POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa i komunikacja człowiek - komputer

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 6

TEMAT ĆWICZENIA: OpenGL – Teksturowanie powierzchni obiektów

Wykonał:	**Bartosz Szymczak, nr indeksu 252734**
Termin:	**Poniedziałek TN 7:30-10:30**
Data wykonania ćwiczenia:	**03.01.2022r.**
Data oddania sprawozdania:	
Ocena:	

Uwagi prowadzącego:

Spis treści

1. Treść zadania

Na kolejnych zajęciach wykonano ćwiczenie o tematyce teksturowania powierzchni obiektów. Celem ćwiczenia jest pokazanie podstawowych technik teksturowania powierzchni obiektów z wykorzystaniem mechanizmów biblioteki OpenGL z rozszerzeniem GLUT. Pierwsze zadanie polega na stworzeniu teksturowanej piramidy, z możliwością usuwania kolejnych boków. Drugie zadanie polega na stworzeniu teksturowanego jajka, na podstawie tego z poprzednich ćwiczeń.

2. Wstęp teoretyczny

Teksturowanie polega na nanoszeniu na powierzchnię elementów modelu sceny obrazów zadanych w postaci map bitowych. Obrazy te zwykle umieszczone są w osobnych plikach. Ogólnie teksturowanie sprowadza się do trzech czynności. Na początku odczytywane są z pliku obrazu tekstury, a pobrane dane umieszczane są w odpowiednim miejscu pamięci. Dalej definiuje się tekstury, czyli określany jest sposób interpretacji pobranych danych. Na koniec tekstury nakładane są na elementy modelu obiektu.

3. Opis programu

3.1 Funkcja pobierająca tekstury

Funkcja LoadTGAImage bazuje na kodzie podanym w instrukcji do ćwiczenia. Jest ona odpowiedzialna za wczytywanie danych obrazu zapisanego w formacie TGA o nazwie FileName, alokuje pamięć i zwraca wskaźnik pBits do bufora w którym znajdują się pobrane dane. Dodatkowo udostępnia szerokość ImWidth, wysokość ImHeight obrazu tekstury oraz dane ImComponents i ImFormat opisujące format obrazu według specyfikacji OpenGL. Funkcja działa jedynie dla obrazów wykorzystujących 8, 24 i 32 bitowe kolory.

```
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)
    {
        cout << "Error: Failed to open a file" << endl;
        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)
    {
        cout << "Error: Incorrect file type" << endl;
        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)
    {
        cout << "Error: Failed to allocate memory for image data" << endl;
        return NULL;
    }
    if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
    {
        cout << "Error: Incorrect file data" << endl;
        free(pbitsperpixel);
        return NULL;
    }
    // Ustawienie formatu OpenGL
```

```
    switch (sDepth)
    {
    case 1:
    //Format RGBA, gdzie A = 1
        *ImFormat = GL_LUMINANCE;
        *ImComponents = GL_LUMINANCE8;
        break;
    case 3:
        //Format BGR, odpowiada pamięci map bitowych Windows (DIB)
        *ImFormat = GL_BGR_EXT;
        *ImComponents = GL_RGB8;
        break;
    case 4:
        //Format BGRA, odpowiada pamięci map bitowych Windows (DIB)
        *ImFormat = GL_BGRA_EXT;
        *ImComponents = GL_RGBA8;
        break;
    };
    fclose(pFile);
    return pbitsperpixel;
}
```

3.2 Funkcja MyInit

Funkcja MyInit została zaktualizowana o kod potrzebny do definicji kluczowych elementów teksturowania. Nowa część kodu to:

```
// 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("tekstura.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);
    //Zdefiniowanie tekstury 2-D, kolejne argumenty to:
    //target - Rodzaj definiowanej tekstury, zawsze powinno być GL_TEXTURE_2D
    //level - Poziom szczegółowości. Jest ustawiany na 0 gdy mipmapy nie są stosowane, gdy mipmapy są
używane level jest liczbą określająca poziom redukcji obrazu tekstury.
    //components - Ilość używanych składowych koloru, liczba od 1 do 4.
    //width - Szerokość obrazu tekstury, zawsze jest potegą liczby 2, może być powiększona o szerokość tak
zwanej ramki tekstury
    //height - Wysokość obrazu tekstury, zawsze jest potęgą liczby 2, może być powiększona o szerokość ramki
tekstury.
    //border - Szerokość ramki tekstury, może wynosić 0, 1 lub 2
    //format - Format danych obrazu tekstury, określa co opisują dane, czy są indeksami kolorów czy ich
składowymi (np. R, G, B) i w jakiej są podane kolejności.
    //type - Typ danych dla punktów obrazu tekstury, określa jak kodowane są dane, istnieje możliwość  używania
różnych formatów liczb, od 8 bitowych liczb stałoprzecinkowych, do 32 bitowych liczb zmiennoprzecinkowych.
    //piksels -  Tablica o rozmiarze  width x height x components, w ktorej znajdują się dane z obrazem tekstury.
    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);
    //Funkcja glTexEnvi() służy do ustalenia tak zwanego trybu teksturowania, czyli sposobu łączenia koloru
piksela obrazu tekstury z kolorem piksela ekranu.
```

```
   //GL_MODULATE - kolor piksela ekranu jest mnożony przez kolor piksela tekstury, następuje w ten sposób mieszanie barw tekstury i tego co jest teksturowane,
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   //Określenie sposobu nakładania tekstur
   //GL_TEXTURE_MIN_FILTER - Opisuje w jaki sposób następuje pominiejszanie tekstury (usuwanie pikseli)
   //GL_LINEAR - filtracja liniowa z odpowiednimi wagami
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

3.3 Funkcja pyramid

Funkcja pyramid służy do stworzenia teksturowanej piramidy.

```
bool side1 = true;
bool side2 = true;
bool side3 = true;
bool side4 = true;
bool side5 = true;

//Funkcja glTexCoord2f() poprzez swoje argumenty określa, które wierzchołki trójkąta naniesionego na wzorzec tekstury odpowiadają , którym wierzchołkom teksturowanego trójkąta.
void pyramid()
{
   glBegin(GL_TRIANGLES);

   if (side1)
   {
      glTexCoord2f(0.0f, 0.0f);
      glVertex3f(0.0f, 1.0f, 0.0f);

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

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

      glBegin(GL_TRIANGLES);


   }

   if (side2)
   {
      glTexCoord2f(0.0f, 0.0f);
      glVertex3f(0.0f, 1.0f, 0.0f);

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

      glTexCoord2f(1.0f, 0.0f);
      glVertex3f(1.0f, -1.0f, -1.0f);
   }

   if (side3)
   {
      glTexCoord2f(0.0f, 0.0f);
      glVertex3f(0.0f, 1.0f, 0.0f);
```

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

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


    }

    if (side4)
    {
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);

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

        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
    }


    glEnd();

    if (side5)
    {
        glBegin(GL_QUADS);

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

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

        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(1.0f, -1.0f, 1.0f);

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

        glEnd();
    }
}
```

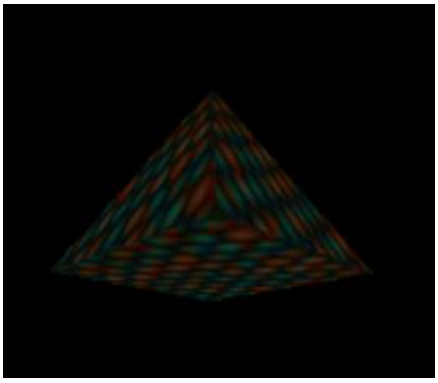
3.4 Funkcja Egg

Funkcja Egg wygląda analogicznie do tej z poprzednich ćwiczeń z wyjątkiem dodanych funkcji określających tekstury. Dodano je w taki sam sposób co w funkcji pyramid, dla każdego wierzchołka trójkąta tworzącego jajko, zdefiniowano kolejno wektor normalny, fragment tekstury oraz współrzędne wierzchołka. Dodatkowo w trakcie tworzenia punktów reprezentujących strukturę jajka, tworzone są również odpowiadające współrzędne tekstury w tablicy texturePoints.

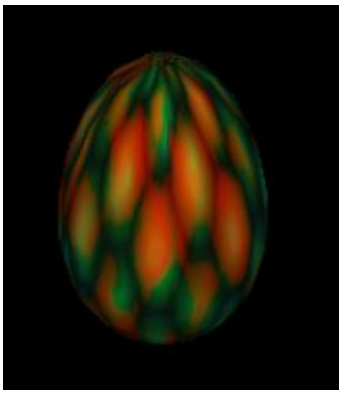
4. Wyniki

```
W celu modyfiakcji polozenia obserwatora nalezy uzyc:
LPM + ruch myszy - Obracanie dookola obiektu
PPM + ruch myszy - Przyblizanie/Oddalanie sie od obiektu
W celu modyfikacji modelu nalezy uzyc:
p - Wyswietlenie piramidy
e - Wyswietlenie jajka
1,2,3,4,5 - Pokazanie/ukrycie odpowiedniej sciany piramidy
```

Rysunek 1: Działanie programu w oknie konsoli



Rysunek 2: Działanie programu w oknie aplikacji, po wciśnięciu klawisza ‘p’



Rysunek 3: Domyślne działanie programu w oknie aplikacji

5. Wnioski

Dzięki dokumentacji oraz instrukcji do ćwiczenia, byłem w stanie stworzyć program teksturujący modele obiektów. Symulacja zwróciła oczekiwane efekty, program został wykonany poprawnie.

6. Źródła

Źródła były dostępne w czasie w dniu wykonywania ćwiczenia.

http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_6_dz/

http://www.paulbourke.net/dataformats/tga/

https://docs.microsoft.com/en-us/windows/win32/opengl/gleximage2d

https://community.khronos.org/t/what-gl-luminance-does/28618