Sprawozdanie z ćwiczenia 6
OpenGL – techniki teksturowania
Maksymilian Iwanow
209946
Poniedziałek 10.15

Celem ćwiczenia było poznanie podstawowych technik teksturowania powierzchni z wykorzystaniem OpenGL i GLUT.

Funkcja wczytywania tekstury:

```
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 MyInit():

```
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("P6_t.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);

//  Definicja materiału z jakiego zrobiony jest czajnik
//  i definicja źródła światła

// 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    = {0.05};
    // 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);
    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 ost() - tworzenie oraz teksturowanie „piramidy” :

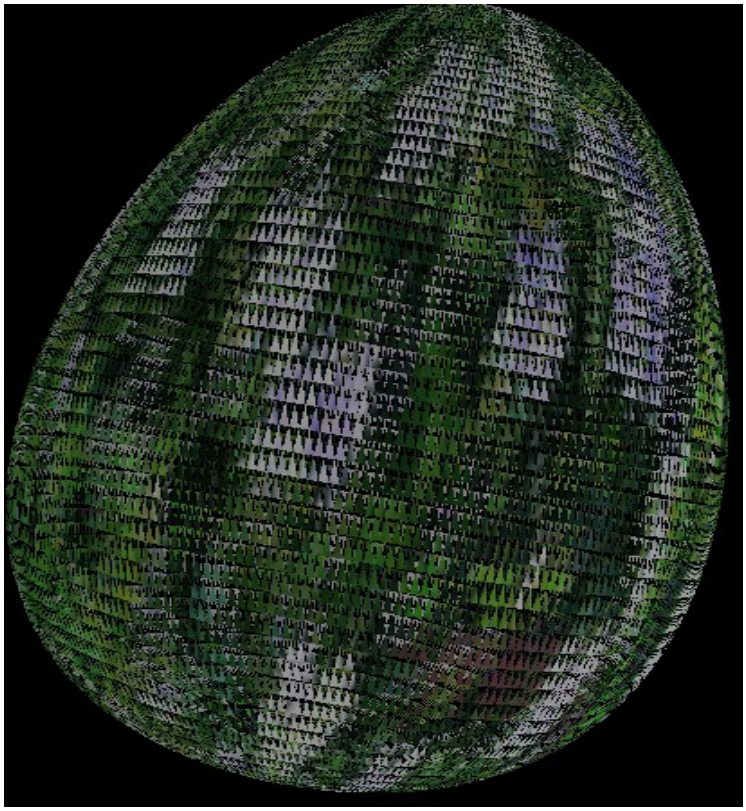
```
void ost()
{

        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 1.0);
        glVertex3f(-3.0,  3.0, 0.0); // D
        glTexCoord2f(1.0, 1.0);
        glVertex3f( 3.0,  3.0, 0.0); // C
        glTexCoord2f(1.0, 0.0);
        glVertex3f( 3.0, -3.0, 0.0); // B
        glTexCoord2f(0.0, 0.0);
        glVertex3f(-3.0, -3.0, 0.0); // A
        glEnd();


    glBegin(GL_TRIANGLES);


        glTexCoord2f(0.0, 0.0);
        glVertex3f(-3.0, -3.0, 0.0);
        glTexCoord2f(1.0, 0.0);
        glVertex3f( 3.0, -3.0, 0.0);
        glTexCoord2f(0.5, 0.5);
        glVertex3f( 0.0,  0.0, 5.0);

        glTexCoord2f(1.0, 0.0);
        glVertex3f( 3.0, -3.0, 0.0);
        glTexCoord2f(1.0, 1.0);
        glVertex3f( 3.0,  3.0, 0.0);
        glTexCoord2f(0.5, 0.5);
        glVertex3f( 0.0,  0.0, 5.0);

        glTexCoord2f(1.0, 1.0);
        glVertex3f( 3.0,  3.0, 0.0);
        glTexCoord2f(0.0, 1.0);
        glVertex3f(-3.0,  3.0, 0.0);
        glTexCoord2f(0.5, 0.5);
        glVertex3f( 0.0,  0.0, 5.0);

        glTexCoord2f(0.0, 1.0);
        glVertex3f(-3.0,  3.0, 0.0);
        glTexCoord2f(0.0, 0.0);
        glVertex3f(-3.0, -3.0, 0.0);
        glTexCoord2f(0.5, 0.5);
        glVertex3f( 0.0,  0.0, 5.0);


    glEnd();

}
```

Funkcja jajo3() - tworzenie teksturowanego jajka:

```
#define NF ((float) )

void jajo3()
{
	float u = 0, v=0;
	int N = 150;
	float tp[150][150][3];
	for(int i=0; i<N; ++i)
	{
		u = (float)i/(N-1);
		for(int j = 0; j<N; ++j)
		{
			v = (float)j/(N-1);
tp[i][j][0] = (-90*potg(u, 5)+225*potg(u, 4)-270*potg(u, 3)+180*potg(u, 2)-45*u)*cos(pi*v);
tp[i][j][1] = 160*potg(u, 4)-320*potg(u, 3)+160*potg(u, 2)-5.0;
tp[i][j][2] = (-90*potg(u, 5)+225*potg(u, 4)-270*potg(u, 3)+180*potg(u, 2)-45*u)*sin(pi*v);

		}
	}
	glBegin(GL_TRIANGLES);
	for(int i=0; i<N-1; i++)
	{

		for(int j=0; j<N-1; j++)
		{
			glBegin(GL_TRIANGLES);
			glTexCoord2f((i+1)/NF, j/NF);
			glVertex3fv(tp[i][j]);
			glTexCoord2f((i)/NF, (j+1)/NF);
			glVertex3fv(tp[i+1][j]);
			glTexCoord2f((i+1)/NF, j/NF);
			glVertex3fv(tp[i][j+1]);
			glEnd();
			glBegin(GL_TRIANGLES);
			glTexCoord2f((i+1)/NF, j/NF);
			glVertex3fv(tp[i+1][j+1]);
			glTexCoord2f((i)/NF, j/NF);
			glVertex3fv(tp[i+1][j]);
			glTexCoord2f((i+1)/NF, (j+1)/NF);
			glVertex3fv(tp[i][j+1]);
			glEnd();
		}
	}
	glEnd();
}
```

Wynik funkcji ost():

Wynik funkcji jajo3():



Wnioski, problemy, podsumowanie:

1. Największym problemem okazało się teksturowanie jajka – z powodu niedoskonałych połączeń poszczególnych trójkątów, tekstura nie wygląda idealnie.
2. Bardzo pomocne były materiały ze strony internetowej http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/ .
3. Kolejnym problemem było samo zrozumienie zasad teksturowania w przypadku tworzenia „piramidy”.