Programowania w OpenGL z użyciem shader'ów, Texturowanie

1. Cel ćwiczenia:

Zapoznanie z funkcjami realizującymi wczytywanie tekstur oraz mechanizmami nakładania tekstur na wielokąty.

2. Wstęp:

W bibliotece OpenGL możliwe jest zastosowanie tekstur:

- 1D – (jednowymiarowa) jest to obraz nieposiadający wysokości lub szerokości. Są to obrazki w postaci paska o szerokości lub wysokości jednego piksela,
- 2D – (dwuwymiarowa) to obrazek szerszy i wyższy niż jeden piksel. Zwykle jest ładowana z pliku *.BMP. Tekstury 2D są używane w celu zastąpienia powierzchni o skomplikowanej geometrii, budynków, drzew itd. Tekstury 2D są używane także w celu stworzenia realistycznego tła.
- 3D – (trójwymiarowe – wolumetryczne) tekstury te są używane w różnych aplikacjach „skanerów" trzeciego wymiaru. Wadą tego typu tekstur jest duża wielkość w pamięci. Mała tekstura 256x256x256 pikseli w skali szarości zajmuje 16 MB pamięci.

3. Wprowadzenie:

Pierwszą rzeczą, którą trzeba wykonać, aby używać tekstur, jest załadowanie ich do aplikacji. Obrazy tekstur mogą być przechowywane w różnych formatach, każdy z własną strukturą i kolejnością danych. Jednym rozwiązaniem byłoby wybranie formatu pliku, na przykład .PNG i napisanie własnego programu ładującego obrazy. Innym rozwiązaniem jest użycie biblioteki ładującej obraz, która obsługuje kilka popularnych formatów np. **stb_image.h**

Przykładowa tekstura crate.bmp, metal.bmp, obraz.bmp, table.bmp, wall.bmp, wood.bmp

Jest to bardzo popularna biblioteka, która jest w stanie załadować najbardziej popularne formaty plików i jest łatwa do zintegrowania z projektem. Po pobraniu, plik nagłówkowy należy dodać do projektu i dołączyć do pliku cpp. Np.:

```
#define STB_IMAGE_IMPLEMENTATION

#include "stb_image.h"
```

Przykładowy kod ładujący plik graficzny i tworzący teksturę:

```
unsigned int texture1;      //Jak do wszystkich obiektów w OpenGL do tekstury można odwołać się poprzez
identyfikator ID                    // texture 1
glGenTextures(1, &texture1);        // Generuje tekstury
glBindTexture(GL_TEXTURE_2D, texture1);                 //Ustawienie tekstury jako bieżącej (powiązanie)
                                                        // set the texture wrapping parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
                                                    // set texture wrapping to GL_REPEAT (default wrapping method)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
                                                    // set texture filtering parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
                                                    // load image, create texture and generate mipmaps
int width, height, nrChannels;
stbi_set_flip_vertically_on_load(true); // tell stb_image.h to flip loaded texture's on the y-axis.
unsigned char *data = stbi_load("metal.jpg", &width, &height, &nrChannels, 0);
if (data)
{
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(data);
```

Użycie tekstury:

Aby tekstura mogła zostać prawidłowo nałożona na wielokąt konieczne jest podanie koordynat tekstury. Konieczna jest modyfikacja tablicy z danymi o wierzchołkach:

Przykładowo dla jednej ściany sześcianu:

GLfloat vertices[] = {

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f,0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f,0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f,1.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f,1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f,1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f,0.0f

};

Dwie ostatnie wartości w wierszach to koordynaty tekstury. Trzeba poinformować OpenGL o rozmiarze generowanego bufora:

glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat)*punkty_ * 8, vertices, GL_STATIC_DRAW);

Następnie aby była możliwość przekazania tych informacji do shadera konieczne jest określenie pozycji tych danych:

GLint TexCoord = glGetAttribLocation(shaderProgram, "aTexCoord");

glEnableVertexAttribArray(TexCoord);

glVertexAttribPointer(TexCoord, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));

Następnie konieczna jest zmiana kodu vertex shadera, aby przyjąć współrzędne tekstury jako atrybut wierzchołka, a następnie przekazać współrzędne do modułu vertex shader:

in vec2 aTexCoord;

out vec2 TexCoord;

A w funkcji main()

TexCoord = aTexCoord;

Fragment shader przyjmuje te dane jako wejściowe:

in vec2 TexCoord;

Do shadera trzeba przekazać jeszcze teksturę. Można to zrobić z użyciem tzw. samplera.

uniform sampler2D texture1;

Ustawienie kolorów fragmentów można uzyskać wywołaniem w funkcji main():

outColor=texture(texture1, TexCoord);

Ostatecznie przed wywołaniem funkcji rysującej trzeba ustawić teksturę na bieżącą:

glBindTexture(GL_TEXTURE_2D, texture1);

4. Ćwiczenie:

Należy przygotować projekt przedstawiający oteksturowany sześcian.