POLITECHNIKA WROCŁAWSKA

Katedra Informatyki Technicznej

Zakład Systemów Komputerowych i Dyskretnych

Grafika komputerowa i komunikacja człowiek – komputer

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 5

„OpenGL – oświetlanie scen 3-D”

Wykonał:	**Adrian Frydmański**
Termin:	**WT/P 12:00-15:00**
Data wykonania ćwiczenia:	**24 XI 2015**
Data oddania sprawozdania:	**8 XII 2015**
Ocena:	

Uwagi prowadzącego:

WSTĘP TEORETYCZNY

Celem ćwiczenia było pokazanie, jak przy pomocy OpenGL można oświetlić obiekt trójwymiarowy.

KOD ŹRÓDŁOWY

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <time.h>
#include <iostream>
using namespace std;

typedef float point3[3];

/*************************************************************************************/
// Stałe i zmienne globalne:
const float pi = 3.14159265;                                    // pi
point3 **pTab;                                                  // tablica na punkty
point3 **nVecTab;                                                       // tablica na kolory
int n = 15;                                                     // do posziału kwadratu jednostkowego
int model = 1;                                                  // 1-punkty, 2-siatka, 3-kolorowe
trojkaty
float squareLen = 1.0;                                                  // długość boku kwadratu
jednostkowego
point3 v = { 0.05, 0.05, 0.05 };                                        // szybkość obracania się
static GLfloat theta[] = { 0.0, 0.0, 0.0 };

/*************************************************************************************/
// Funkcje wyliczające współrzędne punktu (u,v) w przestrzeni 3D
float transformTo3D_x(float u, float v)
{
        float x, a = v * pi;
        x = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
cos(a);
        return x;
}

float transformTo3D_y(float u, float v)
{
        float y;
        y = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2);
        return y - 5;                                           // obniżenie jajka, żeby się ładnie
mieściło
}

float transformTo3D_z(float u, float v)
{
        float z, a = v * pi;
        z = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
sin(a);
        return z;
}

float x_u(float u, float v)
{
        return (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * cos(pi * v);
}

float x_v(float u, float v)
{
        return pi * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 * u)
* sin(pi * v);
}
```

```
float y_u(float u, float v)
{
	return 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
}

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

float z_u(float u, float v)
{
	return (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * sin(pi * v);
}

float z_v(float u, float v)
{
	return -pi * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 *
u) * cos(pi * v);
}

float normalVectorOfEgg_x(float u, float v)
{
	return y_u(u, v) * z_v(u, v) - z_u(u, v) * y_v(u, v);
}

float normalVectorOfEgg_y(float u, float v)
{
	return z_u(u, v) * x_v(u, v) - x_u(u, v) * z_v(u, v);
}

float normalVectorOfEgg_z(float u, float v)
{
	return x_u(u, v) * y_v(u, v) - y_u(u, v) * x_v(u, v);
}

/*****************************************************************************/
// Generowanie siatki punktów:
void eggGenerate()
{
	float step = squareLen / n;
	// współrzędne 2D -> 3D
	float u, v, length;
	for (int i = 0; i<n + 1; i++)
		for (int j = 0; j<n + 1; j++)
		{
			u = j * step;
			v = i * step;
			pTab[i][j][0] = transformTo3D_x(u, v);
			pTab[i][j][1] = transformTo3D_y(u, v);
			pTab[i][j][2] = transformTo3D_z(u, v);
			nVecTab[i][j][0] = normalVectorOfEgg_x(u, v);
			nVecTab[i][j][1] = normalVectorOfEgg_y(u, v);
			nVecTab[i][j][2] = normalVectorOfEgg_z(u, v);
			length = sqrt(nVecTab[i][j][0] * nVecTab[i][j][0] + nVecTab[i][j][1] *
nVecTab[i][j][1] + nVecTab[i][j][2] * nVecTab[i][j][2]);
			for (int k = 0; k < 3; k++)
			{
				if (j < n / 2)
					nVecTab[i][j][k] *= -1;
				nVecTab[i][j][k] /= length;
			}
			if (length == 0)
				if (i ==0 || i == n+1)
				{
```

```
                                nVecTab[i][j][0] = 0;
                                nVecTab[i][j][1] = -1;
                                nVecTab[i][j][2] = 0;
                        }
                        else if (i == n / 2)
                        {
                                nVecTab[i][j][0] = 0;
                                nVecTab[i][j][1] = 1;
                                nVecTab[i][j][2] = 0;
                        }
                }
}

/*************************************************************************************/
// Renderowanie jajka
void Egg()
{
        // Generowanie siatki
        eggGenerate();

        // Ustawienie koloru białego
        glColor3f(1.0, 1.0, 1.0);

        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                {
                        // w jedną stronę
                        glBegin(GL_TRIANGLES);
                        glNormal3fv(nVecTab[i][j + 1]);
                        glVertex3fv(pTab[i][j + 1]);
                        glNormal3fv(nVecTab[i + 1][j]);
                        glVertex3fv(pTab[i + 1][j]);
                        glNormal3fv(nVecTab[i + 1][j + 1]);
                        glVertex3fv(pTab[i + 1][j + 1]);
                        // w drugą stronę
                        glNormal3fv(nVecTab[i][j]);
                        glVertex3fv(pTab[i][j]);
                        glNormal3fv(nVecTab[i + 1][j]);
                        glVertex3fv(pTab[i + 1][j]);
                        glNormal3fv(nVecTab[i][j + 1]);
                        glVertex3fv(pTab[i][j + 1]);
                        glEnd();
                }
}

void allocate()
{
        //Dynamiczna alokacja tablicy punktow
        pTab = new point3*[n + 1];
        for (int i = 0; i < n + 1; i++)
                pTab[i] = new point3[n + 1];

        //Dynamiczna alokacja tablicy i wygenerowanie kolorow losowych dla punktow
        nVecTab = new point3*[n + 1];
        for (int i = 0; i < n + 1; i++)
                nVecTab[i] = new point3[n + 1];
}

void relase()
{
        //Zwolnienie pamięci
        for (int i = 0; i < n + 1; i++)
        {
                delete[] pTab[i];
                delete[] nVecTab[i];
                pTab[i] = 0;
```

```
			nVecTab[i] = 0;
		}
		delete[] pTab;
		delete[] nVecTab;
		pTab = 0;
		nVecTab = 0;
}

/*************************************************************************************/
// Funkcja określająca co ma być rysowane (zawsze wywoływana, gdy trzeba przerysować scenę)
void RenderScene()
{
		// Czyszczenie okna aktualnym kolorem czyszczącym
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		// Czyszczenie macierzy bieżącej
		glLoadIdentity();

		//Rotacje
		glRotatef(theta[0], 1.0, 0.0, 0.0);
		glRotatef(theta[1], 0.0, 1.0, 0.0);
		glRotatef(theta[2], 0.0, 0.0, 1.0);


		//Renderowanie jajka
		Egg();
		//glutSolidTeapot(2.5);

		//czajnik
		//glutSolidTeapot(2);

		// Przekazanie poleceń rysujących do wykonania
		glFlush();

		glutSwapBuffers();
}

/*************************************************************************************/
// Funkcja zwrotna dla klawiszy
void Keys(unsigned char key, int x, int y)
{
		// zmiana "rozdzielczości" jajka
		if (key == '+' || key == '-' || key == 72 || key == 80)
		{
			relase();
			if (key == '+' || key == 72) n += 5;
			if ((key == '-' || key == 80) && n > 0) n -= 5;
			allocate();
		}
		// zmiana rędkości kręcenia się
		if (key == '7') v[0] += 0.05;
		if (key == '4') v[0] -= 0.05;
		if (key == '8') v[1] += 0.05;
		if (key == '5') v[1] -= 0.05;
		if (key == '9') v[2] += 0.05;
		if (key == '6') v[2] -= 0.05;
		RenderScene();
}

/*************************************************************************************/
// Funkcja zwrotna dla obrotu
void spinEgg()
{
		theta[0] -= v[0];
		if (theta[0] > 360.0) theta[0] -= 360.0;
		theta[1] -= v[1];
```

```
	if (theta[1] > 360.0) theta[1] -= 360.0;
	theta[2] -= v[2];
	if (theta[2] > 360.0) theta[2] -= 360.0;

	glutPostRedisplay();		//odświeżenie zawartości okna
}

/*************************************************************************/
// Funkcja ustalająca stan renderowania
void MyInit(void)
{
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);	// Kolor czyszczący (wypełnienia okna) ustawiono
na czarny

	/*************************************************************************/
	// Definicja materiału z jakiego zrobiony jest czajnik
	GLfloat mat_ambient[] = { 1.0, 1.0, 0.0, 1.0 };	// współczynniki ka =[kar,kag,kab] dla
światła otoczenia
	GLfloat mat_diffuse[] = { 1.0, 1.0, 0.0, 1.0 };	// współczynniki kd =[kdr,kdg,kdb]
światła rozproszonego
	GLfloat mat_specular[] = { 1.0, 1.0, 0.0, 1.0 };	// współczynniki ks =[ksr,ksg,ksb] dla
światła odbitego
	GLfloat mat_shininess = { 20.0 };		// współczynnik n opisujący połysk
powierzchni

	/*************************************************************************/
	// Definicja źródła światła
	GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };	// położenie źródła
	GLfloat light_ambient[] = { 0.1, 0.1, 0.1, 1.0 }; // składowe intensywności świecenia
źródła światła otoczenia
							// Ia = [Iar,Iag,Iab]
	GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 }; // składowe intensywności świecenia
źródła światła powodującego

		// odbicie dyfuzyjne Id = [Idr,Idg,Idb]
	GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };// składowe intensywności świecenia
źródła światła powodującego

		// odbicie kierunkowe Is = [Isr,Isg,Isb]
	GLfloat att_constant = { 1.0 };			// składowa stała ds dla modelu zmian
oświetlenia w funkcji
							// odległości od źródła
	GLfloat att_linear = { 0.05f };			// składowa liniowa dl dla modelu zmian
oświetlenia w funkcji

		// odległości od źródła
	GLfloat att_quadratic = { 0.001f };		// składowa kwadratowa dq dla modelu
zmian oświetlenia w funkcji
							// odległości od źródła

	/*************************************************************************/
	// Ustawienie patrametrów 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);

	/*************************************************************************/
	// Ustawienie parametrów źródła
	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_position);
	glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant);
	glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, att_linear);
	glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, att_quadratic);
```

```
	/*************************************************************************************/
	// Ustawienie opcji systemu oświetlania sceny
	glShadeModel(GL_SMOOTH); // właczenie łagodnego cieniowania
	glEnable(GL_LIGHTING);   // właczenie systemu oświetlenia sceny
	glEnable(GL_LIGHT0);     // włączenie źródła o numerze 0
	glEnable(GL_DEPTH_TEST); // włączenie mechanizmu z-bufora
}

/*************************************************************************************/
// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych obiektów w przypadku zmiany
rozmiarów okna.
// Parametry vertical i horizontal są przekazywane do funkcji za każdym razem gdy zmieni się
rozmiar okna.
void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
	// Deklaracja zmiennej AspectRatio  określającej proporcję wymiarów okna
	GLfloat AspectRatio;

	if (vertical == 0)                                // Zabezpieczenie przed dzieleniem przez
0
		vertical = 1;

	// Ustawienie wielkościokna okna widoku (viewport)
	// W tym przypadku od (0,0) do (horizontal, vertical)
	glViewport(0, 0, horizontal, vertical);

	// Przełączenie macierzy bieżącej na macierz projekcji
	glMatrixMode(GL_PROJECTION);

	// Czyszcznie macierzy bieżącej
	glLoadIdentity();

	// Wyznaczenie współczynnika  proporcji okna
	// Gdy okno nie jest kwadratem wymagane jest określenie tak zwanej
	// przestrzeni ograniczającej pozwalającej zachować właściwe
	// proporcje rysowanego obiektu.
	// Do okreslenia przestrzeni ograniczjącej służy funkcja
	// glOrtho(...)
	AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;

	if (horizontal <= vertical)
		glOrtho(-7.5, 7.5, -7.5 / AspectRatio, 7.5 / AspectRatio, 10.0, -10.0);
	else
		glOrtho(-7.5*AspectRatio, 7.5*AspectRatio, -7.5, 7.5, 10.0, -10.0);
	// Przełączenie macierzy bieżącej na macierz widoku modelu
	glMatrixMode(GL_MODELVIEW);
	// Czyszcenie macierzy bieżącej
	glLoadIdentity();
}

/*************************************************************************************/
// Główny punkt wejścia programu. Program działa w trybie konsoli
void main(void)
{
	//włączenie polskich znaków
	setlocale(LC_ALL, "");
	// zaalokuj tablicę
	allocate();
	// inicjowanie randa
	srand((unsigned)time(NULL));
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(800, 800);
	glutCreateWindow("Modelowanie obiektow 3D - jajko (instrukcja w konsoli)");
	// Określenie funkcji RenderScene jako funkcji zwrotnej (callback function)
	// Będzie ona wywoływana za każdym razem, gdy zajdzie potrzba przeryswania okna
```

```
        glutDisplayFunc(RenderScene);
        // Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną za zmiany rozmiaru okna
        glutReshapeFunc(ChangeSize);
        // Funkcja MyInit() wykonuje wszelkie inicjalizacje konieczne przed przystąpieniem do
renderowania
        MyInit();
        // Włączenie mechanizmu usuwania powierzchni niewidocznych
        glEnable(GL_DEPTH_TEST);
        // Funkcja zwrotna dla klawiatury
        glutKeyboardFunc(Keys);
        // Funkcja zwrotna obrotu
        glutIdleFunc(spinEgg);
        // główna petla GLUTa
        glutMainLoop();
        // zwolnij pamięć
        relase();
}
```

OPIS DZIAŁANIA

Aby uzyskać odpowiedni efekt do siatki punktów należy dodać siatkę wektorów normalnych dla każdego z punktów. Wektory zostały obliczone z następującego wzoru:

$$N(u,v) = \left[\begin{vmatrix} y_u & z_u \\ y_v & z_v \end{vmatrix}, \begin{vmatrix} z_u & x_u \\ z_v & x_v \end{vmatrix}, \begin{vmatrix} x_u & y_u \\ x_v & y_v \end{vmatrix} \right] =$$

$$= \left[y_u \cdot z_v - z_u y_v \;,\quad z_u \cdot x_v - x_u z_v \;,\quad x_u \cdot y_v - y_u x_v \right] \neq 0$$

gdzie

$$x_u = \frac{\partial x(u,v)}{\partial u}, \quad x_v = \frac{\partial x(u,v)}{\partial v}$$

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

$$z_u = \frac{\partial z(u,v)}{\partial u}, \quad z_v = \frac{\partial z(u,v)}{\partial v}$$

Po różniczkowaniu otrzymane wzory widoczne są poniżej i należy je wstawić do pierwszego wzoru na wektor normalny.

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

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

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

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

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

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

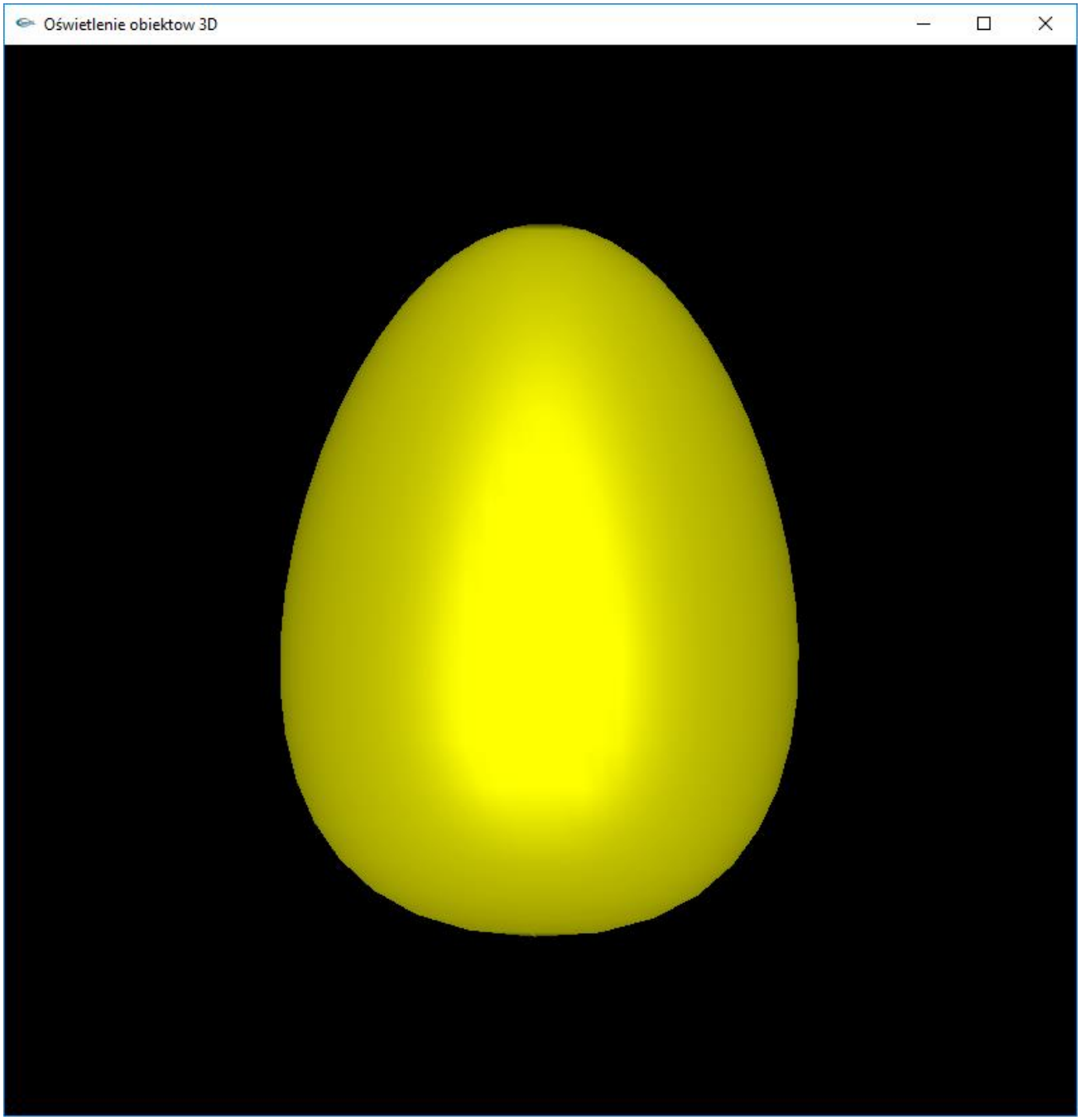
Po samym podstawieniu policzony wektor jest zdenormalizowany, dlatego zachodzi potrzeba normalizacji przez podzielenie składowych przez długość wektora. Dodatkowo dla połowy jajka konieczne jest odwrócenie wektorów, które odbywa się dla każdego punktu kwadratu jednostkowego, gdyż bez tego oświetlona zostaje wewnętrzna część połówki.

```
for (int k = 0; k < 3; k++)
{
	if (j < n / 2)
		nVecTab[i][j][k] *= -1;
	nVecTab[i][j][k] /= length;
}
```

Wektory normalne w programie przechowuje trójwymiarowa tablica `nVecTab`. Uwzględnianie ich przy rysowaniu odbywa się podczas renderowanie jajka przed podaniem punktu, jak widać w przykładzie:

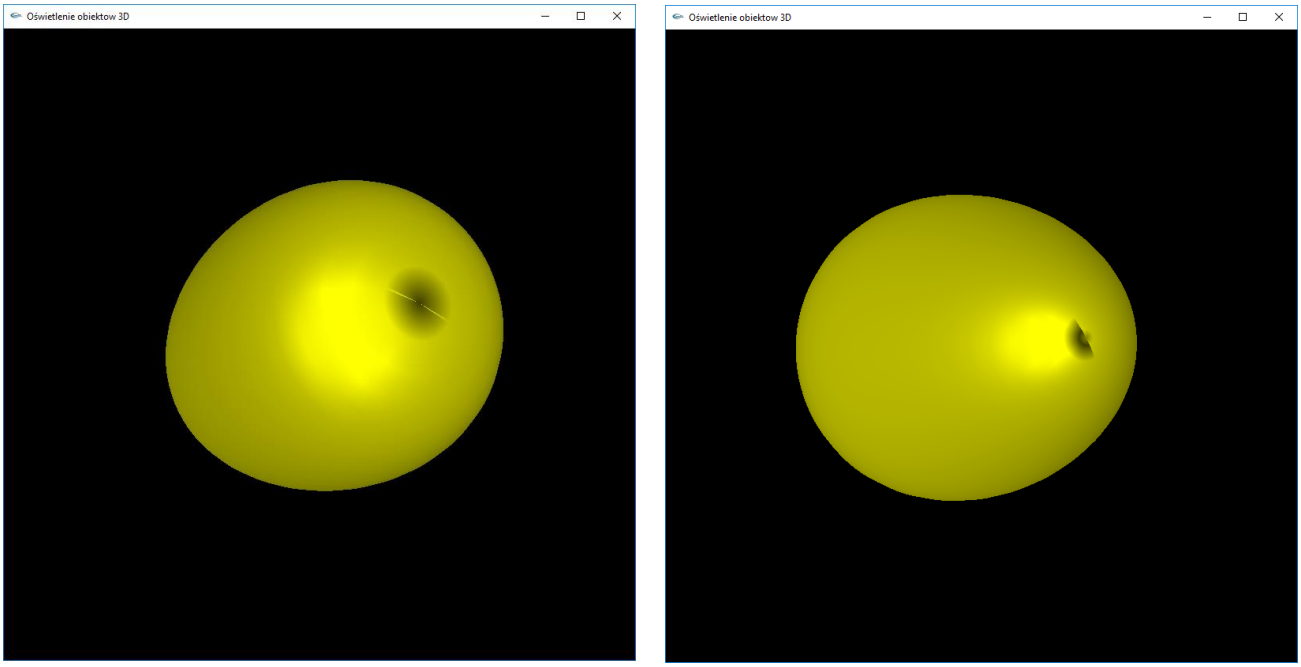
```
glBegin(GL_TRIANGLES);
glNormal3fv(nVecTab[i][j + 1]);
glVertex3fv(pTab[i][j + 1]);
glNormal3fv(nVecTab[i + 1][j]);
glVertex3fv(pTab[i + 1][j]);
glNormal3fv(nVecTab[i + 1][j + 1]);
glVertex3fv(pTab[i + 1][j + 1]);
glEnd();
```

Źródło światła w programie ustawione jest za „kamerą”. Narysowane i oświetlone na żółto jajko prezentuje się następująco:



Rys. 1 Oświetlone jajko

Niestety niektóre wektory normalne zostały niepoprawnie obliczone. Powód nie został dotychczas znaleziony. Na poniższym rysunku widać, jak niektóre punkty górnego i dolnego czubka są oświetlone od środka, miast z zewnątrz.



Rys. 2 Błędy oświetlenia

PODSUMOWANIE

Ćwiczenie pokazało, jak się zachowuje światło w modelowaniu 3D. Ukazane zostały trudności związane z budowaniem poprawnego modelu oświetlenia.