POLITECHNIKA WROCŁAWSKA

Grafika komputerowa i komunikacja człowiek-komputer

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 6

OpenGL - oświetlanie scen 3-D

Wykonał:	**Janusz Pelc 252799**
Termin:	**Np. PN/TP 7:30-10:30**
Data wykonania ćwiczenia:	**15 Grudnia 2021**
Data oddania sprawozdania:	**22 Grudnia 2021**
Ocena:	

Uwagi prowadzącego:

1. Wstęp

Ćwiczenie polegało na nałożeniu tekstury na model 3D. Należało zmodyfikować program z poprzedniego ćwiczenia, tak aby na model nałożona została wybrana tekstura

2. Program

2.1. Zadanie 1

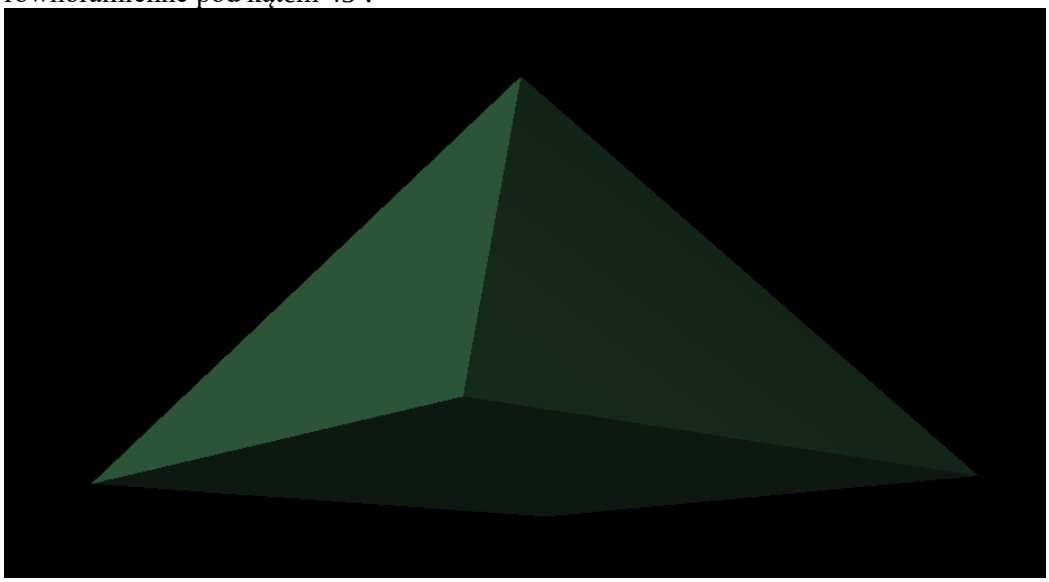
Pierwsze zadanie polegało modyfikacji kodu z poprzedniego zadania poprzez dodanie własnego modelu piramidy

```
void Pyramid() {
        glBegin(GL_QUADS);  //funkcja rysuje wielokąt
        glNormal3f(0, -1.0f, 0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5, -2, -5);

        glNormal3f(0, -1, 0);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-5, -2, 5);

        glNormal3f(0, -1, 0);
        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(5, -2, 5);

        glNormal3f(0, -1, 0);
        glTexCoord2f(0.0f, 1.0f);
        glVertex3f(5, -2, -5);

        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(-1/sqrt(2), 1 / sqrt(2), 0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5, -2, -5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-5, -2, 5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(0, 1 / sqrt(2), 1 / sqrt(2));
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5, -2, 5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(5, -2, 5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(1 / sqrt(2), 1 / sqrt(2), 0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(5, -2, 5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(5, -2, -5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(0, 1 / sqrt(2), -1 / sqrt(2));
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(5, -2, -5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-5, -2, -5);
```

```
        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();
}
```

Funkcje glTexCoord2f() służą przy teksturowaniu, co jest opisane później.

Funkcja ta rysuje kwadratową podstawę, a przy każdej krawędzi rysuje trójkąty równoramienne pod kątem 45º.



Rys1. Piramida

Następnie należało nałożyć na piramidę teksturę. Użyto do tego funkcji z instrukcji

```
GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight, GLint *ImComponents, GLenum
*ImFormat)
{
        /*****************************************************************************/
        // Struktura dla nagłówka pliku  TGA

#pragma pack(1)
        typedef struct
        {
                GLbyte    idlength;
                GLbyte    colormaptype;
                GLbyte    datatypecode;
                unsigned short    colormapstart;
                unsigned short    colormaplength;
                unsigned char     colormapdepth;
                unsigned short    x_orgin;
                unsigned short    y_orgin;
                unsigned short    width;
                unsigned short    height;
                GLbyte    bitsperpixel;
                GLbyte    descriptor;
        }TGAHEADER;
#pragma pack(8)

        FILE *pFile;
        TGAHEADER tgaHeader;
        unsigned long lImageSize;
        short sDepth;
        GLbyte    *pbitsperpixel = NULL;

        /*****************************************************************************/
        // Wartości domyślne zwracane w przypadku błędu
```

```
	*ImWidth = 0;
	*ImHeight = 0;
	*ImFormat = GL_BGR_EXT;
	*ImComponents = GL_RGB8;


	errno_t err = fopen_s(&pFile, FileName, "rb");
	if (pFile == NULL)
		return NULL;

	/*****************************************************************************/
	// Przeczytanie nagłówka pliku

	fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);

	/*****************************************************************************/

	// Odczytanie szerokości, wysokości i głębi obrazu

	*ImWidth = tgaHeader.width;
	*ImHeight = tgaHeader.height;
	sDepth = tgaHeader.bitsperpixel / 8;

	/*****************************************************************************/
	// Sprawdzenie, czy głębia spełnia założone warunki (8, 24, lub 32 bity)

	if(tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHeader.bitsperpixel != 32)
		return NULL;
	/*****************************************************************************/

	// Obliczenie rozmiaru bufora w pamięci
	lImageSize = tgaHeader.width * tgaHeader.height * sDepth;

	/*****************************************************************************/
	// Alokacja pamięci dla danych obrazu

	pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));

	if (pbitsperpixel == NULL)
		return NULL;

	if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
	{
		free(pbitsperpixel);
		return NULL;
	}

	/*****************************************************************************/
	// Ustawienie formatu OpenGL
	switch (sDepth)
	{
	case 3:
		*ImFormat = GL_BGR_EXT;
		*ImComponents = GL_RGB8;
		break;
	case 4:
		*ImFormat = GL_BGRA_EXT;
		*ImComponents = GL_RGBA8;
		break;
	case 1:
		*ImFormat = GL_LUMINANCE;
		*ImComponents = GL_LUMINANCE8;
		break;
	};
	fclose(pFile);
	return pbitsperpixel;
}
```

Do funkcji MyInit() dodano kolejny kod z instrukcji

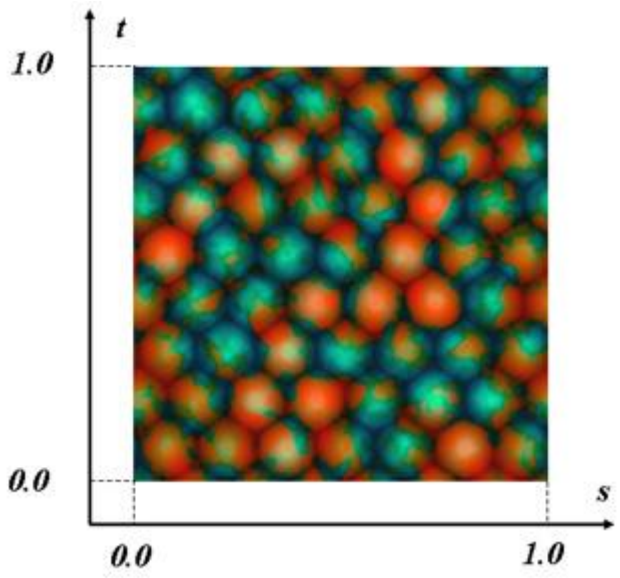
```
void MyInit(void)
{
	GLbyte *pBytes;
	GLint ImWidth, ImHeight, ImComponents;
```

```
        GLenum ImFormat;
        // ..............................
        //       Pozostała część funkcji MyInit()
        // ..............................
/*************************************************************************************/

// Teksturowanie będzie prowadzone tyko po jednej stronie ściany

        glEnable(GLUT_FULLY_COVERED);
        /*************************************************************************************/
        //  Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga

        pBytes = LoadTGAImage("t_256.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);
        /*************************************************************************************/
   // Zdefiniowanie tekstury 2-D

        glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL_UNSIGNED_BYTE,
pBytes);
        /*************************************************************************************/
        // Zwolnienie pamięci

        free(pBytes);

        /*************************************************************************************/
        // Włączenie mechanizmu teksturowania
        glEnable(GL_TEXTURE_2D);

        /*************************************************************************************/
        // Ustalenie trybu teksturowania

        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        /*************************************************************************************/
        // Określenie sposobu nakładania tekstur

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

W funkcji glEnable(GLUT_FULLY_COVERED) ustawiono następujący parametr aby tekstury pokrywały obie strony powierzchni.

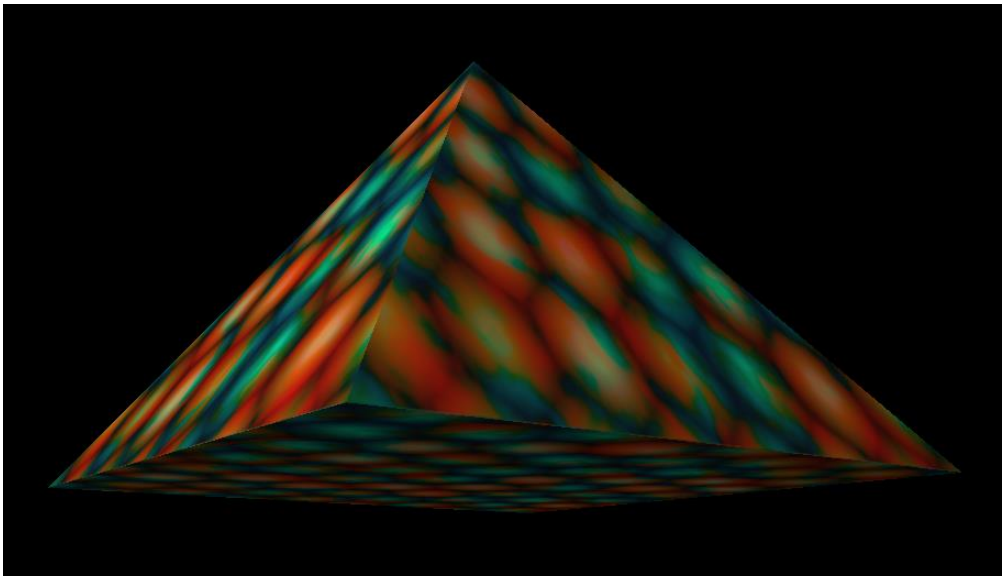
W funkcji LoadTGAImage("t_256.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat) parametr "t_256.tga" oznacza plik tekstury, którą chcemy użyć.

Następnie należało w funkcji Pyramid() dodać do wierzchołków współrzędne na teksturze, co widać we wcześniejszym kodzi.



Rys2. Współrzędne na teksturze

W efekcie otrzymaliśmy



Rys3. Piramida z teksturą

2.2. Zadanie 2

Zadanie polegało na dodaniu tekstury do własnego modelu jajka z poprzednich ćwiczeń.

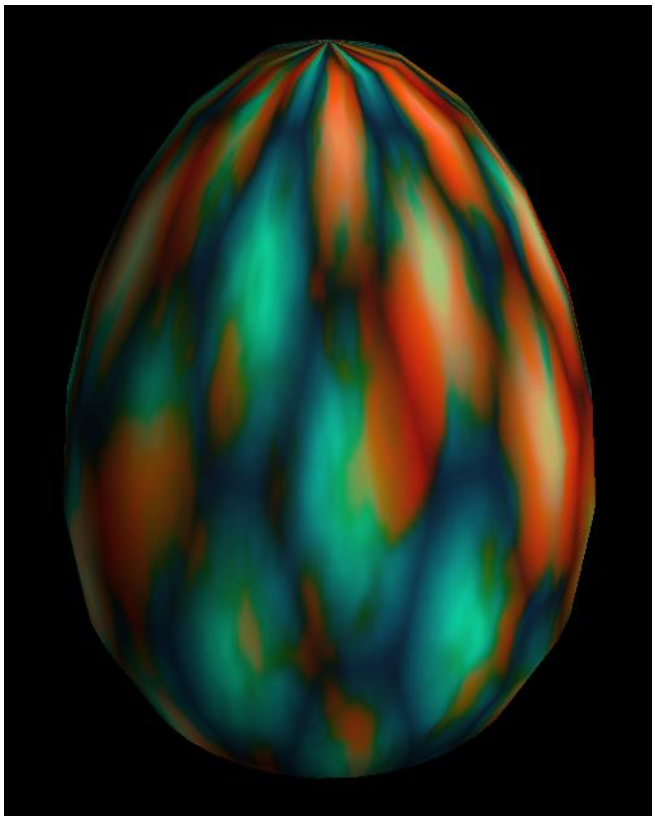
Dane wierzchołków są przechowywane w tablicach NxN, więc aby nałożyć teksturę na jajko, każdemu wierzchołkowi przypisano współrzędna na teksturze, tak że wierzchołkom na wierzchołkach tablicy przypisano współrzędne wierzchołków tekstury, a pozostałym równo rozmieszczone współrzędne pomiędzy. W skrócie na kwadratową siatkę (tablicę) wierzchołków jajka nałożono teksturę od krawędzi do krawędzi.

Następnie, tak jak w przypadku piramidy, w funkcji Egg(), podczas rysowania trójkątów, do każdego wierzchołka przypisano współrzędną tekstury za pomocą glTexCoord2f(x, y);

```
void Egg(void) {
	for (int i = 0; i < N - 1; i++) {
		for (int k = 0; k < N - 1; k++) {
			//rysowane są 2 trójkąty o wspólnym boku pomiędzy 4 sąsiadującymi w tablicy
wierchołkami
			glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
			glNormal3f(Nx[i][k], Ny[i][k], Nz[i][k]);
			glTexCoord2f(t[i][k][0], t[i][k][1]);
			glVertex3f(x[i][k], y[i][k], z[i][k]);

			glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
			glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
			glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

			glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
			glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
			glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
			glEnd();

			glBegin(GL_POLYGON);
			glNormal3f(Nx[i + 1][k + 1], Ny[i + 1][k + 1], Nz[i + 1][k + 1]);
			glTexCoord2f(t[i + 1][k + 1][0], t[i + 1][k + 1][1]);
			glVertex3f(x[i + 1][k + 1], y[i + 1][k + 1], z[i + 1][k + 1]);

			glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
			glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
			glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

			glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
```

```
                        glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
                        glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
                        glEnd();
                }
        }
}
```

Ostatecznie otrzymano



3. Kod źródłowy

```
/*************************************************************************************
/

//  Szkielet programu do tworzenia modelu sceny 3-D z wizualizacją osi
//  układu współrzędnych dla rzutowania perspektywicznego

/*************************************************************************************
/
#define _USE_MATH_DEFINES

#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <iostream>

using namespace std;
typedef float point3[3];
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };
// inicjalizacja położenia obserwatora
```

```
static GLfloat thetax = 0.0;   // kąt obrotu obiektu
static GLfloat thetay = 0.0;   // kąt obrotu obiektu
static GLfloat thetax1 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay1 = 0.0;   // kąt obrotu obiektu
static GLfloat thetax2 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay2 = 0.0;   // kąt obrotu obiektu
static GLfloat pix2angle;     // przelicznik pikseli na stopnie
static GLfloat cameraz = 0.0;
static GLint status1 = 0;       // stan klawiszy myszy
static GLint status2 = 0;         // 0 - nie naciśnięto żadnego klawisza
                                  // 1 - naciśnięty zostać lewy klawisz
static GLint status3 = 1;         //model obrotu (1 - obrót świałami, 0 - obrót
kamerą)
static int x_pos_old = 0;       // poprzednia pozycja kursora myszy
static int y_pos_old = 0;
static int delta_x = 0;        // różnica pomiędzy pozycją bieżącą
static int delta_y = 0;                 // i poprzednią kursora myszy
static int R1 = 10;
const int N = 21; //rozmiar tablicy wierchołków NxN
static GLfloat theta[] = { 0.0, 0.0, 0.0 }; // trzy kąty obrotu
int model = 1;  // 1- jajko, 2- piramida

//tablice współrzędnych wierzchołków
float x[N][N];
float y[N][N];
float z[N][N];

float Nx[N][N];
float Ny[N][N];
float Nz[N][N];

float t[N][N][2];
GLfloat light_position[] = { 10.0, 0.0, 10.0, 1.0 };
// położenie źródła
GLfloat light_position1[] = { -10.0, 0.0, -10.0, 1.0 };
// położenie źródła
/*************************************************************************
/
/*************************************************************************
/
 // Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie
 // FileName, alokuje pamięć i zwraca wskaźnik (pBits) do bufora w którym
 // umieszczone są dane.
 // Ponadto udostępnia szerokość (ImWidth), wysokość (ImHeight) obrazu
 // tekstury oraz dane opisujące format obrazu według specyfikacji OpenGL
 // (ImComponents) i (ImFormat).
 // Jest to bardzo uproszczona wersja funkcji wczytującej dane z pliku TGA.
 // Działa tylko dla obrazów wykorzystujących 8, 24, or 32 bitowy kolor.
 // Nie obsługuje plików w formacie TGA kodowanych z kompresją RLE.
/*************************************************************************
/


GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight, GLint
*ImComponents, GLenum *ImFormat)
{
       /*************************************************************************
******/
       // Struktura dla nagłówka pliku  TGA
#pragma pack(1)
       typedef struct
       {
```

```
		GLbyte	idlength;
		GLbyte	colormaptype;
		GLbyte	datatypecode;
		unsigned short	colormapstart;
		unsigned short	colormaplength;
		unsigned char	colormapdepth;
		unsigned short	x_orgin;
		unsigned short	y_orgin;
		unsigned short	width;
		unsigned short	height;
		GLbyte	bitsperpixel;
		GLbyte	descriptor;
	}TGAHEADER;
#pragma pack(8)

	FILE *pFile;
	TGAHEADER tgaHeader;
	unsigned long lImageSize;
	short sDepth;
	GLbyte	*pbitsperpixel = NULL;
	/*****************************************************************************
******/
	// Wartości domyślne zwracane w przypadku błędu
	*ImWidth = 0;
	*ImHeight = 0;
	*ImFormat = GL_BGR_EXT;
	*ImComponents = GL_RGB8;
	errno_t err = fopen_s(&pFile, FileName, "rb");
	if (pFile == NULL)
		return NULL;
	/*****************************************************************************
******/
	// Przeczytanie nagłówka pliku
	fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);
	/*****************************************************************************
******/

	// Odczytanie szerokości, wysokości i głębi obrazu

	*ImWidth = tgaHeader.width;
	*ImHeight = tgaHeader.height;
	sDepth = tgaHeader.bitsperpixel / 8;
	/*****************************************************************************
******/
	// Sprawdzenie, czy głębia spełnia założone warunki (8, 24, lub 32 bity)

	if (tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 &&
tgaHeader.bitsperpixel != 32)
		return NULL;
	/*****************************************************************************
******/
	// Obliczenie rozmiaru bufora w pamięci
	lImageSize = tgaHeader.width * tgaHeader.height * sDepth;
	/*****************************************************************************
******/
	// Alokacja pamięci dla danych obrazu
	pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));
	if (pbitsperpixel == NULL)
		return NULL;
	if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
	{
		free(pbitsperpixel);
```

```
		return NULL;
	}
	/*****************************************************************************
******/
	// Ustawienie formatu OpenGL
	switch (sDepth)
	{
	case 3:
		*ImFormat = GL_BGR_EXT;
		*ImComponents = GL_RGB8;
		break;
	case 4:
		*ImFormat = GL_BGRA_EXT;
		*ImComponents = GL_RGBA8;
		break;
	case 1:
		*ImFormat = GL_LUMINANCE;
		*ImComponents = GL_LUMINANCE8;
		break;
	};
	fclose(pFile);
	return pbitsperpixel;
}

/*****************************************************************************
/
/*****************************************************************************
/
// Funkcja "bada" stan myszy i ustawia wartości odpowiednich zmiennych globalnych

void Mouse(int btn, int state, int x, int y) {
	if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
	x_pos_old = x;		// przypisanie aktualnie odczytanej pozycji kursora
	y_pos_old = y;					// jako pozycji poprzedniej
		status1 = 1;		// wcięnięty został lewy klawisz myszy
	}
	else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
	x_pos_old = x;		// przypisanie aktualnie odczytanej pozycji kursora
	y_pos_old = y;					// jako pozycji poprzedniej
	status2 = 1;		// wcięnięty został lewy klawisz myszy
	}
	else	status1 = status2 = 0;		// nie został wcięnięty żaden klawisz
}

/*****************************************************************************
/
// Funkcja "monitoruje" położenie kursora myszy i ustawia wartości odpowiednich
// zmiennych globalnych
void Motion(GLsizei x, GLsizei y) {

	delta_x = x - x_pos_old;	// obliczenie różnicy położenia kursora myszy
	x_pos_old = x;			// podstawienie bieżącego położenia jako poprzednie
	delta_y = y - y_pos_old;	// obliczenie różnicy położenia kursora myszy
	y_pos_old = y;			// podstawienie bieżącego położenia jako poprzednie
	glutPostRedisplay();	// przerysowanie obrazu sceny
}
void initEgg() {
	float u = 0, v = 0; //zminne u i v wykorzystywane w funkcjach określających
wierzchołki jajka;
	float fN = N; //wartość stałej N jako float
	float l = 0;
```

```
	float xu;
	float yu;
	float zu;
	float xv;
	float yv;
	float zv;

	for (int i = 0; i < N; i++) {
		u = float(i / (fN-1)); //zmienna zmniejszana proporcjonalnie do stałej N,
aby znajadowała się w zakresie użytej funkcji [0,1]
		for (int k = 0; k < N; k++) {
			v = float(k / (fN-1)); //zmienna jest zmniejszana proporcjonalnie
do stałej N, aby znajadowała się w zakresie użytej funkcji [0,1]
			//na podstawie użytej funkcji wyzaczane są wierzchołki
			x[i][k] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) +
180 * pow(u, 2) - 45 * u) * cos(M_PI * v);
			y[i][k] = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2) - 5;
			z[i][k] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) +
180 * pow(u, 2) - 45 * u) * sin(M_PI * v);

			xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 *
u - 45) * cos(M_PI * v);
			yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
			zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 *
u - 45) * sin(M_PI * v);

			xv = M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) -
180 * pow(u, 2) + 45 * u) * sin(M_PI * v);
			yv = 0;
			zv = -M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) -
180 * pow(u, 2) + 45 * u) * cos(M_PI * v);

			Nx[i][k] = yu * zv - zu * yv;
			Ny[i][k] = zu * xv - xu * zv;
			Nz[i][k] = xu * yv - yu * xv;
			l = sqrt(Nx[i][k] * Nx[i][k] + Ny[i][k] * Ny[i][k] + Nz[i][k] *
Nz[i][k]);
			Nx[i][k] /= l;
			Ny[i][k] /= l;
			Nz[i][k] /= l;
			if (i > N / 2 && i<N-1) {
				Nx[i][k] *= -1;
				Ny[i][k] *= -1;
				Nz[i][k] *= -1;
			}
			if (x[i][k] == 0 && z[i][k] == 0) {
				Nx[i][k] = 0;
				if(y[i][k]==-5)Ny[i][k] = -1;
				else Ny[i][k] = 1;
				Nz[i][k] = 0;
			}
		}
	}
	float fi, fk;
	int k;
	for (int i = 0; i < N; i++) {
		fi = i+1;
		for (k = 0; k < N; k++) {
			fk = k+1;
			t[i][k][0] = fi / (fN);
			t[i][k][1] = fk / (fN);
		}
```

```
	}
}

//Funkcja rysująca jajko
void Egg(void) {
	for (int i = 0; i < N - 1; i++) {
		for (int k = 0; k < N - 1; k++) {
			//rysowane są 2 trójkąty o wspólnym boku pomiędzy 4 sąsiadującymi
w tablicy wierchołkami
			glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
			glNormal3f(Nx[i][k], Ny[i][k], Nz[i][k]);
			glTexCoord2f(t[i][k][0], t[i][k][1]);
			glVertex3f(x[i][k], y[i][k], z[i][k]);

			glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
			glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
			glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

			glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
			glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
			glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
			glEnd();

			glBegin(GL_POLYGON);
			glNormal3f(Nx[i + 1][k + 1], Ny[i + 1][k + 1], Nz[i + 1][k + 1]);
			glTexCoord2f(t[i + 1][k + 1][0], t[i + 1][k + 1][1]);
			glVertex3f(x[i + 1][k + 1], y[i + 1][k + 1], z[i + 1][k + 1]);

			glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
			glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
			glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

			glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
			glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
			glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
			glEnd();
		}
	}
}

void Pyramid() {
	glColor3d(1, 1, 1);
	glBegin(GL_QUADS);  //funkcja rysuje wielokąt
	glNormal3f(0, -1.0f, 0);
	glTexCoord2f(0.0f, 0.0f);
	glVertex3f(-5, -2, -5);

	glNormal3f(0, -1, 0);
	glTexCoord2f(1.0f, 0.0f);
	glVertex3f(-5, -2, 5);

	glNormal3f(0, -1, 0);
	glTexCoord2f(1.0f, 1.0f);
	glVertex3f(5, -2, 5);

	glNormal3f(0, -1, 0);
	glTexCoord2f(0.0f, 1.0f);
	glVertex3f(5, -2, -5);

	glEnd();

	glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
```

```
        glNormal3f(-1/sqrt(2), 1 / sqrt(2), 0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5, -2, -5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-5, -2, 5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(0, 1 / sqrt(2), 1 / sqrt(2));
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5, -2, 5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(5, -2, 5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(1 / sqrt(2), 1 / sqrt(2), 0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(5, -2, 5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(5, -2, -5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();

        glBegin(GL_TRIANGLES);  //funkcja rysuje wielokąt
        glNormal3f(0, 1 / sqrt(2), -1 / sqrt(2));
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(5, -2, -5);

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-5, -2, -5);

        glTexCoord2f(0.5f, 0.5f);
        glVertex3f(0, 5, 0);
        glEnd();
}
// Funkcja określająca co ma być rysowane (zawsze wywoływana, gdy trzeba
// przerysować scenę)
void RenderScene(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // Czyszczenie okna aktualnym kolorem czyszczącym
        glLoadIdentity();
        // Czyszczenie macierzy bie??cej
        gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        // Zdefiniowanie położenia obserwatora
        //Axes();
        // Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej
        if (status3 == 0) {                                          // jeśli model
obrou kamerą
```

```
		if (status1 == 1) {                                 // jeśli lewy
klawisz myszy wcięnięty
			thetax += delta_x * pix2angle / 20;      // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay += delta_y * pix2angle / 20;
			if (thetay > M_PI / 2 - 0.000001) {
				thetay = M_PI / 2 - 0.000001;
			}
			else if (thetay < -M_PI / 2 + 0.000001) {
				thetay = -M_PI / 2 + 0.000001;
			}
		}
		else if (status2 == 1) {
			R1 += delta_y;
			if (R1 < 1) R1 = 1;
			else if (R1 > 25) R1 = 25;
		}
		viewer[0] = R1 * cos(thetax) * cos(thetay);
		viewer[1] = R1 * sin(thetay);
		viewer[2] = R1 * sin(thetax) * cos(thetay);
	}
	else if (status3 == 1) {                                //jeśli model
obrotu światłem
		if (status1 == 1) {                                         // jeśli
lewy klawisz myszy wcięnięty
			thetax1 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay1 += delta_y * pix2angle / 20;
			if (thetay1 > M_PI / 2 - 0.000001) {
				thetay1 = M_PI / 2 - 0.000001;
			}
			else if (thetay1 < -M_PI / 2 + 0.000001) {
				thetay1 = -M_PI / 2 + 0.000001;
			}
			light_position[0] = R1 * cos(-thetax1) * cos(-thetay1);
			light_position[1] = R1 * sin(-thetay1);
			light_position[2] = R1 * sin(-thetax1) * cos(-thetay1);
		}
		else if (status2 == 1) {
			thetax2 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay2 += delta_y * pix2angle / 20;
			if (thetay2 > M_PI / 2 - 0.000001) {
				thetay2 = M_PI / 2 - 0.000001;
			}
			else if (thetay2 < -M_PI / 2 + 0.000001) {
				thetay2 = -M_PI / 2 + 0.000001;
			}
			light_position1[0] = R1 * cos(-thetax2) * cos(-thetay2);
			light_position1[1] = R1 * sin(-thetay2);
			light_position1[2] = R1 * sin(-thetax2) * cos(-thetay2);
		}
	}
	glRotatef(0, 0.0, 0.0, 0.0);  //obrót obiektu o nowy kąt
	//glRotatef(thetay, 1.0, 0.0, 0.0);  //obrót obiektu o nowy kąt
	glLightfv(GL_LIGHT0, GL_POSITION, light_position);
	glLightfv(GL_LIGHT1, GL_POSITION, light_position1);
	glColor3f(1.0f, 1.0f, 1.0f);
	if (model == 2)Pyramid();
	else if(model ==1) Egg();
	// Narysowanie czajnika
	glFlush();
```

```
	// Przekazanie poleceń rysujących do wykonania
	glutSwapBuffers();
}
/*************************************************************************
/
// Funkcja ustalająca stan renderowania
void MyInit(void)
{
	GLbyte *pBytes;
	GLint ImWidth, ImHeight, ImComponents;
	GLenum ImFormat;
	// ..................................
	/*************************************************************************
******/
	// Definicja materiału z jakiego zrobiony jest czajnik
	GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
	// współczynniki ka =[kar,kag,kab] dla światła otoczenia
	GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	// współczynniki kd =[kdr,kdg,kdb] światła rozproszonego
	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 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[] = { -10.0, 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 = { 0.05 };
	// składowa liniowa dl dla modelu zmian oświetlenia w funkcji
	// odległości od źródła
	GLfloat att_quadratic = { 0.001 };
	// składowa kwadratowa dq dla modelu zmian oświetlenia w funkcji
	// odległości od źródła
/*************************************************************************
/
// Ustawienie parametrów materiału i źródła światła
/*************************************************************************
/
// 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);

        // 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

/*************************************************************************
/
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        // Kolor czyszczący (wypełnienia okna) ustawiono na czarny
        initEgg();
        // ................................
/*************************************************************************
/
// Teksturowanie będzie prowadzone tyko po jednej stronie ściany
        glEnable(GLUT_FULLY_COVERED);
        /*************************************************************************
******/
        //  Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga
        pBytes = LoadTGAImage("t_256.tga", &ImWidth, &ImHeight, &ImComponents,
&ImFormat);
        /*************************************************************************
*****/
   // Zdefiniowanie tekstury 2-D
        glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat,
GL_UNSIGNED_BYTE, pBytes);
        /*************************************************************************
******/
        // Zwolnienie pamięci

        free(pBytes);
        /*************************************************************************
******/
        // Włączenie mechanizmu teksturowania
        glEnable(GL_TEXTURE_2D);
        /*************************************************************************
******/
        // Ustalenie trybu teksturowania
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        /*************************************************************************
*****/
        // Określenie sposobu nakładania tekstur
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
/*************************************************************************
/
// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych
// w przypadku zmiany rozmiarów okna.
// Parametry vertical i horizontal (wysokość i szerokość okna) są
// przekazywane do funkcji za każdym razem gdy zmieni się rozmiar okna.

void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
        pix2angle = 360.0 / (float)horizontal;  // przeliczenie pikseli na stopnie
        glMatrixMode(GL_PROJECTION);
        // Przełączenie macierzy bieżącej na macierz projekcji
```

```
	glLoadIdentity();
	// Czyszcznie macierzy bieżącej
	gluPerspective(70, 1.0, 1.0, 30.0);
	// Ustawienie parametrów dla rzutu perspektywicznego
	if (horizontal <= vertical)
		glViewport(0, (vertical - horizontal) / 2, horizontal, horizontal);
	else
		glViewport((horizontal - vertical) / 2, 0, vertical, vertical);
	// Ustawienie wielkości okna okna widoku (viewport) w zależności
	// relacji pomiędzy wysokością i szerokością okna
	glMatrixMode(GL_MODELVIEW);
	// Przełączenie macierzy bieżącej na macierz widoku modelu
	glLoadIdentity();
	// Czyszczenie macierzy bieżącej
}

//fynkcja zwortna wyznaczają model rysowania jajka
void keys(unsigned char key, int x, int y)
{
	if (key == 'e') model = 1; //siatka
	if (key == 'p') model = 2; //wypoełnione trójkąty
	if (key == 'r') status3 = 0; //rotacja kamery
	if (key == 'l') status3 = 1; //rotacja światła
	RenderScene(); // przerysowanie obrazu sceny
}

/*************************************************************************************
/
// Główny punkt wejścia programu. Program działa w trybie konsoli

void main(void)
{
	cout << "Przyciski: " << endl
		<< "l - kontrola światła" << endl
		<< "r - kontrola kamery" << endl
		<< "e - jajko" << endl
		<< "p - piramida" << endl;
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(300, 300);
	glutCreateWindow("Rzutowanie perspektywiczne");
	glutDisplayFunc(RenderScene);
	// Określenie, że funkcja RenderScene będzie funkcją zwrotną
	// (callback function).  Będzie ona wywoływana za każdym razem
	// gdy zajdzie potrzeba przerysowania okna
	glutReshapeFunc(ChangeSize);
	// Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną
	// za zmiany rozmiaru okna
	MyInit();
	// Funkcja MyInit() (zdefiniowana powyżej) wykonuje wszelkie
	// inicjalizacje konieczne  przed przystąpieniem do renderowania
	glEnable(GL_DEPTH_TEST);
	// Włączenie mechanizmu usuwania niewidocznych elementów sceny
	glutKeyboardFunc(keys);
	glutMouseFunc(Mouse);
	// Ustala funkcję zwrotną odpowiedzialną za badanie ruchu myszy
	glutMotionFunc(Motion);
	glutMainLoop();
	// Funkcja uruchamia szkielet biblioteki GLUT
}
/*************************************************************************************
/
```