcje są kluczowe dla interakcji z programem, umożliwiając użytkownikowi modyfikowanie parametrów sceny poprzez ruch myszy i naciśnięcia przycisków. W tym konkretnym przypadku, umożliwiają regulację dwóch źródeł światła w trójwymiarowej przestrzeni.

```
// Funkcja renderująca scenę
void RenderScene(void)
{
    // Wyczyszczenie buforów koloru i głębokości
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset transformacji
    glLoadIdentity();

    // Ustawienie punktu widzenia
    gluLookAt(viewer[0], viewer[1], viewer[2], punkt_obserwacji[0], punkt_obserwacji[1], punkt_obserwacji[2], 0.0, 1.0, 0.0);

    // Narysowanie osi układu współrzędnych
    Axes();

    // Aktualizacja świateł w zależności od statusu manipulacji myszą
    if (status == 3)
    {
        theta_x1 -= (delta_x * pix2angle / 100.0f);
        theta_y1 += (delta_y * pix2angle / 100.0f);
        GLfloat light_position[] = { observerXS(10.0, theta_x1, theta_y1), observerYS(10.0, theta_y1), observerZS(10.0, theta_x1, theta_y1), 1.0 };
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    }
    else if (status == 4)
    {
        theta_x2 -= (delta_x * pix2angle / 100.0f);
        theta_y2 += (delta_y * pix2angle / 100.0f);
        GLfloat light_position[] = { observerXS(10.0, theta_x2, theta_y2), observerYS(10.0, theta_y2), observerZS(10.0, theta_x2, theta_y2), 1.0 };
        glLightfv(GL_LIGHT1, GL_POSITION, light_position);
    }

    // Obrót obiektu
    glRotatef(theta_x, 0.0, 1.0, 0.0);
    glRotatef(theta_y, 1.0, 0.0, 0.0);

    // Ustawienie koloru na biały
    glColor3f(1.0f, 1.0f, 1.0f);

    // Narysowanie jajka
    Egg(N);

    // Zamiana buforów
    glutSwapBuffers();
}
```

Ta funkcja pełni rolę renderowania sceny, czyli generowania obrazu trójwymiarowej przestrzeni na podstawie aktualnych parametrów i ustawień kamery. Po wyczyszczeniu buforów koloru i głębokości oraz zresetowaniu transformacji, funkcja ustawia punkt widzenia (gluLookAt) zgodnie z pozycją obserwatora i punktem obserwacji. Następnie rysuje osie układu współrzędnych za pomocą funkcji Axes(). W zależności od aktualnego statusu manipulacji myszą, funkcja aktualizuje pozycję dwóch źródeł światła, co umożliwia interaktywną kontrolę oświetlenia sceny przez użytkownika. Ponadto, funkcja dokonuje obrotu obiektu na podstawie zmiennych kątów obrotu theta_x i theta_y. Kolor obiektu ustawiony jest na biały, a następnie rysowane jest jajko za pomocą funkcji Egg(N). Na końcu, bufor koloru jest zamieniany, co powoduje aktualizację widocznego obrazu na ekranie. W skrócie, ta funkcja odpowiada za generowanie dynamicznej trójwymiarowej sceny, uwzględniając manipulacje myszą, oświetlenie, obroty obiektu i renderowanie geometrii jajka.

```
// Funkcja inicjalizująca ustawienia sceny
void MyInit(void) {
    // Ustawienie koloru tła na czarny
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

// Parametry materiału bardziej połyskującego
GLfloat mat_ambient[] = { 0.5, 0.5, 0.5, 1.0 };  // Refleksja otoczenia (ambient reflection)
GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };  // Rozproszenie światła (diffuse reflection)
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 }; // Połysk (specular reflection)
GLfloat mat_shininess = 100.0;                   // Współczynnik połysku (shininess)


    // Pozycje świateł
    GLfloat light_position1[] = { observerXS(10.0, theta_x1, theta_y1), observerYS(10.0, theta_y1), observerZS(10.0, theta_x1, theta_y1), 1.0 };
    GLfloat light_position2[] = { observerXS(10.0, theta_x2, theta_y2), observerYS(10.0, theta_y2), observerZS(10.0, theta_x2, theta_y2), 1.0 };

    // Parametry światła
    GLfloat light_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
    GLfloat light_diffuse[] = { 1.0, 0.0, 0.0, 1.0 };
    GLfloat light_specular[] = { 1.0, 0.0, 0.0, 0.5 };
    GLfloat att_constant = 1.0;
    GLfloat att_linear = 0.05;
    GLfloat att_quadratic = 0.001;

    // Ustawienia materiału
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    // Ustawienia świateł
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position1);

    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant);
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, att_linear);
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, att_quadratic);

    GLfloat light_diffuse1[] = { 0.0, 0.0, 1.0, 1.0 };
    GLfloat light_specular1[] = { 0.0, 0.0, 1.0, 0.5 };

    glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse1);
    glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular1);
    glLightfv(GL_LIGHT1, GL_POSITION, light_position2);

    glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, att_constant);
    glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, att_linear);
    glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, att_quadratic);

    // Model cieniowania
    glShadeModel(GL_SMOOTH);

    // Włączenie obsługi oświetlenia i testu głębokości
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
}
```

Funkcja MyInit pełni istotną rolę w przygotowaniu środowiska graficznego dla trójwymiarowej sceny. Po pierwsze, następuje ustawienie koloru tła na czarny za pomocą glClearColor, co ma wpływ na ogólny wygląd sceny. Kolejnym krokiem jest definiowanie właściwości materiału, który określa, w jaki sposób obiekty na scenie oddziałują z oświetleniem. Parametry, takie jak refleksja otoczenia (mat_ambient), rozproszenie światła (mat_diffuse), połysk (mat_specular) oraz współczynnik połysku (mat_shininess), decydują o wyglądzie powierzchni obiektów. Następnie funkcja określa pozycje dwóch źródeł światła (light_position1 i light_position2), a także definiuje parametry oświetlenia, takie jak składowe ambient, diffuse i specular, oraz współczynniki tłumienia dla świateł punktowych. Te informacje są później przekazywane do systemu OpenGL za pomocą odpowiednich funkcji, takich jak glMaterialfv, glLightfv, glLightf. Dodatkowo, funkcja ustawia model cieniowania (GL_SMOOTH), co wpływa na sposób interpolacji kolorów pomiędzy wierzchołkami obiektów, nadając im bardziej naturalny wygląd. Włącza także obsługę oświetlenia (glEnable(GL_LIGHTING)) oraz testu głębokości (glEnable(GL_DEPTH_TEST)), co jest

istotne dla poprawnego renderowania trójwymiarowych obiektów, eliminując sytuacje, w których obiekty z tyłu zasłaniałyby te z przodu. Ogólnie rzecz biorąc, funkcja MyInit konfiguruje podstawowe właściwości oświetlenia i materiału, dostosowując środowisko renderowania do określonych parametrów, co ma istotne znaczenie dla realistycznego wyglądu trójwymiarowej sceny.

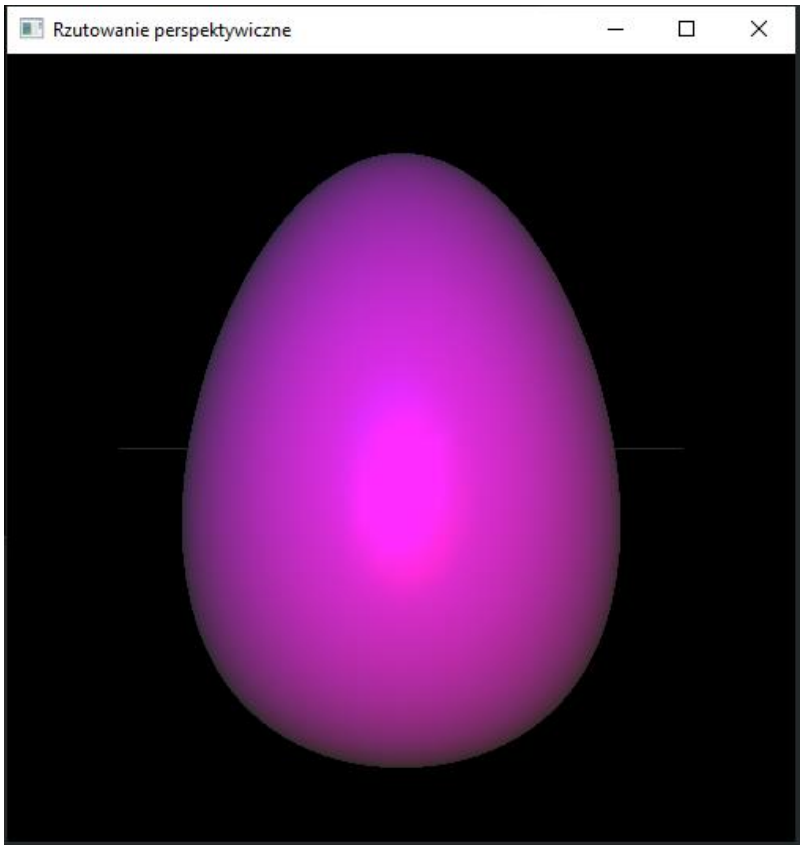
5. Efekty i wnioski

W trakcie realizacji tego zadania związane z programowaniem grafiki trójwymiarowej, zdobyłem obszerne doświadczenie z użyciem biblioteki OpenGL w języku C++. Tworzenie trójwymiarowej sceny przedstawiającej obiekt w postaci jajka, którego kształt opiera się na funkcjach matematycznych, pozwoliło mi nie tylko na praktyczne zastosowanie wiedzy z zakresu geometrii, ale także na zrozumienie skomplikowanych aspektów związanych z oświetleniem, materiałami oraz manipulacją obiektami w przestrzeni trójwymiarowej.

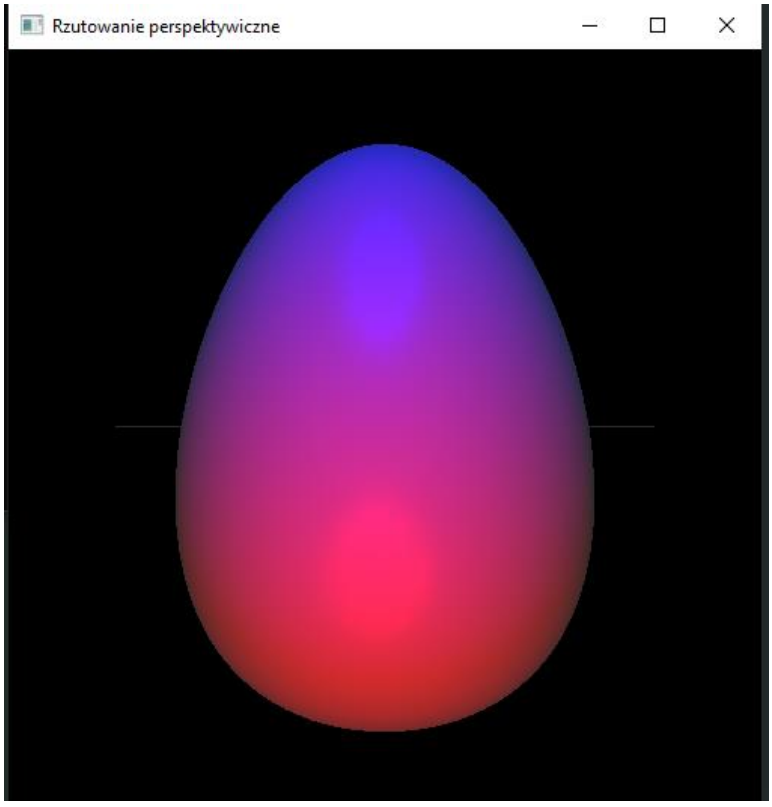
Podczas tworzenia kodu w języku C++, przyswoiłem techniki rysowania obiektów trójwymiarowych, obsługi kamery, a także manipulacji źródłami światła. Ponadto, zrozumiałem znaczenie wektorów normalnych w renderowaniu obiektów, co przyczyniło się do uzyskania bardziej realistycznego oświetlenia.

Ostatecznie, to zadanie pozwoliło mi na pogłębienie umiejętności programowania w kontekście grafiki trójwymiarowej, a także na zastosowanie matematyki w praktyce. Dzięki temu doświadczeniu lepiej zrozumiałem kompleksowe aspekty tworzenia trójwymiarowych renderów, co stanowi cenne zaplecze do dalszego rozwoju w dziedzinie programowania grafiki komputerowej.

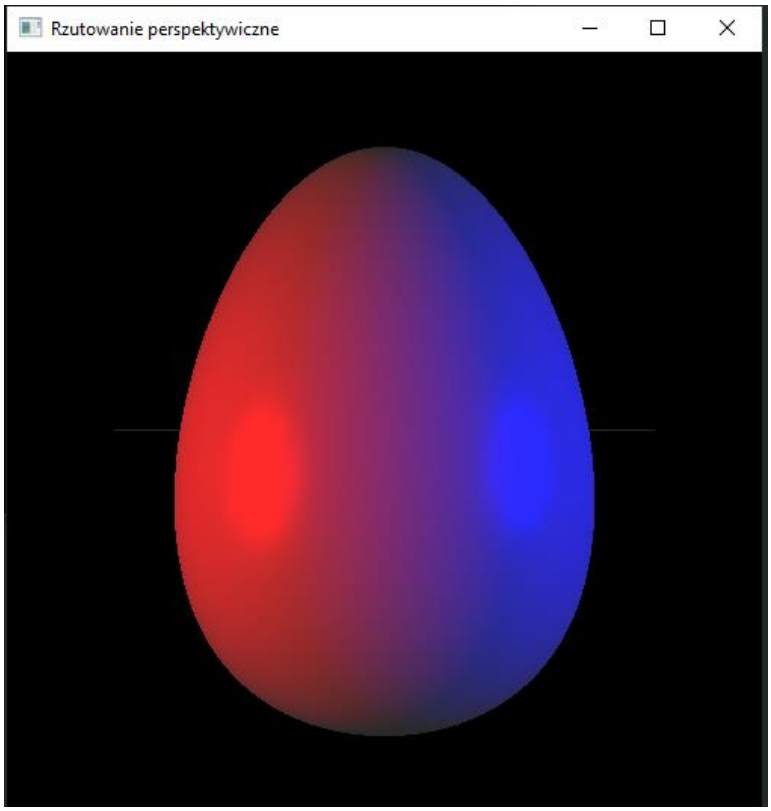
W wyniku wykonania tego ćwiczenia z powodzeniem udało mi się opracować i wdrożyć efektywne rozwiązania dotyczące oświetlenia modelu jajka. Implementacja ta charakteryzuje się realistycznym zachowaniem, co jest szczególnie istotne dla uzyskania wiarygodnych efektów wizualnych. Dzięki zastosowaniu odpowiednich parametrów oświetleniowych oraz ustawień materiałów, udało się osiągnąć efekt, który wzbogaca wizualny aspekt sceny, zwiększając jej autentyczność i atrakcyjność. W procesie tworzenia tej implementacji zdobyłem praktyczne doświadczenie, rozwijając umiejętności związane z programowaniem grafiki komputerowej i zrozumienie złożonych mechanizmów związanych z oświetleniem trójwymiarowych obiektów.



Rys 3. Jajko z nałożonymi na siebie światłami



Rys 4. Jajko z osobno światłami z góry i dołu



Rys 5. Jajko z osobno światłami z lewa i prawa