UNIWERSYTET
TECHNOLOGICZNO-HUMANISTYCZNY
im. Kazimierza Pułaskiego w Radomiu

WYDZIAŁ TRANSPORTU, ELEKTROTECHNIKI I INFORMATYKI

Kierunek: INFORMATYKA

Przedmiot: BIBLIOTEKA GRAFICZNA OPENGL

Autor: DR ARTUR HERMANOWICZ

Tekstury

1. Ładowanie i mapowanie

Gładkie powierzchnie obiektów wyglądają mało realistycznie. W celu uczynienia ich powierzchni ciekawszymi dla oka można posłużyć się teksturami. Tekstura jest fragmentem bitmapy, którą nakładamy na powierzchnię obiektów. Sterując samym kolorem i oświetleniem ciężko uzyskać efekt, który daje nałożenie gotowego obrazu. Tekstury umożliwiają również zwiększenie wydajności aplikacji poprzez zastąpienie skomplikowanych obiektów ich obrazem.

Tekstury mogą być jedno-, dwu- lub trójwymiarowe. Tekstury jednowymiarowe głównie służą do uzyskania np. wielokolorowej linii. Tekstury dwuwymiarowe zazwyczaj służą jako mapa kolorów nakładanych na powierzchnie figur. Tekstury trójwymiarowe umożliwiają tworzenie map przestrzennych służących np. do tworzenia realistycznych obłoków, dymu itp. W dalszej części będziemy zajmować się tylko teksturami dwuwymiarowymi.

Oprócz zastosowania jako mapa kolorów tekstury mogą być wykorzystywane do innych celów jak np. mapy przezroczystości, mapy wektorów normalnych itp.

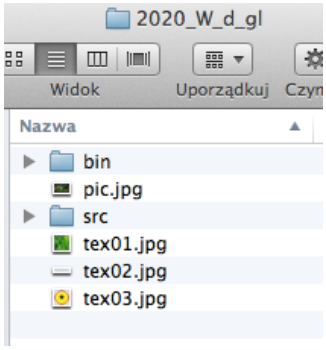
W najprostszym ujęciu tekstura będzie prostokątnym fragmentem bitmapy przechowującym dane koloru. Punkty tekstury nazywamy tekselami. Pamiętać należy, że w praktycznym zastosowaniu teksel nie będzie miał nic wspólnego z pikselem. Przykładowo tekstura o wymiarach 1 na 1 teksel nałożona na prostokąt o wymiarach ekranu spowoduje, że każdy piksel na ekranie otrzyma kolor jednego i tego samego teksela.

Współrzędne tekstur liczone są od lewego dolnego rogu bitmapy – punkt (0, 0). Oś pozioma nazywa się **s**, pionowa **t** (Rys. 1).

Rys. 1. Układ współrzędnych tekstury 2D

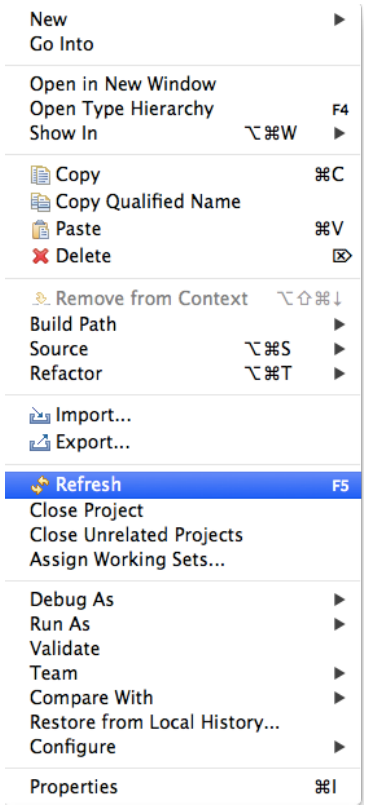
Pierwszą czynnością, którą musimy wykonać aby posługiwać się teksturami jest pozyskanie odpowiednich obrazów. Przykładowe pliki (tekstury.zip) należy skopiować do

katalogu głównego projektu. Pliki te oczywiście można załadować z dowolnego miejsca ale wymaga to podania odpowiedniej ścieżki dostępu. W celu uproszczenia przykładu znajdują się one w katalogu bieżącym aplikacji. Katalog projektu w tym przykładzie powinien wyglądać następująco:



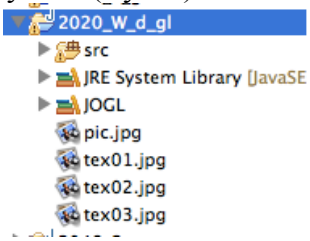
Rys. 2. Katalog projektu z teksturami

Jeżeli w Eclipse odświeżymy (Rys. 3) zawartość projektu pliki powinny ukazać się w zasobach.



Rys. 3. Opcja Refresh projektu

Jeżeli wszystko poszło pomyślnie (Rys. 4) można zacząć eksperymenty.



Rys. 4. Widok projektu z teksturami

W celu posługiwania się teksturami muszą mieć one zdefiniowane odpowiednie identyfikatory. Są to liczby całkowite (int). W naszym przykładzie (plik p07.java) wykorzystane są dwie tekstury. Dlatego w sekcji pól klasy wstawiamy:

```
private int texture,texture1;
```

Zezwolenie na wykorzystanie tekstur znajduje się w metodzie **Init**, jak również ich załadowanie z plików (w tym programie nie planów aby obrazy zmieniały się w trakcie wykonywania):

```
gl.glEnable(GL2.GL_TEXTURE_2D);
try{
     File f=new File("pic.jpg");
     Texture t=TextureIO.newTexture(f, true);
     texture=t.getTextureObject(gl);
}catch(IOException e){
     e.printStackTrace();
}
try{
     File f=new File("tex01.jpg");
     Texture t=TextureIO.newTexture(f, true);
     texture1=t.getTextureObject(gl);
}catch(IOException e){
     e.printStackTrace();
}
```

W JOGL jest to niezwykle prosta sprawa: Wystarczy posłużyć się metodą **newTexture** klasy **TextureIO**. Po załadowaniu tekstury jej identyfikator pobieramy metodą **getTextureObject**. Ze względu na możliwość wystąpienia błędów podczas dostępu do plików ładowanie należy przeprowadzić z obsługą wyjątków.

Mając załadowane tekstury „przypinamy" je do obiektów metodą **glBindTexture**. Dla każdego wierzchołka należy przypisać odpowiednie koordynaty tekstury metodą **glTexCoord**. I to już wystarczy (na początek) do zabawy teksturami.

W naszym przykładzie (Rys. 5 – p07.java) pierwsza tekstura (**texture** – pic.jpg) służy do zrobienia tła, druga (texture1 – tex01.jpg) wykorzystana jest do pokrycia obracającego się sześcianu.

Rys. 5. Okno programu p07

Tworząc tło warto pamiętać o dwóch rzeczach. Po pierwsze zwykle wygląda niekorzystnie jeżeli jest oświetlone. Dlatego przed rysowaniem tła wyłączamy oświetlenie, potem je włączamy. Druga sprawa to bufor głębi. Tło nie powinno wchodzić w interakcję

z obiektami sceny. Dlatego przed jego rysowaniem wyłączamy obsługę bufora głębi, a na zakończenie rysowania tła przywracamy. Stąd rysowanie tła znajduje się (metoda **Display**) między zestawami instrukcji:

```
gl.glDisable(GL2.GL_DEPTH_TEST);
gl.glDisable(GL2.GL_LIGHTING);
…
gl.glEnable(GL2.GL_LIGHTING);
gl.glEnable(GL2.GL_DEPTH_TEST);
```

Powstaje problem jak narysować tło tak, aby niezależnie od punktu widzenia, ruchów kamery itp. zawsze zajmowało cały obszar okna. No jak obliczyć wymiary w 3D tego prostokąta… Rozwiązanie jest niezwykle proste. Wystarczy na czas rysowania tła przełączyć się w tryb rzutowania równoległy (domyślny w OpenGL). Wtedy po pierwsze odpada nam problem głębokości, po drugie współrzędne krawędzi okna mają wartości +-1.

Przełączamy się w tryb macierzy rzutowania. Odkładamy stan na stosie – resztę dzieła chcemy mieć namalowaną w rzutowaniu perspektywicznym. Przywrócenia rzutowania równoległego jest proste – wystarczy zresetować macierz metodą **glLoadIdentity** (rzutowanie równoległe jest domyślne w OpenGL). Przełączamy znów do macierzy modelowania, reset i możemy malować tło.

```
gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glPushMatrix();
gl.glLoadIdentity();
gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();
```

Po namalowaniu tła przywrócenie ustawień perspektywy to ponownie przełączenie trybu macierzy i zdjęcie ze stosu wcześniej zapamiętanych ustawień dla perspektywy:

```
gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glPopMatrix();
gl.glMatrixMode(GL2.GL_MODELVIEW);
```

W samym środku tego kodu natomiast znajduje się narysowanie prostokąta z odpowiednią teksturą:

```
gl.glBindTexture(GL2.GL_TEXTURE_2D, texture);
gl.glBegin(GL2.GL_QUADS);
gl.glTexCoord2f(0.0f,0.0f); gl.glVertex3f(-1.0f,-1.0f,-1.0f);
gl.glTexCoord2f(1.0f,0.0f); gl.glVertex3f( 1.0f,-1.0f,-1.0f);
gl.glTexCoord2f(1.0f,1.0f); gl.glVertex3f( 1.0f, 1.0f,-1.0f);
gl.glTexCoord2f(0.0f,1.0f); gl.glVertex3f(-1.0f, 1.0f,-1.0f);
gl.glEnd();
```

Pierwszą czynnością jest przypięcie odpowiedniej tekstury metodą **glBindTexture**. Samo tło to zwykły prostokąt, a nawet kwadrat. W tym trybie rzutowania wszystkie współrzędne krawędzi to +-1. Przed każdym wierzchołkiem musi znajdować się wywołanie metody **glTexCoord**, która przypisuje mu odpowiednie koordynaty.

Pierwszy wierzchołek (lewy dolny róg) ma być tożsamy z lewym dolnym rogiem tekstury, stąd współrzędne (0, 0). Następny wierzchołek (prawy dolny róg) ma być tożsamy z prawym dolnym rogiem tekstury, stąd współrzędne (1, 0). Analogicznie z pozostałymi wierzchołkami.

Możemy teraz przystąpić do namalowania obracającego się sześcianu. Po zestawie obrotów względem każdej osi, przypinamy teksturę, którą będzie pokryta każda ścianka sześcianu:

```
gl.glBindTexture(GL2.GL_TEXTURE_2D, texture1);
```

Sam sześcian utworzony jest z jednego czworokąta. Odpowiednio obracając jedną ścianką uzyskujemy cały sześcian:

```
for(int i=0;i<6;i++){
      gl.glBegin(GL2.GL_QUADS);
      gl.glNormal3f(0.0f, 0.0f, 1.0f);
      gl.glTexCoord2f(0.0f,0.0f); gl.glVertex3f(-1.0f,-1.0f,1.0f);
      gl.glTexCoord2f(1.0f,0.0f); gl.glVertex3f( 1.0f,-1.0f,1.0f);
      gl.glTexCoord2f(1.0f,1.0f); gl.glVertex3f( 1.0f, 1.0f,1.0f);
      gl.glTexCoord2f(0.0f,1.0f); gl.glVertex3f(-1.0f, 1.0f,1.0f);
      gl.glEnd();
      if(i%2==0) gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
      else gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
}
```

Wektor normalny ustawiony jest w celu uzyskania odpowiedniego oświetlenia. W przypadku zastosowania tekstur oświetlenie jest zwykle niepotrzebne. W tym wypadku na niekorzyść sześcianu działa jeszcze to, że jego ścianki są całymi kwadratami – bez przekrojów zapewniających lepszy efekt oświetlenia.