**POLITECHNIKA WROCŁAWSKA**
**Instytut Informatyki, Automatyki i Robotyki**
**Zakład Systemów Komputerowych**

**Grafika komputerowa**

**Kurs: INEK00012L**

**Sprawozdanie z ćwiczenia nr 6**

# OPEN GL – teksturowanie powierzchni obiektów

| | |
|---|---|
| **Wykonała:** | **Weronika Luźna** |
| **Termin:** | **PN/N 16.15-19.15** |
| **Data wykonania ćwiczenia:** | **11.01.2016** |
| **Data oddania sprawozdania:** | **25.01.2016** |
| **Ocena:** | |

| |
|---|
| **Uwagi prowadzącego:** |

**1. Cel projektu.**

Celem projektu było zaimplementowanie 2 programów do teksturowania obiektów 3-D.

**2. Ostrosłup.**

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

2.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          piramida.cpp
//
//  OPIS:                           Program służy do rysowania otekstorowanego ostrosłupa.
//
//  AUTOR:                          Weronika Luźna
//
//  DATA                            25 Stycznia 2016 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                          Kompilator:         Microsoft Visual Studio 2015
//
//  MATERIA£Y               http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_6_dz/
//     ŹRÓDŁOWE:
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

#define _CRT_SECURE_NO_WARNINGS

#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <fstream>


typedef float point3[3];

static GLfloat viewer[] = { 0.0, 0.0, 5.0 };

static GLfloat       theta_y = 0.0, theta_x = 0.0;
static GLfloat       pix2angle_x, pix2angle_y;
static GLint  status = 0;

static int    x_pos_old = 0, y_pos_old = 0, delta_x = 0, delta_y = 0;
float verLength = 2.0;
bool sciany[5] = { true, true, true, true, true };

GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight, GLint
*ImComponents, GLenum *ImFormat)
{

#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;


/*****************************************************************************/
// Wartoœci domyœlne zwracane w przypadku b³êdu
/*****************************************************************************/

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

	pFile = fopen(FileName, "rb");
	if (pFile == NULL)
		return NULL;

/*****************************************************************************/
// Przeczytanie nag³ówka pliku
/*****************************************************************************/

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


/*****************************************************************************/
// Odczytanie szerokoœci, wysokoœci i g³êbi obrazu
/*****************************************************************************/

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


/*****************************************************************************/
// Sprawdzenie, czy g³êbia spe³nia za³o¿one warunki (8, 24, lub 32 bity)
/*****************************************************************************/

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

/*****************************************************************************/
// Obliczenie rozmiaru bufora w pamiêci
/*****************************************************************************/
```

```
lImageSize = tgaHeader.width * tgaHeader.height * sDepth;

/*****************************************************************************/
// Alokacja pamiêci 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;

}

void Axes(void) {
	point3  x_min = { -5.0, 0.0, 0.0 };
```

```
        point3  x_max = { 5.0, 0.0, 0.0 };

        point3  y_min = { 0.0, -5.0, 0.0 };
        point3  y_max = { 0.0, 5.0, 0.0 };

        point3  z_min = { 0.0, 0.0, -5.0 };
        point3  z_max = { 0.0, 0.0, 5.0 };

        glColor3f(1.0f, 0.0f, 0.0f);  // kolor rysowania osi - czerwony
        glBegin(GL_LINES); // rysowanie osi x
        glVertex3fv(x_min);
        glVertex3fv(x_max);
        glEnd();

        glColor3f(0.0f, 1.0f, 0.0f);  // kolor rysowania - zielony
        glBegin(GL_LINES);  // rysowanie osi y
        glVertex3fv(y_min);
        glVertex3fv(y_max);
        glEnd();

        glColor3f(0.0f, 0.0f, 1.0f);  // kolor rysowania - niebieski
        glBegin(GL_LINES); // rysowanie osi z
        glVertex3fv(z_min);
        glVertex3fv(z_max);
        glEnd();
}



void Pyramid(float x, float y, float z){

        glColor3f(1.0, 1.0, 1.0);

        if (sciany[0]){
                glBegin(GL_QUADS);
                glNormal3f(0.0, -1.0, 0.0);
                glTexCoord2f(0.0, 0.0);
                glVertex3f(x, y, z);
                glNormal3f(0.0, -1.0, 0.0);
                glTexCoord2f(1.0, 0.0);
                glVertex3f(x + verLength, y, z);
                glNormal3f(0.0, -1.0, 0.0);
                glTexCoord2f(1.0, 1.0);
                glVertex3f(x + verLength, y, z + verLength);
                glNormal3f(0.0, -1.0, 0.0);
                glTexCoord2f(0.0, 1.0);
                glVertex3f(x, y, z + verLength);
                glEnd();
        }

        if (sciany[1]){
                glBegin(GL_TRIANGLES);
                glNormal3f(0.0, 0.0, -1.0);
                glTexCoord2f(0.0, 0.0);
                glVertex3f(x, y, z);

                glNormal3f(0.0, 0.0, -1.0);
                glTexCoord2f(1.0, 0.0);
                glVertex3f(x + verLength / 2, y + 2.0, z + verLength / 2);

                glNormal3f(0.0, 0.0, -1.0);
                glTexCoord2f(0.5, 0.5);
```

```
                glVertex3f(x + verLength, y, z);
                glEnd();
        }

        if (sciany[2]){
                glBegin(GL_TRIANGLES);
                glNormal3f(1.0, 0.0, 0.0);
                glTexCoord2f(0.0, 0.0);
                glVertex3f(x + verLength, y, z);

                glNormal3f(1.0, 0.0, 0.0);
                glTexCoord2f(1.0, 0.0);
                glVertex3f(x + verLength / 2, y + 2.0, z + verLength / 2);

                glNormal3f(1.0, 0.0, 0.0);
                glTexCoord2f(0.5, 5.0);
                glVertex3f(x + verLength, y, z + verLength);
                glEnd();
        }

        if (sciany[3]){
                glBegin(GL_TRIANGLES);
                glNormal3f(0.0, 0.0, 1.0);
                glTexCoord2f(0.0, 0.0);
                glVertex3f(x + verLength, y, z + verLength);

                glNormal3f(0.0, 0.0, 1.0);
                glTexCoord2f(1.0, 0.0);
                glVertex3f(x + verLength / 2, y + 2.0, z + verLength / 2);

                glNormal3f(0.0, 0.0, 1.0);
                glTexCoord2f(0.5, 0.5);
                glVertex3f(x, y, z + verLength);
                glEnd();
        }

        if (sciany[4]){
                glBegin(GL_TRIANGLES);
                glNormal3f(-1.0, 0.0, 0.0);
                glTexCoord2f(0.0, 0.0);
                glVertex3f(x, y, z + verLength);

                glNormal3f(-1.0, 0.0, 0.0);
                glTexCoord2f(1.0, 0.0);
                glVertex3f(x + verLength / 2, y + 2.0, z + verLength / 2);

                glNormal3f(-1.0, 0.0, 0.0);
                glTexCoord2f(0.5, 0.5);
                glVertex3f(x, y, z);
                glEnd();
        }
}

void RenderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glLoadIdentity();

        if (status == 1) {   // jeœli lewy klawisz myszy wciœniêty
                theta_x += delta_x*pix2angle_x;          // modyfikacja k¹ta obrotu o k¹t
proporcjonalny
                theta_y += delta_y*pix2angle_y;          // do ró¿nicy po³o¿eñ kursora myszy
```

```
	}
	else if (status == 2) {
		viewer[2] += delta_y;
		if (viewer[2] <= 4.0)		// ograniczenie zblizenia
			viewer[2] = 4.0;
		if (viewer[2] >= 30) // ograniczenie oddalenia
			viewer[2] = 30;
	}

	gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

	//Axes();

	glRotatef(theta_x, 0.0, 1.0, 0.0);
	glRotatef(theta_y, 1.0, 0.0, 0.0);

	Pyramid(-1.0, 0.0, -1.0);

	//glutWireTeapot(3.0);

	/*glBegin(GL_TRIANGLES);

	glNormal3f(0.0f,0.0f,1.0f);
	glTexCoord2f(0.0f, 0.0f);
	glVertex3f(0.0f, 0.0f, 0.0f);

	glNormal3f(0.0f, 0.0f, 1.0f);
	glTexCoord2f(1.0f, 0.0f);
	glVertex3f(1.0f, 0.0f, 0.0f);

	glNormal3f(0.0f, 0.0f, 1.0f);
	glTexCoord2f(0.5f, 1.0f);
	glVertex3f(0.5f, 1.0f, 0.0f);

	glEnd();*/

	glFlush();
	glutSwapBuffers();
}

/*********************************************************************************/
//Zmiana rodzaju modelu w zaleznosci od klawisza
/*********************************************************************************/
void Keys(unsigned char key, int x, int y)
{
	if (key == '1') sciany[0] = !sciany[0];
	if (key == '2') sciany[1] = !sciany[1];
	if (key == '3') sciany[2] = !sciany[2];
	if (key == '4') sciany[3] = !sciany[3];
	if (key == '5') sciany[4] = !sciany[4];

	RenderScene();
}

void MyInit(void) {
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);


	GLbyte *pBytes;
	GLint ImWidth, ImHeight, ImComponents;
	GLenum ImFormat;
```

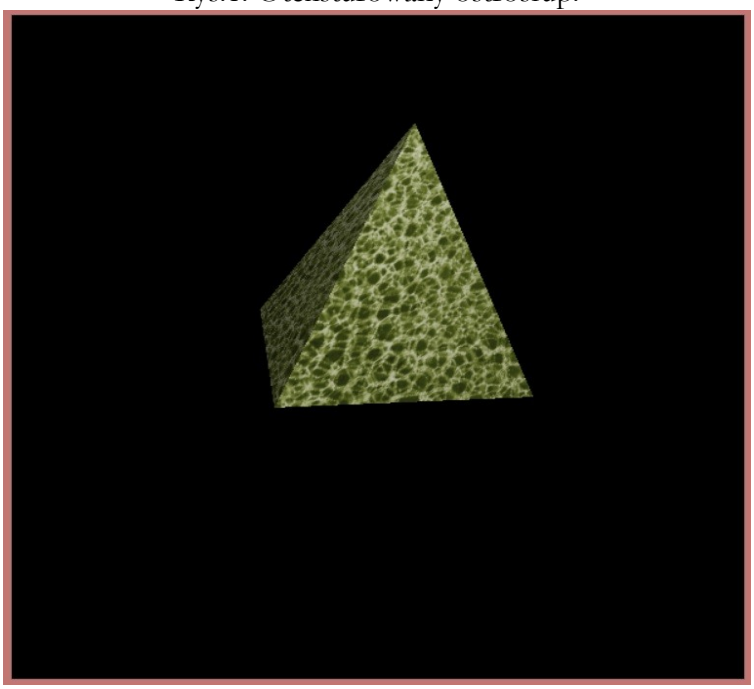
```
// Teksturowanie bêdzie prowadzone tyko po jednej stronie œciany
	glEnable(GL_CULL_FACE);

/*********************************************************************************/
//  Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga

	pBytes = LoadTGAImage("tekstura.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);

/*********************************************************************************/
// Zdefiniowanie tekstury 2-D

	glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat,
GL_UNSIGNED_BYTE, pBytes);

/*********************************************************************************/
// Zwolnienie pamiêci

	free(pBytes);

/*********************************************************************************/
// W³¹czenie mechanizmu teksturowania

	glEnable(GL_TEXTURE_2D);

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

	glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/*********************************************************************************/
// Okreœlenie sposobu nak³adania tekstur

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

/*********************************************************************************/
// Definicja materia³u i źródła oświetlenia

	GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki ka =[kar,kag,kab] dla œwiat³a otoczenia

	GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki kd =[kdr,kdg,kdb] œwiat³a rozproszonego

	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki ks =[ksr,ksg,ksb] dla œwiat³a odbitego

	GLfloat mat_shininess = { 20.0 };
	// wspó³czynnik n opisuj¹cy po³ysk powierzchni

	GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };
	// po³o¿enie Ÿród³a

	GLfloat light_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a otoczenia
	// Ia = [Iar,Iag,Iab]

	GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie dyfuzyjne Id = [Idr,Idg,Idb]

	GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie kierunkowe Is = [Isr,Isg,Isb]

	GLfloat att_constant = { 1.0 };
	// sk³adowa sta³a ds dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_linear = { 0.05f };
	// sk³adowa liniowa dl dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_quadratic = { 0.001f };
	// sk³adowa kwadratowa dq dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	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);

	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 oœwietlania sceny

	glShadeModel(GL_SMOOTH); // w³aczenie ³agodnego cieniowania
	glEnable(GL_LIGHTING);   // w³aczenie systemu oœwietlenia sceny
	glEnable(GL_LIGHT0);     // w³¹czenie Ÿród³a o numerze 0
	glEnable(GL_DEPTH_TEST); // w³¹czenie mechanizmu z-bufora
}

void ChangeSize(GLsizei horizontal, GLsizei vertical) {
	pix2angle_x = 360.0 / (float)horizontal; // przeliczenie pikseli na stopnie
	pix2angle_y = 360.0 / (float)vertical;

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	gluPerspective(80, 1.0, 1.0, 30.0);

	if (horizontal <= vertical)
		glViewport(0, (vertical - horizontal) / 2, horizontal, horizontal);
	else
		glViewport((horizontal - vertical) / 2, 0, vertical, vertical);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

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
		status = 1;          // wciœniêty zosta³ lewy klawisz myszy
```

```
	}
	else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
		y_pos_old = y;			// przypisanie aktualnie odczytanej pozycji kursora
		// jako pozycji poprzedniej
		status = 2;			//wciœniêty zosta³ prawy klawisz myszy
	}
	else
		status = 0;		// nie zosta³ wciœniêty ¿aden klawisz
}

void Motion(GLsizei x, GLsizei y) {
	delta_x = x - x_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	x_pos_old = x;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	delta_y = y - y_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	y_pos_old = y;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	glutPostRedisplay(); // przerysowanie obrazu sceny
}

void main(void) {
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(600, 600);

	glutCreateWindow("Piramida teksturowa");

	glutDisplayFunc(RenderScene);
	glutReshapeFunc(ChangeSize);
	MyInit();
	glEnable(GL_DEPTH_TEST);
	glutMouseFunc(Mouse);
	glutMotionFunc(Motion);
	glutKeyboardFunc(Keys);
	glutMainLoop();
}
```

2.2. Demonstracja działania.

Rys.1. Oteksturowany ostrosłup.



### 3. Jajko

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

3.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          Source.cpp
//
//  OPIS:                           Program służy do rysowania otekstorowanego jajka.
//
//  AUTOR:                          Weronika Luźna
//
//  DATA                            25 Stycznia 2016 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                          Kompilator:         Microsoft Visual Studio 2015
//
//  MATERIA£Y               http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_6_dz/
//     ŹRÓDŁOWE:
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <time.h>
#include <stdio.h>

typedef float point3[3];
typedef float point2[2];

const float PI = 3.14159265;
point3 **coordinates;
point3 **norm_coordinates;
point2 **file_coordinates;

static GLfloat viewer[] = { 0.0, 0.0, 10.0 };

static GLfloat pix2angle;     // przelicznik pikseli na stopnie
static GLfloat       phi = 0.0, theta = 0.0;
static GLfloat       pix2angle_x = 0.0, pix2angle_y = 0.0;
static GLint  status = 0;
static int    x_pos_old = 0, y_pos_old = 0;
static int    delta_x = 0, delta_y = 0;


//Parametry programu
int N = 50;                         //Liczba punktow na jaka dzielimy kwadrat jednostkowy
float verLength = 1.0;  //Dlugosc boku kwadratu
float viewerR = 10.0;       //Promien sfery obserwatora

//Tablica katow obrotu
```

```
static GLfloat angle[] = { 0.0, 0.0, 0.0 };

GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight, GLint
*ImComponents, GLenum *ImFormat)
{

#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;

/*****************************************************************************/
// Wartoœci domyœlne zwracane w przypadku b³êdu

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

	pFile = fopen(FileName, "rb");
	if (pFile == NULL)
		return NULL;
/*****************************************************************************/
// Przeczytanie nag³ówka pliku

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

/*****************************************************************************/
// Odczytanie szerokoœci, wysokoœci i g³êbi obrazu

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

/*****************************************************************************/
// Sprawdzenie, czy g³êbia spe³nia za³o¿one warunki (8, 24, lub 32 bity)

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

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

```
// Obliczenie rozmiaru bufora w pamiêci

	lImageSize = tgaHeader.width * tgaHeader.height * sDepth;

/*****************************************************************************/
// Alokacja pamiêci 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;
}

float Calculate_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 Calculate_y(float u, float v) {
	float y;

	y = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2);
	return y - 5;
}

float Calculate_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 Calculate_norm_x(float u, float v) {
	float x, a = v*PI;

	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	x = (GLfloat)(yu*zv - zu*yv);
	return x;
}

float Calculate_norm_y(float u, float v) {
	float y, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
* u)*sin(a);
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	y = (GLfloat)(zu*xv - xu*zv);
	return y;
}

float Calculate_norm_z(float u, float v) {
	float z, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
* u)*sin(a);
	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;

	z = (GLfloat)(xu*yv - yu*xv);
	return z;
}


void Mesh() {
	float stepXY = verLength / N;

	for (int i = 0; i<N + 1; i++) {
		for (int j = 0; j<N + 1; j++) {
			coordinates[i][j][0] = j*stepXY;
			coordinates[i][j][1] = i*stepXY;
		}
	}

	float u, v;
	for (int i = 0; i<N + 1; i++) {
		for (int j = 0; j<N + 1; j++) {
			v = coordinates[i][j][0];
			u = coordinates[i][j][1];

			file_coordinates[i][j][0] = v;
			file_coordinates[i][j][1] = u;
```

```
                    coordinates[i][j][0] = Calculate_x(u, v);
                    coordinates[i][j][1] = Calculate_y(u, v);
                    coordinates[i][j][2] = Calculate_z(u, v);

                    float x = Calculate_norm_x(u, v);
                    float y = Calculate_norm_y(u, v);
                    float z = Calculate_norm_z(u, v);

                    if (i < N / 2) {
                           norm_coordinates[i][j][0] = x / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                           norm_coordinates[i][j][1] = y / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                           norm_coordinates[i][j][2] = z / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                    }
                    if (i > N / 2) {
                           norm_coordinates[i][j][0] = -1.0*x / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                           norm_coordinates[i][j][1] = -1.0*y / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                           norm_coordinates[i][j][2] = -1.0*z / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                    }

                    if (i == N / 2) {
                           norm_coordinates[i][j][0] = 0;
                           norm_coordinates[i][j][1] = 1;
                           norm_coordinates[i][j][2] = 0;
                    }

                    if (i == 0 || i == N)
                    {
                           norm_coordinates[i][j][0] = 0;
                           norm_coordinates[i][j][1] = -1;
                           norm_coordinates[i][j][2] = 0;
                    }
             }
       }
}


void Egg(void) {

       Mesh();
       glColor3f(1.0, 1.0, 1.0);

       for (int i = 0; i < N; i++) {
              for (int j = 0; j < N; j++) {

                    glBegin(GL_TRIANGLES);
                    glNormal3fv(norm_coordinates[i][j + 1]);
                    glTexCoord2fv(file_coordinates[i][j + 1]);
                    glVertex3fv(coordinates[i][j + 1]);
                    glNormal3fv(norm_coordinates[i + 1][j]);
                    glTexCoord2fv(file_coordinates[i + 1][j]);
                    glVertex3fv(coordinates[i + 1][j]);
                    glNormal3fv(norm_coordinates[i + 1][j + 1]);
                    glTexCoord2fv(file_coordinates[i + 1][j + 1]);
                    glVertex3fv(coordinates[i + 1][j + 1]);
                    glEnd();
```

```
                        glBegin(GL_TRIANGLES);
                        glNormal3fv(norm_coordinates[i][j]);
                        glTexCoord2fv(file_coordinates[i][j]);
                        glVertex3fv(coordinates[i][j]);
                        glNormal3fv(norm_coordinates[i + 1][j]);
                        glTexCoord2fv(file_coordinates[i + 1][j]);
                        glVertex3fv(coordinates[i + 1][j]);
                        glNormal3fv(norm_coordinates[i][j + 1]);
                        glTexCoord2fv(file_coordinates[i][j + 1]);
                        glVertex3fv(coordinates[i][j + 1]);
                        glEnd();
                }
        }
}

void Axes(void)
{
        point3  x_min = { -2.0, 0.0, 0.0 };
        point3  x_max = { 2.0, 0.0, 0.0 };
        // pocz¹tek i koniec obrazu osi x

        point3  y_min = { 0.0, -2.0, 0.0 };
        point3  y_max = { 0.0,  2.0, 0.0 };
        // pocz¹tek i koniec obrazu osi y

        point3  z_min = { 0.0, 0.0, -2.0 };
        point3  z_max = { 0.0, 0.0,  2.0 };
        //  pocz¹tek i koniec obrazu osi y

        glColor3f(1.0f, 0.0f, 0.0f);  // kolor rysowania osi - czerwony
        glBegin(GL_LINES); // rysowanie osi x
        glVertex3fv(x_min);
        glVertex3fv(x_max);
        glEnd();

        glColor3f(0.0f, 1.0f, 0.0f);  // kolor rysowania - zielony
        glBegin(GL_LINES);  // rysowanie osi y
        glVertex3fv(y_min);
        glVertex3fv(y_max);
        glEnd();

        glColor3f(0.0f, 0.0f, 1.0f);  // kolor rysowania - niebieski
        glBegin(GL_LINES); // rysowanie osi z
        glVertex3fv(z_min);
        glVertex3fv(z_max);
        glEnd();
}

void RenderScene(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // Czyszczenie okna aktualnym kolorem czyszcz¹cym
        glLoadIdentity();
        // Czyszczenie macierzy bie¿¹cej

        if (status == 1)                    // jeœli lewy klawisz myszy wciêniêty
        {
                theta += 0.01*delta_x*pix2angle;
                phi += 0.01*delta_y*pix2angle;
                // modyfikacja k¹ta obrotu o kat proporcjonalny
        }
```

```
	//Wspolrzedne obserwatora
	viewer[0] = viewerR * cos(theta) * cos(phi);
	viewer[1] = viewerR * sin(phi);
	viewer[2] = viewerR * sin(theta) * cos(phi);

	gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, cos(phi), 0.0);
	// Zdefiniowanie po³o¿enia obserwatora

	Axes();
	// Narysowanie osi przy pomocy funkcji zdefiniowanej powy¿ej

	//Rotacje
	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);

	//Renderowanie jajka
	Egg();

	glFlush();
	// Przekazanie poleceñ rysuj¹cych do wykonania

	glutSwapBuffers();
}

//Funkcja callback dla obrotu
void spinEgg()
{

	angle[0] -= 0.5;
	if (angle[0] > 360.0) angle[0] -= 360.0;
	angle[1] -= 0.5;
	if (angle[1] > 360.0) angle[1] -= 360.0;
	angle[2] -= 0.5;
	if (angle[2] > 360.0) angle[2] -= 360.0;

	glutPostRedisplay(); //odœwie¿enie zawartoœci aktualnego okna
}

// Funkcja ustalaj¹ca stan renderowania
void MyInit(void)
{
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

// Zmienne dla obrazu tekstury
	GLbyte *pBytes;
	GLint ImWidth, ImHeight, ImComponents;
	GLenum ImFormat;

// Definicja materia³u z jakiego zrobiony jest przedmiot
	GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };

	GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };

	GLfloat mat_shininess = { 100.0 };

	GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };

	GLfloat light_ambient[] = { 0.1f, 0.1f, 0.1f, 1.0f };
```

```
	GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

	GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };

	GLfloat att_constant = { 1.0 };

	GLfloat att_linear = { 0.05f };

	GLfloat att_quadratic = { 0.001f };

// Ustawienie patrametrów materia³u
//-----------------------------------------------------
	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 Ÿród³a œwiat³a
//-----------------------------------------------------
	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 oœwietlania sceny
//-----------------------------------------------------
	glShadeModel(GL_SMOOTH); // w³aczenie ³agodnego cieniowania
	glEnable(GL_LIGHTING);   // w³aczenie systemu oœwietlenia sceny
	glEnable(GL_LIGHT0);     // w³¹czenie Ÿród³a o numerze 0
	glEnable(GL_DEPTH_TEST); // w³¹czenie mechanizmu z-bufora

// Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga
	pBytes = LoadTGAImage("tekstura.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);

// Zdefiniowanie tekstury 2-D
	glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat,
GL_UNSIGNED_BYTE, pBytes);

// Zwolnienie pamiêci
	free(pBytes);

// W³¹czenie mechanizmu teksturowania
	glEnable(GL_TEXTURE_2D);

// Ustalenie trybu teksturowania
	glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

// Okreœlenie sposobu nak³adania tekstur
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}


void ChangeSize(GLsizei horizontal, GLsizei vertical) {
	pix2angle_x = 360.0*0.1 / (float)horizontal; // przeliczenie pikseli na stopnie
	pix2angle_y = 360.0*0.1 / (float)vertical;
```

```
	glMatrixMode(GL_PROJECTION);
	// Przeł¹czenie macierzy bie¿¹cej na macierz projekcji

	glLoadIdentity();
	// Czyszcznie macierzy bie¿¹cej

	gluPerspective(70.0, 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 wielkoœci okna okna widoku (viewport) w zale¿noœci
	// relacji pomiêdzy wysokoœci¹ i szerokoœci¹ okna

	glMatrixMode(GL_MODELVIEW);
	// Przeł¹czenie macierzy bie¿¹cej na macierz widoku modelu

	glLoadIdentity();
	// Czyszczenie macierzy bie¿¹cej
}

void Mouse(int btn, int state, int x, int y) {
	if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		x_pos_old = x;
		y_pos_old = y;				// przypisanie aktualnie odczytanej
pozycji kursora
								// jako pozycji poprzedniej
		status = 1;		// wciêniêty zosta³ lewy klawisz myszy
	}
	else
		status = 0;		// nie zosta³ wciœniêty ¿aden klawisz
}

void Motion(GLsizei x, GLsizei y) {
	delta_x = x - x_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	x_pos_old = x;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	delta_y = y - y_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	y_pos_old = y;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	glutPostRedisplay(); // przerysowanie obrazu sceny
}

void main(void)
{
	srand((unsigned)time(NULL));

	coordinates = new point3*[N + 1];
	for (int i = 0; i<N + 1; i++) {
		coordinates[i] = new point3[N + 1];
	}

	norm_coordinates = new point3*[N + 1];
	for (int i = 0; i < N + 1; i++) {
		norm_coordinates[i] = new point3[N + 1];
	}

	file_coordinates = new point2*[N + 1];
	for (int i = 0; i < N + 1; i++) {
```

```
		file_coordinates[i] = new point2[N + 1];
	}

	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(800, 600);

	glutCreateWindow("Oteksturowane jajko");

	glutDisplayFunc(RenderScene);

	glutReshapeFunc(ChangeSize);

	MyInit();

	glutMouseFunc(Mouse);

	glutMotionFunc(Motion);

	glutIdleFunc(spinEgg);

	glutMainLoop();

	for (int i = 0; i < N + 1; i++) {
		delete[] coordinates[i];
		delete[] norm_coordinates[i];
		delete[] file_coordinates[i];
		coordinates[i] = 0;
		norm_coordinates[i] = 0;
		file_coordinates[i] = 0;
	}
	delete[] coordinates;
	delete[] norm_coordinates;
	delete[] file_coordinates;
	coordinates = 0;
	norm_coordinates = 0;
	file_coordinates = 0;
}
```

3.2. Demonstracja działania.

Rys.2. Oteksturowane jajko.