1 Odwzorowania tekstur

Tematem bieżących ćwiczeń będzie implementacja obsługi obiektu tekstury w OpenGL. W ogólności tekstura jest reprezentowana w postaci binarnego ciągu opisującego atrybuty kolejnych fragmentów teksturowanej powierzchni. Tekstury jednowymiarowe pozwalają na definiowanie wzorów kreskowania teksturowanej powierzchni, co przy odpowiednim doborze parametrów renderowania pozwala na uzyskanie efektu kreskówki [1]). Teksturowanie trójwymiarowe wykorzystuje obraz do utworzenia pofalowanej, trójwymiarowej powierzchni w oparciu o projekcję map wysokości. Najprostszym i najpopularniejszym sposobem na odwzorowanie tekstury na obiekcie jest teksturowanie dwuwymiarowe. W tej technice zachodzi „rozpięcie" dwywymiarowego, płaskiego obrazu na wierzchołkach czworokąta ograniczającego daną bryłę. Obecnie zrealizujemy sposób teksturowania dwuwymiarowego z użyciem pliku bmp. W tym celu zbudujemy krok po kroku kod odpowiedzialny za ładowanie tekstury z pliku mapy bitowej do postaci zasobu zarządzalnego z poziomu biblioteki *OpenGL*.

1.1 Utworzenie obiektu tekstury

Biblioteka *OpenGL* traktuje tekstury jako swoje wewnętrzne obiekty opatrzone unikalnymi identyfikatorami liczbowymi. Obiekt tekstury powstaje na skutek wywołania funkcji konwertującej, pracującej na zbiorze danych wejściowych tekstury pozyskiwanych zwykle z pliku dyskowego. Chcąc zatem utworzyć obiekt tekstury reprezentujący dwuwymiarową mapę bitową, należy odczytać wartości pikseli z pliku bmp i przekazać je do konstruktora obiektu tekstury. W bieżącej implementacji procedura tworząca obiekty tekstur wymaga podania plików BMP o głębi kolorów 24bit/piksel. Ponadto każdy z obrazów przekształcanych na teksturę winien mieć rozmiary będące naturalną potęgą liczby 2 (dotyczy to wszystkich grafik, nie tylko map bitowych).

1.1.1 Przebieg ćwiczenia

1. Utwórz nowy projekt programistyczny i podłącz do niego pliki załączone do niniejszej instrukcji
2. Skompiluj i uruchom projekt. Na bieżącym etapie winno powstać okno aplikacji z widokiem sceny trójwymiarowej zawierającym biały sześcian
3. Sprawdź możliwość obracania sześcianu (klawisze strzałkowe) oraz zmiany jego rozmiaru (klawisze F3,F4). W projekcie zachowano implementację modelu oświetlania Ambient+Diffuse. Zmiana intensywności oświetlania światłem ambient jest możliwa za pośrednictwem klawiszy F1 oraz F2.
4. Zapoznaj się z implementacją klasy `glTexture` zawartej w pliku `texture.cpp`
5. Uzupełnij kod konstruktora obiektu `glTexture` do następującej postaci.

```
char *ImgData = NULL;
int ImgWidth;
int ImgHeight;

// otworz plik BMP i zaladuj jego zawartosc do ImgData
LoadBMP(FileName,ImgWidth,ImgHeight,&ImgData);

// przygotuj id dla obiektu tekstury
glGenTextures(1, &TextureId);

// podlacz obiekt tekstury do pracy
glBindTexture(GL_TEXTURE_2D, TextureId);
```

[1] teksturowanie 1D jest nieodłącznie związane z techniką "kreskówkową"(ang. *tooning technique*

```
// zaladuj plik graficzny do obiektu tekstury
glTexImage2D(GL_TEXTURE_2D,
                        0,
                        GL_RGB,
                        ImgWidth,
                        ImgHeight,
                        0,
                        GL_BGR,
                        GL_UNSIGNED_BYTE,
                        ImgData);

// zwolnij pamiec zajeta przez plik graficzny
free(ImgData);

// przygotuj sampler tekstury
glGenSamplers(1, &SamplerId);

// ustaw parametry filtrowania
glSamplerParameteri(SamplerId, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glSamplerParameteri(SamplerId, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

if (TextureId)
       sprintf(_msg,"Zaladowano %s jako ID=%d",FileName,TextureId);
else
       sprintf(_msg,"Nie udalo sie utworzyc tekstury z %s",FileName);

 PrintLog(_msg);
```

Kod ten jest przykładem typowej implementacji tworzenia obiektu tekstury. Obiekt tekstury jest zdefiniowany poprzez swój indentyfikator `TextureID`. W danej chwili biblioteka pracować może tylko z jednym obiektem tekstury, który wybieramy poprzez polecenie `glBindTexture`. Z teksturą skojarzony jest tzw. sampler, którego zadaniem jest zapewnienie przenoszenia obrazu tekstury na powierzchnię renderowanego prymitywu. Na pracę samplera wpływają tzw. parametry filtracji tekstry, którymi zajmiemy się w dalszej części ćwiczeń.

6. Na podstawie dokumentacji internetowej polecenia `glTexImage2D` wyjaśnij przeznaczenie poszczególnych parametrów wywołania.
7. Skompiluj i uruchom program. W razie występowania błędów kompilacji zweryfikuj wprowadzony kod konstruktora klasy.
8. Wprowadź deklarację instancji o nazwie `Face` obiektu `glTexture` jako składową prywatną klasy `Scene`
9. W kodzie metody `PrepareObjects` utwórz instancję obiektu `Face` przed kodem budującym geometrię kostki `Cube`

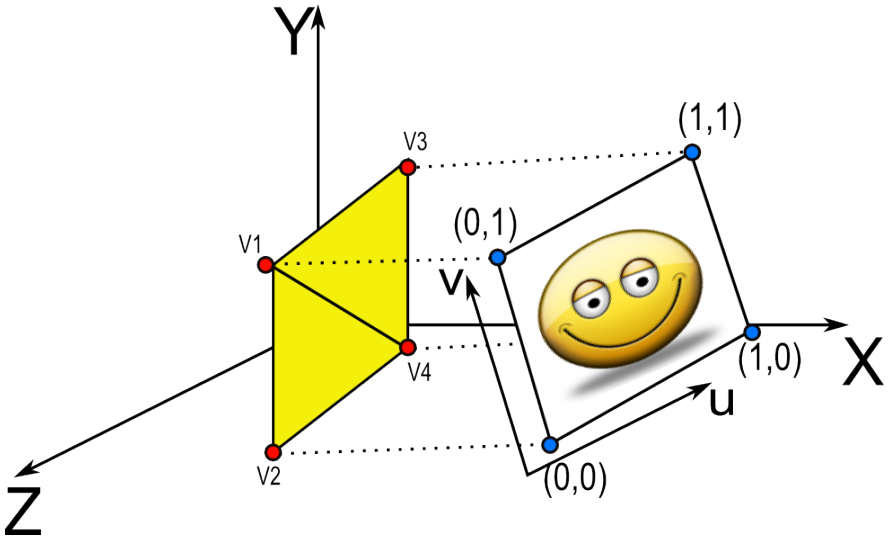
```
Cube = new glObject();
Face = new glTexture("textures\\happy.bmp");
```

10. Zadaniem powyższego kodu jest utworzenie obiektu tekstury na podstawie pliku graficznego `happy.bmp` zawartego w materiałach do ćwiczeń. Sprawdź położenie folderu z teksturami względem plików w projekcie programistycznym

11. Skompiluj i uruchom program. Sprawdź okno Log aplikacji, czy pojawił się tam komunikat o prawidłowym utworzeniu obiektu tekstury. Jaki identyfikator uzyskał obiekt tekstury?

1.2 Nakładanie tekstury

Do tej pory utworzyliśmy obiekt tekstury w OpenGL. Aby móc z niego korzystać, należy określić, w jaki sposób obraz tekstury ma być naklejony na powierzchnię prymitywów w przestrzeni sceny 3D. W tym celu należy rozbudować opis geometryczny obiektów o atrybuty wierzchołków powiązane z atrybutami tekstury. Każdy z utworzonych obiektów tekstury dwuwymiarowej posiada skojarzony dwuwymiarowy układ współrzędnych UV. Początek układu współrzędnych jest skojarzony z lewym, dolnym narożnikiem obrazu. Wartości współrzędnych tekstury są normalizowane, co oznacza, że prawy górny narożnik posiada współrzędne (u, v) równe $(1.0, 1.0)$. Proces nakładania tekstury na prymityw polega na powiązaniu z każdym wierzchołkiem prymitywu atrybutu określającego współrzędne punktu na obrazie tekstury, który ma ten wierzchołek przykrywać. Interpretację graficzną dla tego opisu zaprezentowano na rys. 1.



Rysunek 1: Interpretacja graficzna dla atrybutu współrzędnej tekstury

1.2.1 przebieg ćwiczenia

1. Zapoznaj się z implementacją klasy `glObject`. Zwróć uwagę na sposób zrealizowania metod `AddVertex` oraz `BeginObject`. W jaki sposób są dodawane współrzędne u, v tekstury do tablicy atrybutów obiektów? Jaki indeks atrybutu obiektów skojarzony jest ze współrzędną tekstury?

2. Uzupełnij metodę `BeginObject` tak, aby przy definiowaniu rodzaju prymitywu móc powiązać z nim obiekt tekstury, która ma być używana do rysowania tego prymitywu.

```
void glObject::BeginObject(GLenum P,GLuint TextureId)
{

       lVAO++;
       // przypisz rodzaj prymitywu do narysowania VAO
       Primitives[lVAO-1] = P;
```

```
        // przypisz Id tekstury do narysowania VAO
        Textures[lVAO-1] = TextureId;

 // dalszy kod
```

3. Uzupełnij metodę `Draw` w klasie `glObject` aby móc automatycznie przełączać obiekty tekstur przy rysowaniu. Polecenie `glEnable` oraz `glDisable` pozwala włączać/wyłączać proces teksturowania.

```
void glObject::Draw()
{
        for (int i = 0; i < lVAO; i++)
        {
                if (Textures[i] == 0)
                {
                        glDisable(GL_TEXTURE_2D);
                }
                        else
                {
                                glBindTexture(GL_TEXTURE_2D, Textures[i]);
                                glEnable(GL_TEXTURE_2D);
                }
                glBindVertexArray(VAO[i]);
                glDrawArrays(Primitives[i],0,lCoords[i]/3);
                glBindVertexArray(0);
        }
}
```

4. Uzupełnij kod shadera wierzchołków o dyrektywy pozwalające na pobieranie z tablic atrybutów współrzędnych tekstur.

```
#version 330

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inColor;
layout (location = 2) in vec3 inNormal;

layout (location = 3) in vec3 inTexCoord;

out vec3 vNormal;
smooth out vec4 kolorek;

out vec2 texCoord;

void main()
{
        gl_Position = projectionMatrix*modelViewMatrix*vec4(inPosition, 1.0);
```

```
        vec4 vRes = normalMatrix*vec4(inNormal, 0.0);
        vNormal = vRes.xyz;
        kolorek = vec4(inColor,1.0);
        texCoord = vec2(inTexCoord[0],inTexCoord[1]);
}
```

5. Uzupełnij kod shadera fragmentów o wykorzystanie zmiennej `texCoord` przekazywanej z shadera wierzchołków

```
#version 330

uniform vec3 LightColor;
uniform vec3 LightDirection;
uniform float LightAmbient;

in vec4 kolorek;
in vec3 vNormal;
in vec2 texCoord;
out vec4 outputColor;

uniform sampler2D gSampler;

void main()
{
        float LightDiffuse = max(0.0, dot(normalize(vNormal), -LightDirection));
        //outputColor = kolorek*vec4(LightColor*(LightAmbient+LightDiffuse),1.0);
        outputColor = texture2D(gSampler, texCoord);
}
```

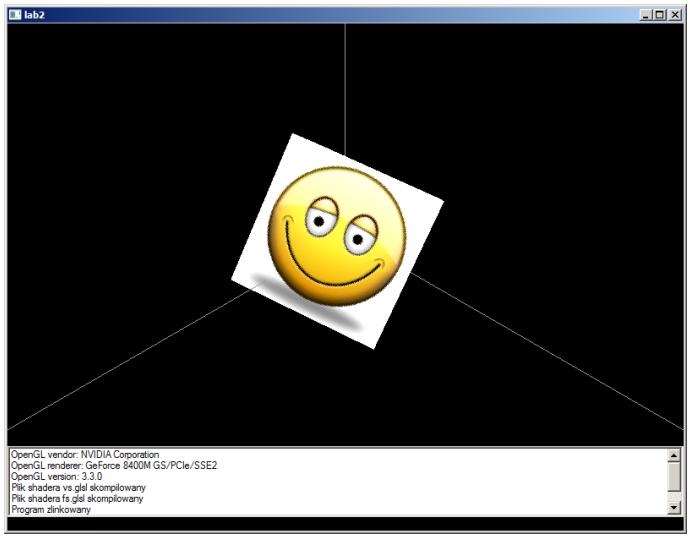
6. Shader fragmentów zwraca kolor fragmentu prymitywu przykrytego korespondującym fragmentem tekstury. Za proces pobierania koloru z właściwego miejsca w teksturze odpowiada sampler. Wywołuje się go przy użyciu polecenia glsl `texture2D`

7. Przypisz atrybuty współrzędnych tekstury do współrzędnych wierzchołków na jednej ze ścian sześcianu. Poniższy kod realizuje to zadanie dla ściany prostopadłej do osi OZ

```
// sciany prostopadle do OZ
        Cube->SetNormal(0.0,0.0,1.0);
        Cube->BeginObject(GL_TRIANGLE_STRIP,Face->Bind());
        Cube->AddVertex(-0.5,0.5,0.5,0.0,1.0);
        Cube->AddVertex(-0.5,-0.5,0.5,0.0,0.0);
        Cube->AddVertex(0.5,0.5,0.5,1.0,1.0);
        Cube->AddVertex(0.5,-0.5,0.5,1.0,0.0);
        Cube->EndObject();
```

   Skompiluj i uruchom program. Na tym etapie aplikacja powinna wytworzyć obraz zbliżony do tego z rys.2. W razie kłopotów z kompilacją lub uruchomieniem prześledź wykonanie powyższych kroków.

8. Na podstawie kodu narysuj na papierze opisywaną ścianę w rzucie 3D i oznacz kolejność zdefiniowania wierzchołków. Obok rzutu ściany narysuj kwadrat symbolizujący obraz tekstury i zaznacz na jego wierzchołkach kolejność przypisania współrzędnych tekstury do współrzędnych wierzchołków.



Rysunek 2: Przykład realizacji teksturowania kostki

9. Zmień kolejność przypisania współrzędnych tekstury, aby obrazek pojawił się do góry nogami
10. Sprawdź doświadczalnie, jak zachowa się proces teksturowania przy podaniu ułamkowych wartości współrzędnych tekstury, a jak przy podaniu wartości powyżej 1.0
11. Wyrenderuj widok kostki, z czterema buźkami
12. Rozbuduj kod aplikacji tak, aby na sąsiadujących ze sobą ścianach widniały dwie różne tekstury
13. Zwróć uwagę na sposób renderowania osi układu współrzędnych. Po wprowadzeniu teksturowania utraciły one swoje pierwotne kolory. Jaka jest przyczyna tego efektu? Rozbuduj kod shadera fragmentów, aby przy rysowaniu obiektów uwzględniał zarówno kolor prymitywu, kolor tekstury jak i modyfikację koloru wynikającą z istniejącego modelu oświetlenia.

1.3 Filtrowanie tekstur

Proces nakładania koloru tekstury na obszar prymitywu jest optymalny w sytuacji, gdy rozmiar teksturowanego obszaru odpowiada rozmiarowi obrazka tekstury. Bardzo często zdarza się jednak, że rozmiary wymienionych obiektów różnią się drastycznie od siebie. W takiej sytuacji zachodzi konieczność interpolowania lub ekstrapolowania pikseli pobieranych z tekstury. Jest to jeden z aspektów procesu filtrowania tekstury, który odpowiada za zrealizowanie poprawnego odwzorowania widoku obrazu na obszarze prymitywu. Obecnie porównamy dwa najpopularniejsze rodzaje filtracji w kontekście operacji powiększania i pomniejszania obrazu tekstury.

1.3.1 Przebieg ćwiczenia

1. Uruchom aplikację z widokiem poteksturowanej kostki. Powiększ rozmiar kostki klawiszem F3 aby móc zaobserwować szczegóły krawędzi na teksturze. Zachowaj widok sceny (zrób zrzut ekranowy)
2. Zmodyfikuj algorytm filtrowania użyty w teksturowaniu. W tym celu zmodyfikuj kod konstruktora klasy `glTexture`

```
// ustaw parametry filtrowania
glSamplerParameteri(SamplerId, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(SamplerId, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

3. Ponownie uruchom program. Porównaj efekt renderowania krawędzi tekstury w powiększonej kostce z zarejestrowanym uprzednio zrzutem.

1.4 Animacja na scenie

Proces renderowania widoku sceny zachodził dotychczas w odpowiedzi na konkretne zdarzenie systemowe związane np. z naciśnięciem klawisza klawiatury lub koniecznością odrysowania zawartości okna aplikacji. Tymczasem możliwe jest wprowadzenie do aplikacji obiektu timera, który będzie cyklicznie wywoływał proces odrysowania zawartości sceny. To podejście w połączeniu z kodem modyfikującym nieco zawartość sceny przy każdorazowym odrysowaniu daje w konsekwencji efekt animacji. Obecnie przećwiczymy proces dodawania do projektu zasobu timera systemowego i obsługę związanego z tym komunikatu.

1.4.1 Przebieg ćwiczenia

1. Otwórz kod pliku `main.cpp` implementujący współpracę aplikacji z systemem operacyjnym.
2. Wprowadź definicję identyfikatora dla timera systemowego

```
//---------------------------------
// ZMIENNE GLOBALNE
//---------------------------------
int window; // uchwyt do okna OGL
Scene *SC; // scena OpenGL

#define IDT_TIMER WM_USER + 200
```

3. Rozbuduj kod punktu wejścia do aplikacji WinMain o procedury tworzące i zwalniające obiekt timera

```
try
	{
		SC = new Scene(wWidth,wHeight);

		// utworz okno z widokiem sceny OPENGL
		if (!CreateGLWindow(PROJECT_NAME,wWidth,wHeight,16))
			return 0;

		// ustaw obiekt timera
		SetTimer(hWnd,IDT_TIMER,10,(TIMERPROC) NULL);

		// pobierz komunikat z kolejki systemowej
		while(GetMessage( &msg, NULL, 0, 0))
		{
			// przetwarzaj komunikat w obszarze okna
			TranslateMessage(&msg);
			// usun komunikat z kolejki systemowej
			DispatchMessage(&msg);
```

```
                }

        }
        catch (char *e)
        {
                        MessageBox(0,e,"Wystapil wyjatek",0);
        }
        catch(...) // unhandled exceptions
        {
                        KillGLWindow(); // usun zasoby okna
                        if (SC) delete SC;
        }
        // usun zasob timera
        KillTimer(hWnd,IDT_TIMER);
        KillGLWindow(); // usun zasoby okna
```

4. Rozbuduj obsługę komunikatu `WM_TIMER` w kodzie `main.cpp`

```
case WM_PAINT: // odrysowanie okna
                {
                         SC->Draw();
                         SwapBuffers(hDC);
                        break;
                }

                case WM_TIMER: // zdarzenie timera
                {
                        SC->Animate();
                        SC->Draw();
                        SwapBuffers(hDC);
                        break;
                }
```

5. Skompiluj i uruchom program. Sprawdź treść metody `Animate` zaimplementowanej w klasie `Scene`