pwr**POLITECHNIKA WROCŁAWSKA**

**Katedra Informatyki Technicznej**

**Zakład Systemów Komputerowych i Dyskretnych**

**Grafika komputerowa i komunikacja człowiek – komputer**

**Kurs: INEK00012L**

**Sprawozdanie z ćwiczenia nr 6**

**„OpenGL – teksturowanie powierzchni obiektów”**

|  |  |
| --- | --- |
| **Wykonał:** | **Adrian Frydmański** |
| **Termin:** | **WT/P 12:00-15:00** |
| **Data wykonania ćwiczenia:** | **8 XII 2015** |
| **Data oddania sprawozdania:** | **5 I 2016** |
| **Ocena:** |  |

|  |
| --- |
| **Uwagi prowadzącego:** |

Spis treści

[Wstęp teoretyczny 2](#_Toc439762863)

[Kod źródłowy dla ostrosłupa i trójkąta 2](#_Toc439762864)

[Opis teksturowania 8](#_Toc439762865)

[Opis działania rysowania ostrosłupa i trójkąta 8](#_Toc439762866)

[Kod źródłowy dla jajka 11](#_Toc439762867)

[Opis działania rysowania jajka 17](#_Toc439762868)

[Podsumowanie 18](#_Toc439762869)

# Wstęp teoretyczny

Celem ćwiczenia było pokazanie, jak przy pomocy OpenGL można teksturować obiekt trójwymiarowy. Teksturowane były trzy obiekty, ostrosłup, trójkąt i jajko.

# Kod źródłowy dla ostrosłupa i trójkąta

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Adrian Frydmański

// 209865

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <windows.h>

#include <gl/gl.h>

#include <gl/glut.h>

#include <math.h>

#include <time.h>

#include <stdio.h>

using namespace std;

typedef float point3[3];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Stałe i zmienne globalne:

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 };

bool walls[5] = { true, true, true, true, true };

bool pt[2] = { true, false };

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Rysowanie trójkąta

void triangle()

{

glColor3f(1, 1, 1);

//rysowanie trójkąta

glBegin(GL\_TRIANGLES);

glNormal3f(0, 0, 1);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-4, -2, 0);

glNormal3f(0, 0, 1);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(4, -2, 0);

glNormal3f(0, 0, 1);

glTexCoord2f(0.5f, 1.0f);

glVertex3f(0, 3, 0);

glEnd();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Rysowanie ostrosłupa

void pyramid()

{

//ostrosłup

float v[5][3] = { { -5,-2,5 },{ 5,-2,5 },{ 5,-2,-5 },{ -5,-2,-5 },{ 0,5,0 } };

float n[5][3] = { { 0, 0.55, 0.77 }, { 0.77, 0.55, 0 }, { 0, 0.55, -0.77 }, { -0.77, 0.55, 0 }, { 0, -1, 0 } };

if (walls[0])

{

glBegin(GL\_TRIANGLES);

glNormal3fv(n[0]);

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(v[0]);

glNormal3fv(n[0]);

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(v[1]);

glNormal3fv(n[0]);

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(v[4]);

glEnd();

}

//

if (walls[1])

{

glBegin(GL\_TRIANGLES);

glNormal3fv(n[1]);

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(v[1]);

glNormal3fv(n[1]);

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(v[2]);

glNormal3fv(n[1]);

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(v[4]);

glEnd();

}

//

if (walls[2])

{

glBegin(GL\_TRIANGLES);

glNormal3fv(n[2]);

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(v[2]);

glNormal3fv(n[2]);

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(v[3]);

glNormal3fv(n[2]);

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(v[4]);

glEnd();

}

//

if (walls[3])

{

glBegin(GL\_TRIANGLES);

glNormal3fv(n[3]);

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(v[3]);

glNormal3fv(n[3]);

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(v[0]);

glNormal3fv(n[3]);

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(v[4]);

glEnd();

}

//

if (walls[4])

{

glBegin(GL\_QUADS);

glNormal3fv(n[4]);

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(v[0]);

glNormal3fv(n[4]);

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(v[3]);

glNormal3fv(n[4]);

glTexCoord2f(1.0f, 1.0f);

glVertex3fv(v[2]);

glNormal3fv(n[4]);

glTexCoord2f(0.0f, 1.0f);

glVertex3fv(v[1]);

glEnd();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 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);

// zakomentować niepotrzebne

if (pt[0]) pyramid();

if (pt[1]) triangle();

// Przekazanie poleceń rysujących do wykonania

glFlush();

glutSwapBuffers();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 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;

pFile = fopen(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 zwrotna dla obrotu

void spin()

{

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)

{

// Zmienne dla obrazu tekstury

GLbyte \*pBytes;

GLint ImWidth, ImHeight, ImComponents;

GLenum ImFormat;

// Teksturowanie będzie prowadzone tyko po jednej stronie ściany

glEnable(GL\_CULL\_FACE);

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

pBytes = LoadTGAImage("teksturka2.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);

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Kolor czyszcący (wypełnienia okna) ustawiono na czarny

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 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[] = { 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();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// funkcja dla klawiatury

void keys(unsigned char key, int x, int y)

{

if (key == '2') walls[0] = !walls[0];

if (key == '6') walls[1] = !walls[1];

if (key == '8') walls[2] = !walls[2];

if (key == '4') walls[3] = !walls[3];

if (key == '5') walls[4] = !walls[4];

if (key == 'p') pt[0] = !pt[0];

if (key == 't') pt[1] = !pt[1];

RenderScene(); // przerysowanie obrazu sceny

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

void main(void)

{

// inicjowanie randa

srand((unsigned)time(NULL));

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(800, 800);

glutCreateWindow("Tekstury");

// 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 obrotu i klawiszy

glutIdleFunc(spin);

glutKeyboardFunc(keys);

// główna petla GLUTa

glutMainLoop();

}

# Opis teksturowania

Teksturowanie pojedynczej figury odbywa się przez podanie następujących współrzędnych każdego z punktów (w nawiasach nazwy wywoływanych funkcji):

Rysunek 1 Diagram opisujący teksturowanie figury

Punkty powinny być podawane przeciwnie do ruchu wskazówek zegara. Podanie ich w odwrotną stronę spowodowałoby zateksturowanie drugiej strony i przy wywołaniu podczas inicjalizacji funkcji glEnable(GL\_CULL\_FACE) narysowanie trójkąta tylko w przypadku, kiedy byłby widoczny jego awers z teksturą. Rewers pozostałby niewidoczny.

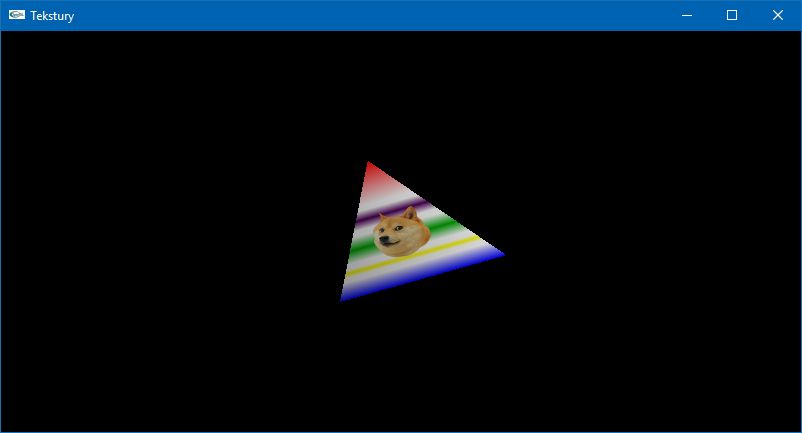
# Opis działania rysowania ostrosłupa i trójkąta

Po zakomentowaniu wywołania funkcji triangle lub pyramid w RenderScene można było wybrać rysowanie trójkąta, bądź całego ostrosłupa. Trójkąt otrzymano dzięki użyciu GL\_TRIANGLE. Na ostrosłup składają się trójkąty i kwadrat GL\_QUADS.

Użyto poniższej tekstury:

  
Rysunek 2 Tekstura dla trójkąta i ostrosłupa

Narysowany trójkąt wygląda następująco:

  
Rysunek 3 Zateksturowany trójkąt

Rysowanie ostrosłupa odbywa się przez narysowanie czterech trójkątów i kwadratu (GL\_QUADS). Współrzędne zostały dobrane zgodnie z poniższym rysunkiem i umieszczone w tablicy v:

Rysunek 4 Ostrosłup do narysowania

(-5, -2, 5)

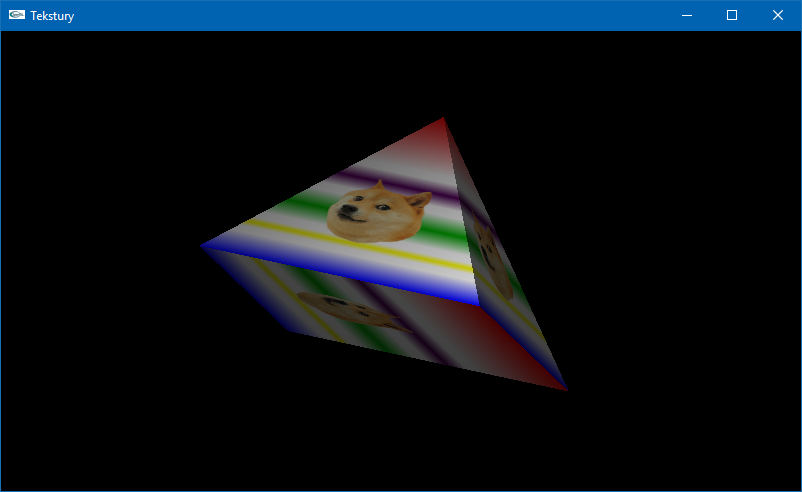
(5, -2, 5)

(5, -2, -5)

(-5, -2, -5)

(0, 5, 0)

Dodatkowo należy podać wektory normalne dla punktów. Są one podane w tablicy n dla każdej z pięciu płaszczyzn. Po narysowaniu obiekt prezentuje się następująco:

  
Rysunek 5 Zateksturowany ostrosłup

Możliwe jest sterowanie widocznością. Klawisze *P* i *T* umożliwiają włączenie i wyłączenie ostrosłupa i samego trójkąta. Klawiszami *2*, *4*, *6* i *8* włączane i wyłączane są ściany, *5* zaś podstawa ostrosłupa.

# Kod źródłowy dla jajka

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Adrian Frydmański

// 209865

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <windows.h>

#include <gl/gl.h>

#include <gl/glut.h>

#include <stdio.h>

#include <math.h>

const float pi = 3.1415926535897932384626433832795;

const int n = 512; // liczba punktów w kwadracie jednostkowym

typedef float point3[3];

point3 pTab[n][n]; // Deklaracja tablicy zawierajacej punkty

point3 lTab[n][n]; // Deklaracja tablicy opisującej oświetlenie

point3 tTab[n][n]; // Deklaracja tablicy zawierającej współrzędne tekstury

static GLfloat thetaH = 0.0; // kąt obrotu obiektu (horizontal)

static GLfloat thetaV = 0.0; // kąt obrotu obiektu (vertical)

static GLfloat pix2angleH; // przelicznik pikseli na stopnie (horizontal)

static GLfloat pix2angleV; // przelicznik pikseli na stopnie (vertical)

// inicjalizacja położenia obserwatora

static GLfloat theta = 0.0; // kąt azymutu

static GLfloat phi = 0.0; // kąt elewacji

static GLfloat R = 16; // promień sfery

static GLfloat errY = 1;

static GLint status = 0; // stan klawiszy myszy: 0 - nie naciśnięto żadnego klawisza, 1 - naciśnięty został lewy klawisz

static int x\_pos\_old = 0; // poprzednia pozycja kursora myszy (x)

static int y\_pos\_old = 0; // poprzednia pozycja kursora myszy (y)

static int delta\_x = 0; // różnica pomiędzy pozycją bieżącą i poprzednią kursora myszy

static int delta\_y = 0; // różnica pomiędzy pozycją bieżącą i poprzednią kursora myszy

static float pos\_z = 10.0; // oddalenie obserwatora od jajka

static GLfloat viewer[] = { 0.0, 0.0, pos\_z }; // inicjalizacja położenia obserwatora

int mouse\_mode = 2; // Wybór trybu myszy: 1 - wyświetlanie jajka, 2 - położenie obserwatora

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie

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;

pFile = fopen(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 generujaca chmurę punktow w kształcie jajka

void EggInit()

{

// wypełnianie tablicy punktów

for (int i = 0; i<n; i++)

{

for (int k = 0; k<n; k++)

{

float u = (float)i / (n - 1);

float v = (float)k / (n - 1);

pTab[i][k][0] = (-90.0\*u\*u\*u\*u\*u + 225.0\*u\*u\*u\*u - 270.0\*u\*u\*u + 180.0\*u\*u - 45.0\*u)\*cos(pi\*v);

pTab[i][k][1] = (160.0\*u\*u\*u\*u - 320.0\*u\*u\*u + 160.0\*u\*u) - 5.0;

pTab[i][k][2] = (-90.0\*u\*u\*u\*u\*u + 225.0\*u\*u\*u\*u - 270.0\*u\*u\*u + 180.0\*u\*u - 45.0\*u)\*sin(pi\*v);

}

}

// wypełnianie tablicy oświetlenia

for (int i = 0; i<n; i++)

{

for (int k = 0; k<n; k++)

{

float u = (float)i / (n - 1);

float v = (float)k / (n - 1);

tTab[i][k][1] = u;

tTab[i][k][0] = 1 - v;

//

float xu = (-450 \* u\*u\*u\*u + 900 \* u\*u\*u - 810 \* u\*u + 360 \* u - 45)\*cos(pi\*v);

float xv = pi\*(90 \* u\*u\*u\*u\*u - 225 \* u\*u\*u\*u + 270 \* u\*u\*u - 180 \* u\*u + 45 \* u)\*sin(pi\*v);

float yu = 640 \* u\*u\*u - 960 \* u\*u + 320 \* u;

float yv = 0;

float zu = (-450 \* u\*u\*u\*u + 900 \* u\*u\*u - 810 \* u\*u + 360 \* u - 45)\*sin(pi\*v);

float zv = -pi\*(90 \* u\*u\*u\*u\*u - 225 \* u\*u\*u\*u + 270 \* u\*u\*u - 180 \* u\*u + 45 \* u)\*cos(pi\*v);

//

float x = (GLfloat)(yu\*zv - zu\*yv);

float y = (GLfloat)(zu\*xv - xu\*zv);

float z = (GLfloat)(xu\*yv - yu\*xv);

// normalizacja wektorów

if (i < n / 2)

{

lTab[i][k][0] = x / (float)sqrt(x\*x + y\*y + z\*z);

lTab[i][k][1] = y / (float)sqrt(x\*x + y\*y + z\*z);

lTab[i][k][2] = z / (float)sqrt(x\*x + y\*y + z\*z);

}

if (i >= n / 2)

{

lTab[i][k][0] = (-1)\*x / (float)sqrt(x\*x + y\*y + z\*z);

lTab[i][k][1] = (-1)\*y / (float)sqrt(x\*x + y\*y + z\*z);

lTab[i][k][2] = (-1)\*z / (float)sqrt(x\*x + y\*y + z\*z);

}

if (i == 0 || i == n - 1)

{

lTab[i][k][0] = 0;

lTab[i][k][1] = -1;

lTab[i][k][2] = 0;

}

}

}

}

void Egg()

{

EggInit();

// rysowanie

for (int i = 0; i<n; i++)

{

for (int k = 0; k<n; k++)

{

if (i < n - 1 && k < n - 1)

{

if (i < n / 2)

{

glBegin(GL\_TRIANGLES);

glNormal3fv(lTab[i][k]);

glTexCoord2f(tTab[i][k][0], tTab[i][k][1]);

glVertex3fv(pTab[i][k]);

glNormal3fv(lTab[i + 1][k]);

glTexCoord2f(tTab[i + 1][k][0], tTab[i + 1][k][1]);

glVertex3fv(pTab[i + 1][k]);

glNormal3fv(lTab[i + 1][k + 1]);

glTexCoord2f(tTab[i + 1][k + 1][0], tTab[i + 1][k + 1][1]);

glVertex3fv(pTab[i + 1][k + 1]);

glNormal3fv(lTab[i][k]);

glTexCoord2f(tTab[i][k][0], tTab[i][k][1]);

glVertex3fv(pTab[i][k]);

glNormal3fv(lTab[i + 1][k + 1]);

glTexCoord2f(tTab[i + 1][k + 1][0], tTab[i + 1][k + 1][1]);

glVertex3fv(pTab[i + 1][k + 1]);

glNormal3fv(lTab[i][k + 1]);

glTexCoord2f(tTab[i][k + 1][0], tTab[i][k + 1][1]);

glVertex3fv(pTab[i][k + 1]);

glEnd();

}

else

{

glBegin(GL\_TRIANGLES);

glNormal3fv(lTab[i][k]);

glTexCoord2f(tTab[i][k][0], tTab[i][k][1]);

glVertex3fv(pTab[i][k]);

glNormal3fv(lTab[i + 1][k]);

glTexCoord2f(tTab[i + 1][k][0], tTab[i + 1][k][1]);

glVertex3fv(pTab[i + 1][k]);

glNormal3fv(lTab[i + 1][k + 1]);

glTexCoord2f(tTab[i + 1][k + 1][0], tTab[i + 1][k + 1][1]);

glVertex3fv(pTab[i + 1][k + 1]);

glNormal3fv(lTab[i][k]);

glTexCoord2f(tTab[i][k][0], tTab[i][k][1]);

glVertex3fv(pTab[i][k]);

glNormal3fv(lTab[i][k + 1]);

glTexCoord2f(tTab[i][k + 1][0], tTab[i][k + 1][1]);

glVertex3fv(pTab[i][k + 1]);

glNormal3fv(lTab[i + 1][k + 1]);

glTexCoord2f(tTab[i + 1][k + 1][0], tTab[i + 1][k + 1][1]);

glVertex3fv(pTab[i + 1][k + 1]);

glEnd();

}

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja okreslajaca co ma byc rysowane (zawsze wywolywana gdy trzeba przerysowac scene)

void RenderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Czyszczenie okna aktualnym kolorem czyszczącym

glLoadIdentity(); // Czyszczenie macierzy bieżącej

// Zdefiniowanie położenia obserwatora

if (status == 1) // wciśnięty lewy przycisk myszy

{

// zmiana phi

phi += delta\_y\*pix2angleH / 10.0;

if (phi <= 0) phi += 2 \* 3.14;

if (phi >= 2 \* 3.14) phi -= 2 \* 3.14;

// zmiana theta

theta += delta\_x\*pix2angleV / 10.0;

if (theta <= 0) theta += 2 \* 3.14;

if (theta >= 2 \* 3.14) theta -= 2 \* 3.14;

// Sprawdzenie wartości kąta elewacji i modyfikacja

// składowej y wektora określającego skrócenie kamery

// co zapewni płynny obrót obiektu w płaszczyźnie y

if ((phi >= 0 && phi < 3.14 / 2) || (phi >= 3 \* 3.14 / 2 && phi < 2 \* 3.14)) errY = 1.0;

else errY = -1.0;

}

if (status == 2) // wciśnięty prawy przycisk myszy

{

R += delta\_x;

if (R <= 9.1) R = 9.0;

if (R >= 25.1) R = 25.0;

}

// ustawienie położenia obserwatora

viewer[0] = R\*cos(theta)\*cos(phi);

viewer[1] = R\*sin(phi);

viewer[2] = R\*sin(theta)\*cos(phi);

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

//

Egg(); // Rysowanie jajka

glFlush(); // Przekazanie poleceń rysujących do wykonania

glutSwapBuffers();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja przechwytująca stan myszy

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; // wciśnięty został lewy klawisz myszy

}

else if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

status = 2;

}

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

glutPostRedisplay();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 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 bieżacego położenia jako poprzednie

delta\_y = y - y\_pos\_old; // obliczenie różnicy położenia kursora myszy

y\_pos\_old = y; // podstawienie bieżacego położenia jako poprzednie

glutPostRedisplay(); // przerysowanie obrazu sceny

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja inicjująca scen? do przerysowania

void MyInit(void)

{

// Kolor czyszczący (wypełnienia okna) ustawiono na czarny

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

// Definicja materiału z jakiego zrobione jest jajko

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] dla 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[] = { 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 odległo?ci 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 światła

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

// Zmienne dla obrazu tekstury

GLbyte \*pBytes;

GLint ImWidth, ImHeight, ImComponents;

GLenum ImFormat;

//glEnable(GL\_CULL\_FACE); // Teksturowanie prowadzone tyko po jednej stronie ściany

pBytes = LoadTGAImage("teksturka3.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat); // Przeczytanie obrazu tekstury z pliku

glTexImage2D(GL\_TEXTURE\_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL\_UNSIGNED\_BYTE, pBytes); // Zdefiniowanie tekstury 2-D

free(pBytes); // zwolnienie pamięci

glEnable(GL\_TEXTURE\_2D); // Włączenie mechanizmu teksturowania

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE); // Ustalenie trybu teksturowania

// 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 stalych proporcji rysowanych w przypadku zmiany rozmiarow okna.

// Parametry vertical i horizontal (wysokosc i szerokosc okna) sa przekazywane do funkcji za kazdym razem gdy zmieni sie rozmiar okna.

void ChangeSize(GLsizei horizontal, GLsizei vertical)

{

pix2angleH = 360.0 / (float)horizontal; // przeliczenie pikseli na stopnie

pix2angleV = 360.0 / (float)vertical; // 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

// Ustawienie wielkości okna okna widoku (viewport) w zależności relacji pomiędzy wysokością i szerokością okna

if (horizontal <= vertical)

glViewport(0, (vertical - horizontal) / 2, horizontal, horizontal);

else

glViewport((horizontal - vertical) / 2, 0, vertical, vertical);

glMatrixMode(GL\_MODELVIEW); // Przełączenie macierzy bieżącej na macierz widoku modelu

glLoadIdentity(); // Czyszczenie macierzy bieżącej

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Główny punkt wejścia programu

void main(void)

{

// Ustawienie trybu wyswietlania GLUT\_DOUBLE - podwójne buforowanie, GLUT\_RGB - tryb RGB, GLUT\_DEPTH - bufor głębokości

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Teksturowanie obiektów 3D - Adrian Frydmański");

glutDisplayFunc(RenderScene);

glutReshapeFunc(ChangeSize);

glutMouseFunc(Mouse);

glutMotionFunc(Motion);

glEnable(GL\_DEPTH\_TEST);

MyInit();

glutMainLoop();

}

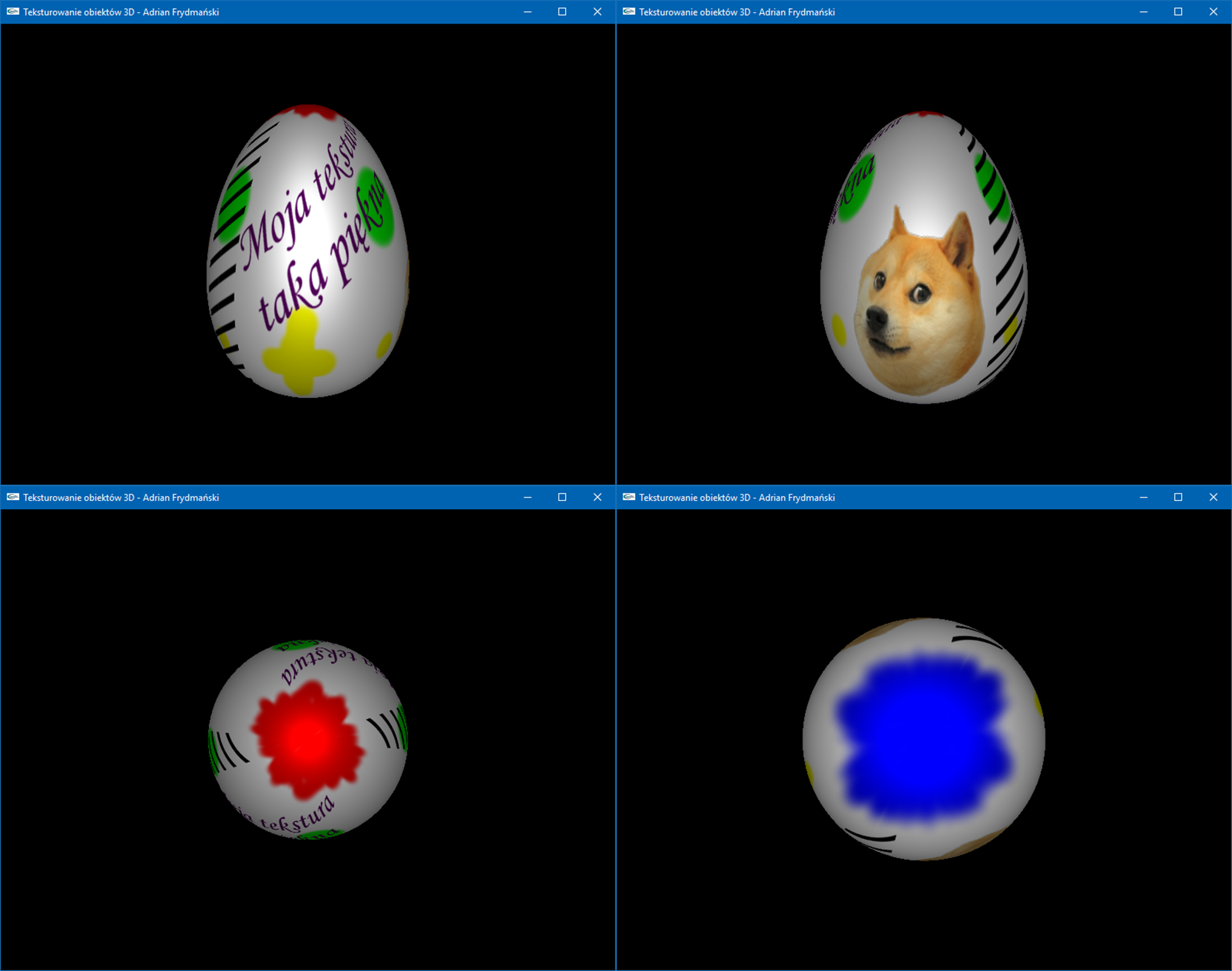
# Opis działania rysowania jajka

Kolejnym zadaniem było zateksturowanie jajka. Tak, jak punkty z kwadratu jednostkowego przekształcane są w model jajka, podobnie powinna zachować się tekstura. Stąd punkty zaczepienia tekstury również będą miały zakres [0,1], jak początkowo w kwadracie jednostkowym.

Użyto następującej tekstury, aby pokazać ciągłość na styku połówek jajka:

  
Rysunek 6 Tekstura dla jajka

Poniżej widoczne jest zateksturowane jajko z boku, góry i dołu:

  
Rysunek 7 Zateksturowane jajko z kilku stron

# Podsumowanie

Ćwiczenie pokazało, w jaki sposób teksturować obiekty trójwymiarowe w OpenGL i trudności z nim związane. Po raz kolejny ujawniło się, jak ważne jest podawanie punktów do rysowania w odpowiedniej kolejności.