Kurs:
Grafika komputerowa i komunikacja człowiek-komputer - Laboratorium

OpenGL - oświetlanie scen 3D

Autor:
Aleksandra Chrustek, 263900

Prowadzący:
dr inż. Jan Nikodem

13.12.2023

1 Wstęp

Celem czwartego laboratorium było zapoznanie się z zagadnieniem oświetlania obiektów graficznych na scenie 3D. Do zrozumienia tematu niezbędne było zapoznanie się z nowymi funkcjami bibliotek OpenGL i GLUT, jak również z instrukcją zamieszczoną na stronie ZSK, która pozwalała na osiągnięcie wiedzy postawowej i niezbędnej, aby przystąpienie do ćwiczenia było możliwe.

2 Przebieg laboratorium

Pierwszym uruchomionym programem był czajnik z poprzednich zajęć oświetlony światłem białym. To ćwiczenie umożliwiło zapoznanie się z możliwościami bibliotek graficznych używanych w takich celach, jak również z funkcjami potrzebnymi do uzyskania zadowalających efektów. Drugim zadaniem było oświetlenie jajka wykonanego na poprzednich zajęciach. Efekt nie był zadowalający i dlatego należało samodzielnie stworzyć jedno źródła światła dla jajka, które będzie wyglądać bardziej realistycznie. Gdy to się udało, należało dodać drugie źródło światła i umożliwić zmiane rozmieszczenia źródeł światła przy pomocy przycisków i kursora myszy.

3 Realizacja zadania

3.1 Jedno źródło światła

Pierwszym zadaniem było oświetlenie jajka jednym źródłem światła. Światło miało barwę żółtą, natomiast jajko było białe. Oświetlenie jajka bardziej realistycznie wymagało zapoznania się z definicją wektorów normalnych. Do funkcji tworzącej jajko trzeba było dodać wektory normalne (według instrukcji na stronie zsk).

```
float ux, uy, uz, vx, vy, vz, length;

// obliczenie wartosci wektorow normalnych wedlug wzorow z instrukcji
ux = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)
*cos(M_PI * v);
uy = (640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u);
uz = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)
*sin(M_PI * v);

vx = M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180
* pow(u, 2) + 45 * u) * sin(M_PI * v);
vy = 0;
```

```
vz = -M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180
* pow(u, 2) + 45 * u) * cos(M_PI * v);

vectorNorm[i][j][0] = uy * vz - uz * vy;
// opisanie wektorow normalnych wedlug instrukcji
vectorNorm[i][j][1] = uz * vx - ux * vz;
vectorNorm[i][j][2] = ux * vy - uy * vx;

length = sqrt(vectorNorm[i][j][0] * vectorNorm[i][j][0]
+ vectorNorm[i][j][1] * vectorNorm[i][j][1]
+ vectorNorm[i][j][2] * vectorNorm[i][j][2]);
// obliczenie dlugosci wektorow - konieczne do normalizacji wektorow

if (i < N / 2)  //jesli jestesmy w pierwszej polowie jajka
{
vectorNorm[i][j][0] = (uy * vz - uz * vy) / length;
vectorNorm[i][j][1] = (uz * vx - ux * vz) / length;
vectorNorm[i][j][2] = (ux * vy - uy * vx) / length;
}
else if (i > N / 2) //jesli jestesmy w drugiej polowie jajka
{
vectorNorm[i][j][0] = -1 * (uy * vz - uz * vy) / length;
vectorNorm[i][j][1] = -1 * (uz * vx - ux * vz) / length;
vectorNorm[i][j][2] = -1 * (ux * vy - uy * vx) / length;
}
else if (i == 0 || i == N)
{
vectorNorm[i][j][0] = 0;
vectorNorm[i][j][1] = -1;
vectorNorm[i][j][2] = 0;
}
else
{
vectorNorm[i][j][0] = 0;
vectorNorm[i][j][1] = 1;
vectorNorm[i][j][2] = 0;
}
```

3.1.1 Funkcja Mouse(int btn, int state, int x, int y)

Funkcja Mouse(int btn, int state, int x, int y) obsługuje zdarzenia myszy, reagując na naciśnięcia i zwolnienia przycisków myszy.

Parametry:
-btn: Przechowuje informację o przycisku myszy, który został naciśnięty lub zwolniony (np. GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON).
-state: Przechowuje informację o stanie przycisku myszy - czy został naciśnięty (GLUT_DOWN) czy zwolniony (GLUT_UP).
-x i y: Przechowują aktualne położenie kursora myszy w oknie.

Zadania funkcji:
-Naciśnięcie lewego przycisku myszy (GLUT_LEFT_BUTTON): Zapisuje aktualne położenie kursora myszy (x_pos_old, y_pos_old). Ustawia stan przycisku myszy na 1 (status = 1).
-Naciśnięcie prawego przycisku myszy (GLUT_RIGHT_BUTTON): Zapisuje aktualne położenie kursora myszy (z_pos_old). Ustawia stan przycisku myszy na 2 (status = 2).
-Inne przypadki (np. zwolnienie przycisku): Ustawia stan przycisku myszy na 0 (status = 0).

```
void Mouse(int btn, int state, int x, int y)
{
if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
// przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
x_pos_old = x;
y_pos_old = y;
status = 1;           // wcisniety zostal lewy klawisz myszy
}

else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
{
z_pos_old = y;
status = 2;          // wciśnięty został prawy klawisz myszy
}

else
status = 0;          // nie został wciśnięty żaden klawisz
```

```
}
```

3.1.2 Motion(GLsizei x, GLsizei y)

Motion(GLsizei x, GLsizei y) monitoruje położenie kursora myszy i dostosowuje wartości zmiennych globalnych związanych z ruchem myszy.
Zadania funkcji:
-Obliczenie różnicy położenia kursora myszy.
-Jeżeli lewy przycisk myszy jest naciśnięty (status == 1), to kąty obrotu theta i fi są modyfikowane na podstawie różnic pozycji kursora.
-Jeżeli prawy przycisk myszy jest naciśnięty (status == 2), to odległość R (promień) jest modyfikowana na podstawie różnicy pozycji kursora.
-Zaktualizowanie poprzednich pozycji kursora.
-Przerysowanie obrazu sceny.

```
void Motion(GLsizei x, GLsizei y)
{
delta_x = x - x_pos_old; // Obliczenie roznicy polozenia kursora myszy
delta_y = y - y_pos_old; // Obliczenie roznicy polozenia kursora myszy

if (status == 1) //Jesli lewy klawisz wciśnięty
{
theta += delta_x * pix2angle_x; //modyfikacja kata obrotu
fi += delta_y * pix2angle_y; //do roznicy poozenia kursora myszy
if (theta >= 360.0) //Jesli kat >= 360 stopni
theta = 0.0; //to kat = 0
if (fi >= 360.0)
fi = 0.0;
}
else if (status == 2) //Jesli lewy klawisz wcisniety
{
R += 0.01 * delta_y; //Przyblizanie sie obserwatora do obiektu
if (R <= 8.0) //ustalone maksymalne przyblizenia i oddalenia
R = 8.0; //aby nie wejsc w srodek jajka
if (R >= 13.0)
R = 13.0;
}
x_pos_old = x; //Ustawienie aktualnego polozenia jako poprzednie
y_pos_old = y;
z_pos_old = y;
```

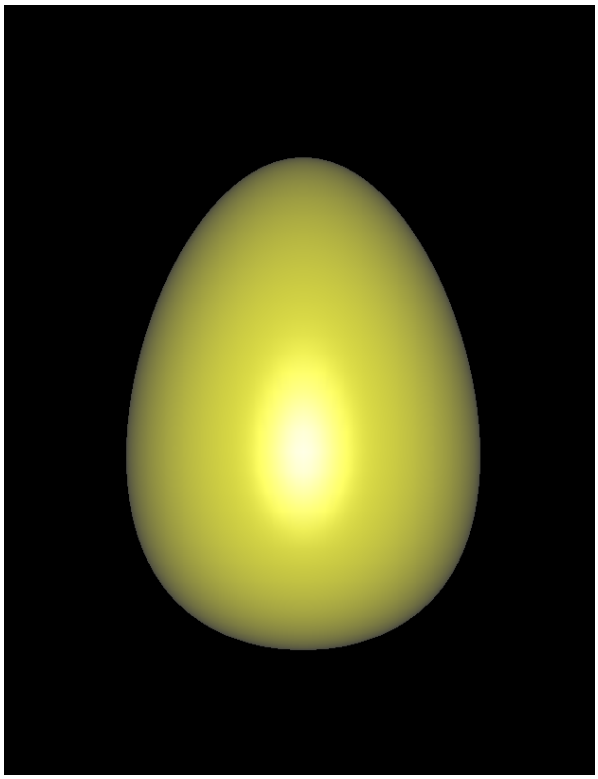
```
glutPostRedisplay(); // Przerysowanie obrazu sceny
}
```

3.1.3 RenderScene()

Funkcja RenderScene jest odpowiedzialna za renderowanie sceny, czyli rysowanie obiektów, osi, oraz zarządzanie ustawieniami kamery.
Zadania funkcji:
-Czyszczenie bufora kolorów i bufora głębokości: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
-Ustawienie macierzy na jednostkową glLoadIdentity().
-Określanie pozycji obserwatora (viewer) w przestrzeni trójwymiarowej na podstawie kątów theta i fi oraz promienia R.
-Zdefiniowanie położenia obserwatora: gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, cos(fi), 0.0);
-Rysowanie osi współrzędnych: Wywołuje funkcję Axes(), która rysuje trzy osie układu współrzędnych.
-Obrót obiektu zgodnie z wartościami kątów obrotu: glRotatef(theta, 0.0, 1.0, 0.0); glRotatef(theta2, 1.0, 0.0, 0.0);.
-Wywoływanie funkcji odpowiedzialnej za rysowanie jajka Egg(). Jajko jest generowane w oparciu o parametry parametryczne i korzysta z wektorów normalnych dla oświetlenia.
-Renderowanie (glFlush() i glutSwapBuffers()): glFlush() przesyła polecenia do wykonania. glutSwapBuffers() zamienia przedni bufor z tylnym, co pozwala na płynne przejścia między klatkami.

```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Czyszczenie okna aktualnym kolorem czyszczacym
glLoadIdentity();
// Czyszczenie macierzy biezacej
Axes();
// Narysowanie osi przy pomocy funkcji zdefiniowanej wyzej

viewer[0] = R * cos(theta) * cos(fi);
viewer[1] = R * sin(fi);
viewer[2] = R * sin(theta) * cos(fi);

gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, cos(fi), 0.0);
```

```
glRotatef(theta, 0.0, 1.0, 0.0);  //obrot obiektu o nowy kat
glRotatef(theta2, 1.0, 0.0, 0.0);

glRotatef(angle[0], 1.0, 0.0, 0.0);
glRotatef(angle[1], 0.0, 1.0, 0.0);
glRotatef(angle[2], 0.0, 0.0, 1.0);

Egg();

glFlush();
// Przekazanie polecen rysujacych do wykonania

glutSwapBuffers();
// Zamienienie bufora przedniego na tylny

}
```



Rysunek 1: Jajko z jednym źródłem światła

3.2 Dwa źródła światła

Kolejnym zadaniem do wykonania było stworzenie drugiego źródła światła i umożliwienie poruszania poszczególnymi źródłami za pomocą przycisków i kursora myszy.

3.2.1 Funkcja Mouse(int btn, int state, int x, int y)

Funkcja Mouse(int btn, int state, int x, int y) obsługuje zdarzenia myszy, reagując na naciśnięcia i zwolnienia przycisków myszy.

Parametry:
-btn: Przechowuje informację o przycisku myszy, który został naciśnięty lub zwolniony (np. GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON).
-state: Przechowuje informację o stanie przycisku myszy - czy został naciśnięty (GLUT_DOWN) czy zwolniony (GLUT_UP).
-x i y: Przechowują aktualne położenie kursora myszy w oknie.

Zadania funkcji:
-Naciśnięcie lewego przycisku myszy (GLUT_LEFT_BUTTON): Zapisuje aktualne położenie kursora myszy (x_pos_old, y_pos_old). Ustawia zmienną status zgodnie z naciśniętym przyciskiem myszy.
-Inne przypadki (np. zwolnienie przycisku): Ustawia stan przycisku myszy na 0 (status = 0).

```
void Mouse(int btn, int state, int x, int y)
{
if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
// przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
x_pos_old = x;
y_pos_old = y;
status = 1;   // wcisniety zostal lewy klawisz myszy
}
else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
{
x_pos_old = x;
y_pos_old = y;
status = 2;
}
else
{
```

```
status = 0;          // nie zostal wcisniety zaden klawisz
}
}
```

3.2.2 Funkcja Motion(GLsizei x, GLsizei y)

Funkcja Motion(GLsizei x, GLsizei y) monitoruje ruch myszy i aktualizuje wartości zmiennych globalnych, co umożliwia interakcję z programem w zależności od ruchu myszy.

Parametry:
-x i y: Aktualne położenie kursora myszy w oknie.

Zadania funkcji:
-Obliczanie różnicy pomiędzy aktualną a poprzednią pozycją kursora myszy.
-Zapisywanie aktualnych współrzędnych do x_pos_old i y_pos_old.
-Przerysowanie sceny.

```
void Motion(GLsizei x, GLsizei y)
{
delta_x = x - x_pos_old; // Obliczenie roznicy polozenia kursora myszy
delta_y = y - y_pos_old; // Obliczenie roznicy polozenia kursora myszy
x_pos_old = x;
y_pos_old = y;

glutPostRedisplay(); // Przerysowanie obrazu sceny
}
```

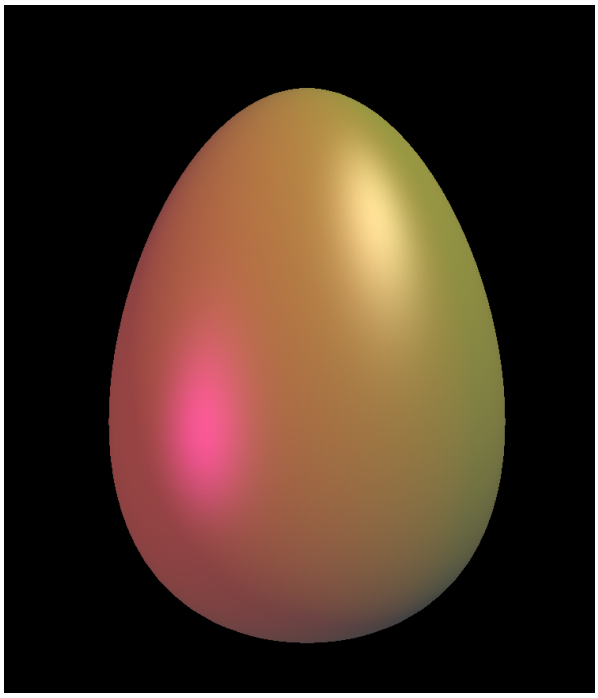
3.2.3 RenderScene()

Funkcja RenderScene jest odpowiedzialna za renderowanie sceny, czyli rysowanie obiektów, osi, oraz zarządzanie ustawieniami kamery.
Zadania funkcji:
-Czyszczenie bufora kolorów i bufora głębokości: glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
-Ładowanie macierzy jednostkowej, czyli zerowanie wszelkich transformacji wcześniej zdefiniowanych glLoadIdentity().
-Rysowanie osi współrzędnych: Wywołuje funkcję Axes(), która rysuje trzy osie układu współrzędnych.
-Ustawienie kamery za pomocą funkcji gluLookAt. Określa położenie kamery, punkt na

który patrzy oraz wektor wskazujący kierunek "góra"kamery gluLookAt(0, 0, viewer[2], 0.0, 0.0, 1.0, 0.0, 1.0, 0.0).
-Sprawdzenie, czy lewy przycisk myszy jest naciśnięty (status == 1). Jeśli tak, to aktualizacja wartości kątów obrotu (theta i fi) na podstawie ruchu myszy.
-Sprawdzenie, czy prawy przycisk myszy jest naciśnięty (status == 2). Jeśli tak, to aktualizacja wartości kątów obrotu (theta2 i fi2) na podstawie ruchu myszy.
-Ograniczenia dziedziny kątów fi i theta oraz fi2 i theta2 do zakresu [0, 2 * pi].
-Ustawienie położenia źródeł światła wykorzystując wartości theta, fi, theta2, fi2 do ustawienia położenia dwóch źródeł światła.
-Rysowanie obiektu 3D: Wywołanie funkcji Egg(), która rysuje jajko na scenie. Jajko jest generowane w oparciu o parametry parametryczne i korzysta z wektorów normalnych dla oświetlenia.
-Renderowanie (glFlush() i glutSwapBuffers()): glFlush() przesyła polecenia do wykonania. glutSwapBuffers() zamienia przedni bufor z tylnym, co pozwala na płynne przejścia między klatkami.

```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Czyszczenie okna aktualnym kolorem czyszczacym
glLoadIdentity();
// Czyszczenie macierzy biezacej
Axes();
// Narysowanie osi przy pomocy funkcji zdefiniowanej wyzej
gluLookAt(0, 0, viewer[2], 0.0, 0.0, 1.0, 0.0, 1.0, 0.0);

if (status == 1) {
theta += delta_x * pix2angle;     // modyfikacja kąta obrotu o kat proporcjonalny
fi += delta_y * pix2angle;
}

if (status == 2) {
theta2 += delta_x * pix2angle;    // modyfikacja kąta obrotu o kat proporcjonalny
fi2 += delta_y * pix2angle;
}

float pi = 3.14;
if (fi > 2 * pi) fi = 2 * pi;   // Dziedzina dla pi i fi
if (theta > 2 * pi) theta = 2 * pi;
if (fi < 0) fi = 0;
```

```
if (theta < 0) theta = 0;

if (fi2 > 2 * pi) fi2 = 2 * pi;
if (theta2 > 2 * pi) theta2 = 2 * pi;
if (fi2 < 0) fi2 = 0;
if (theta2 < 0) theta2 = 0;

//Ustalenie położenia źródła swiatła pierwszego według instrukcji
light_position[0] = zoom * cos(theta2) * cos(fi2);
light_position[1] = zoom * sin(fi2);
light_position[2] = zoom * sin(theta2) * cos(fi2);

//Ustalenie położenia źródła swiatła drugiego według instrukcji
light_positionSecond[0] = zoom * cos(theta) * cos(fi);
light_positionSecond[1] = zoom * sin(fi);
light_positionSecond[2] = zoom * sin(theta) * cos(fi);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT1, GL_POSITION, light_positionSecond);

Egg();

glFlush();
// Przekazanie polecen rysujacych do wykonania

glutSwapBuffers();
// Zamienienie bufora przedniego na tylny

}
```



Rysunek 2: Jajko z dwoma źródłami światła

4 Wnioski

Zadanie pozwoliło na poznanie i zastosowanie metod oświetlania scen 3D. Korzystanie z bibliotek OpenGL i GLUT okazało się skutecznym narzędziem do implementacji źródeł światła. Zaimplementowanie dwóch różnych źródeł pozwoliło zobaczyć, jak dokładnie przebiega stworzenie oświetlenia i pomogło zrozumieć lepiej, jak różne parametry wpływają na światło jak i obiekt. Oświetlanie obiektów ma swoje zastosowanie w różnych kontekstach wizualnych. Pozwala to na lepsze przyjrzenie się strukturze modelu, jak i przede wszystkim zwiększyć realizm obiektów. W rezultacie, zdobyta wiedza i umiejętności są cennym dodatkiem do mojej wiedzy programistycznej. Eksperymentowanie z różnymi aspektami grafiki komputerowej otwiera drzwi do dalszego zgłębiania tego fascynującego obszaru informatyki.