Tekstury w OpenGL

April 16, 2024

1 Introduction

Celem jest stosowanie tekstur używając funkcje biblioteki OpenGL.

Obiekty 3D o jednolitym kolorze wyglądają ładnie, ale są ciekawe. Ich jednolite kolory nie mają wyglądu, powiedzmy, ceglanego muru lub kratki. Trójwymiarowe obiekty mogą być bardziej interesujące i bardziej realistyczne dzięki dodaniu tekstury do ich powierzchni. Tekstura jest generalnie odmianą od piksela do piksela w pojedynczym prymitywie. Rozważymy tylko jeden rodzaj tekstury: tekstury obrazu. Teksturę obrazu można zastosować do powierzchni, aby kolor powierzchni zmieniał się z punktu na punkt, podobnie jak malowanie kopii obrazu na powierzchni. Oto zdjęcie, które pokazuje sześć obiektów z różnymi teksturami obrazu (patrz Fig.1).

Tekstury mogą być najbardziej skomplikowaną częścią OpenGL i są częścią, która przetrwała i stała się bardziej skomplikowana w najnowocześniejszych wersjach, ponieważ są one tak istotne dla efektywnego tworzenia realistycznych obrazów. Tu używamy tylko część interfejsu API tekstury OpenGL.

Zauważ, że obraz używany jako tekstura powinien mieć szerokość i wysokość, które są potęgami 2, naprzykład 128, 256 lub 512. Jest to wymaganie w OpenGL 1.1. W niektórych wersjach wymaganie może być złagodzone, ale nadal dobrym pomysłem jest użycie tekstur z rozmiarami potęgi 2. Niektóre rzeczy, omówione tutaj, nie będą działać z teksturami innymi niż potęgi 2, nawet w nowoczesnych systemach.

Gdy tekstura obrazu zostanie zastosowana na powierzchni, domyślnym zachowaniem jest mnożenie składowych kolorów RGBA pikseli na powierzchni przez składniki kolorów z obrazu. Kolor powierzchni zostanie zmodyfikowany przez efekty świetlne, jeśli oświetlenie jest włączone, zanim zostanie pomnożone przez kolor tekstury. Często używa się **bieli jako koloru powierzchni**. Jeśli na powierzchni zostanie użyty inny kolor, doda „odcień" do koloru z obrazu tekstury.

2 Współrzędne tekstury

Gdy tekstura jest nakładana na powierzchnię, każdy punkt na powierzchni musi odpowiadać punktowi w teksturze. Musi istnieć sposób na określenie sposobu

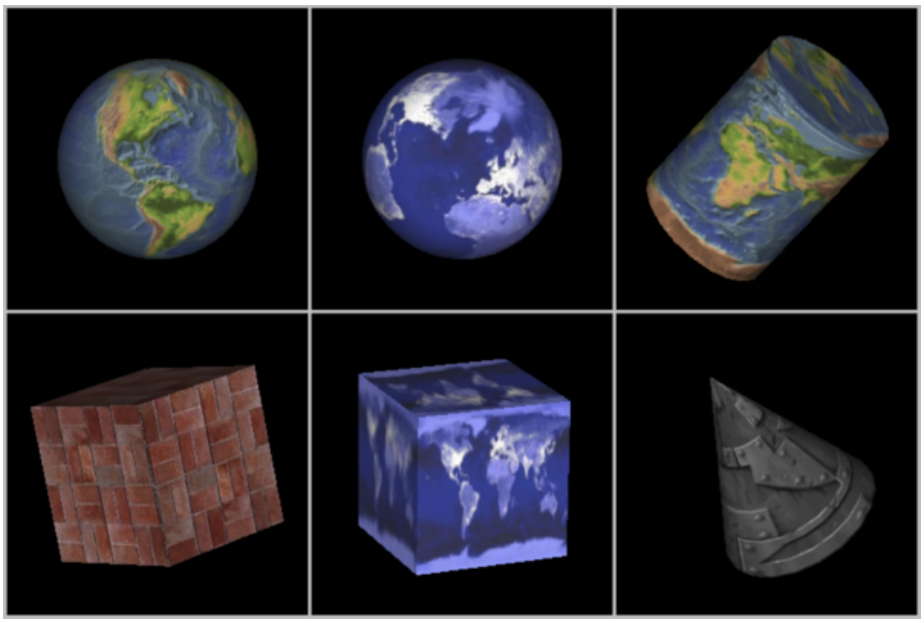

Figure 1: Przykłady użycia tekstur

obliczania tego odwzorowania. W tym celu obiekt potrzebuje współrzędnych tekstury. Jak zwykle w OpenGL, **współrzędne tekstury są określone dla każdego wierzchołka prymitywu**. Współrzędne tekstury dla punktów wewnątrz prymitywu są obliczane przez interpolację wartości z wierzchołków prymitywu.

Obraz tekstury ma własny układ współrzędnych 2D. Tradycyjnie s używane dla współrzędnej poziomej na obrazie i t jest używane dla współrzędnej pionowej. Współrzędna s to liczba rzeczywista, która zmienia się od 0 po lewej stronie obrazu do 1 po prawej stronie, podczas gdy t zmienia się od 0 na dole do 1 na górze. Wartości s lub t poza zakresem od 0 do 1 nie znajdują się wewnątrz obrazu, ale takie wartości są nadal ważne jako współrzędne tekstury. Zauważ, że współrzędne tekstury nie są oparte na pikselach. Bez względu na rozmiar obrazu, wartości s i t między 0 a 1 pokrywają cały obraz.

Aby narysować prymityw teksturowany, potrzebujemy pary liczb (s, t) dla każdego wierzchołka. Są to współrzędne tekstury dla tego wierzchołka. One wskazują, który punkt obrazu jest odwzorowany na wierzchołku. Załóżmy na przykład, że chcemy zastosować część obrazu *EarthAtNight* do prymitywu trójkąta. Powiedzmy, że obszar na obrazie, który ma być odwzorowany na prymityw to trójkąt pokazany tutaj, zaznaczony grubą pomarańczą:

Wierzchołki tego obszaru mają współrzędne (s, t) $(0.3, 0.1)$, $(0.45, 0.6)$ i $(0.25, 0.7)$. Te współrzędne z obrazu powinny być używane jako współrzędne tekstury dla wierzchołków prymitywu trójkąta.

Współrzędne tekstury wierzchołka są atrybutem wierzchołka, podobnie jak kolor, normalne wektory i właściwości materiału. Współrzędne tekstury są

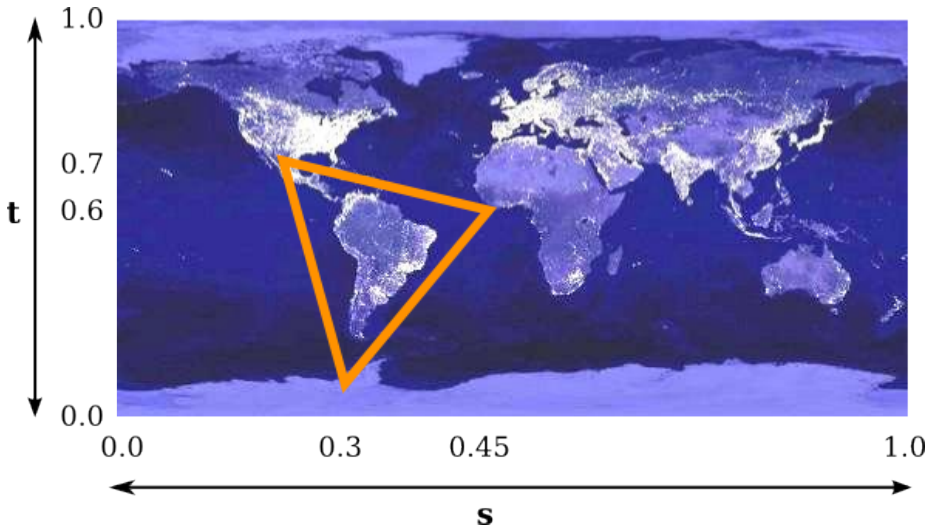

Figure 2: Współrzędne tekstury

określone przez rodzinę funkcji `glTexCoord *`, w tym funkcje `glTexCoord2f (s, t)`, `glTexCoord2d (s, t)`, `glTexCoord2fv (tablica)` i `glTexCoord2dv (tablica)`. Stan OpenGL zawiera bieżący zestaw współrzędnych tekstury, określony przez te funkcje. Gdy określisz wierzchołek za pomocą `glVertex *`, bieżące współrzędne tekstury zostaną skopiowane i staną się atrybutem powiązanym z wierzchołkiem. Jak zwykle oznacza to, że współrzędne tekstury dla wierzchołka muszą być określone przed wywołaniem `glVertex *`. Każdy wierzchołek prymitywu będzie wymagał innego zestawu współrzędnych tekstury.

Na przykład, aby zastosować trójkątny obszar na obrazie Fig. 2 do trójkąta w płaszczyźnie xyz z wierzchołkami $(0,0)$, $(0,1)$ i $(1,0)$:

```
glNormal3d(0,0,1);       // This normal works for all three vertices.
glBegin(GL_TRIANGLES);
glTexCoord2d(0.3,0.1);   // Texture coords for vertex (0,0)
glVertex2d(0,0);
glTexCoord2d(0.45,0.6);  // Texture coords for vertex (0,1)
glVertex2d(0,1);
glTexCoord2d(0.25,0.7);  // Texture coords for vertex (1,0)
glVertex2d(1,0);
glEnd();
```

Zauważ, że nie ma szczególnej zależności między współrzędnymi (x, y) wierzchołka, które dają jego pozycję w przestrzeni, oraz współrzędne tekstury (s, t) związane z wierzchołkiem. W rzeczywistości w tym przypadku trójkąt, który rysujemy, ma inny kształt niż trójkątny obszar na obrazie i ten fragment obrazu będzie musiał zostać rozciągnięty i zniekształcony, aby pasował. Takie zniekształcenia występują w większości zastosowań obrazów tekstur.

Czasami trudno jest zdecydować, jakich współrzędnych tekstury użyć. Jeden przypadek, w którym łatwo jest zastosować całą teksturę do prostokąta. Oto segment kodu, który rysuje kwadrat na płaszczyźnie xy z odpowiednimi współrzędnymi tekstury, aby odwzorować cały obraz na kwadrat:

```
glBegin(GL_TRIANGLE_FAN);
glNormal3f(0,0,1);
glTexCoord2d(0,0);     // Texture coords for lower left corner
glVertex2d(-0.5,-0.5);
glTexCoord2d(1,0);     // Texture coords for lower right corner
glVertex2d(0.5,-0.5);
glTexCoord2d(1,1);     // Texture coords for upper right corner
glVertex2d(0.5,0.5);
glTexCoord2d(0,1);     // Texture coords for upper left corner
glVertex2d(-0.5,0.5);
glEnd();
```

3 MipMapy i filtrowanie

Gdy tekstura jest nakładana na powierzchnię, piksele w teksturze zazwyczaj nie pasują do pojedynczych pikseli z powierzchni, i ogólnie tekstura musi być rozciągnięta lub skurczona podczas mapowania na powierzchni. Czasami kilka pikseli tekstury będzie mapowanych na ten sam piksel na powierzchni. W tym przypadku kolor zastosowany do piksela powierzchni musi w jakiś sposób zostać obliczony na podstawie kolorów wszystkich pikseli tekstury, które są do niego przypisane. To jest przykład **„filtrowania”**; w szczególności wykorzystuje **filtr minifikacji**, ponieważ tekstura jest zmniejszana. Gdy jeden piksel z tekstury pokrywa więcej niż jeden piksel na powierzchni, tekstura musi zostać powiększona, a my potrzebujemy **filtra powiększenia**.

Jeszcze jedna terminologia, zanim przejdziemy dalej: Piksele w teksturze są określane jako **teksele**, skrót od „piksel tekstury” lub „element tekstury”, i od teraz będę używać tego terminu.

Decydując o tym, jak zastosować teksturę do punktu na powierzchni, OpenGL zna współrzędne tekstury dla tego punktu. Te współrzędne tekstury odpowiadają jednemu punktowi tekstury, a ten punkt leży w jednym z tekseli tekstury. Najłatwiej jest zastosować kolor tego teksela do punktu na powierzchni. Nazywa się to **„filtrowaniem najbliższego texela”**. Jest bardzo szybki, ale zazwyczaj nie daje dobrych wyników. Nie bierze pod uwagę różnicy w rozmiarze między pikselami na powierzchni a teksturami w obrazie. Ulepszeniem filtrowania najbliższego teksela jest **„filtrowanie liniowe”**, które może przyjąć średnią kilku kolorów tekselowych, aby obliczyć kolor, który zostanie zastosowany na powierzchni.

Problem z filtrowaniem liniowym polega na tym, że będzie on bardzo nieefektywny, gdy na znacznie mniejszą powierzchnię zostanie nałożona duża tekstura. W tym przypadku wiele texeli mapuje się na jeden piksel, a obliczenie średniej z

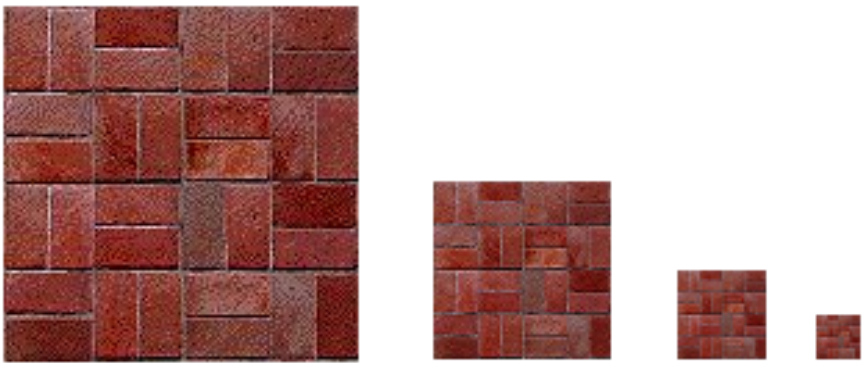

Figure 3: Mipmapy cegły

tak wielu tekseli staje się bardzo nieefektywne. Jest na to świetne rozwiązanie: **mipmapy**.

Mipmap dla tekstury jest zmniejszoną wersją tej tekstury. Kompletny zestaw mipmap składa się z pełnowymiarowej tekstury, wersji o połowie rozmiaru, w której każdy wymiar jest podzielony przez dwa, wersji o jednej czwartej, wersji o jednej ósmej i tak dalej. Jeśli jeden wymiar kurczy się do pojedynczego piksela, nie jest on dalej redukowany, ale drugi wymiar będzie nadal cięty na pół, aż osiągnie jeden piksel. W każdym razie ostateczna mipmapa składa się z pojedynczego piksela. Oto kilka pierwszych obrazów w zestawie mipmap dla tekstury cegły:

Mipmapy bardzo szybko stają się małe. Całkowita pamięć używana przez zestaw mipmap jest tylko o jedną trzecią większa niż pamięć używana dla oryginalnej tekstury, więc dodatkowe wymagania dotyczące pamięci nie stanowią dużego problemu podczas używania mipmap.

Mipmapy są używane tylko do filtrowania minifikacji. Aby teksturować piksel, OpenGL może najpierw wybrać mipmapę, której tekstury najlepiej pasują do rozmiaru piksela. Następnie może wykonać filtrowanie liniowe na tej mipmapie, aby obliczyć kolor, i będzie musiał uśrednić co najwyżej kilka tekseli, aby to zrobić.

W nowszych wersjach OpenGL ma możliwości do automatycznego generowania mipmap. W OpenGL 1.1, jeśli chcesz używać mipmap, musisz albo załadować każdą mipmapę indywidualnie, albo wygenerować ją samodzielnie. (Biblioteka GLU ma metodę `gluBuild2DMipmaps`, której można użyć do wygenerowania zestawu mipmap dla tekstury 2D).

4 Docelowe tekstury i parametry tekstury

OpenGL może faktycznie używać tekstur jednowymiarowych i trójwymiarowych, a także dwuwymiarowych. Z tego powodu wiele funkcji OpenGL związanych

z teksturami traktuje teksturę docelową jako parametr, aby stwierdzić, czy funkcja ma być stosowana do tekstur jedno, dwu lub trójwymiarowych. Dla nas jedyną teksturą docelową będzie `GL_TEXTURE_2D`.

Istnieje wiele opcji, które odnoszą się do tekstur, aby kontrolować szczegóły dotyczące nakładania tekstur na powierzchnie. Niektóre opcje można ustawić za pomocą funkcji `glTexParameteri ()`, w tym dwóch związanych z filtrowaniem. OpenGL obsługuje kilka różnych technik filtrowania dla minifikacji i powiększenia. Filtry można ustawić za pomocą `glTexParameteri ()`:

```
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, magFilter);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minFilter);
```

Wartości `magFilter` i `minFilter` są stałymi, które określają algorytm filtrowania. Dla `magFilter` jedynymi opcjami są `GL_NEAREST` i `GL_LINEAR`, dając najbliższy texel i liniowe filtrowanie. Domyślnym filtrem MAG jest `GL_LINEAR` i rzadko trzeba go zmieniać. Dla `minFiltera`, oprócz `GL_NEAREST` i `GL_LINEAR`, istnieją cztery opcje, które używają mipmap do efektywniejszego filtrowania. Domyślnym filtrem MIN jest `GL_NEAREST_MIPMAP_LINEAR`, który uśrednia między mipmapami a filtrowaniem najbliższego texela w obrębie każdej mipmapy. Aby uzyskać jeszcze lepsze wyniki, kosztem większej nieefektywności, możesz użyć `GL_LINEAR_MIPMAP_LINEAR`, który wykonuje uśrednianie zarówno pomiędzy, jak i wewnątrz mipmap. Pozostałe dwie opcje to `GL_NEAREST_MIPMAP_NEAREST` i `GL_LINEAR_MIPMAP_NEAREST`.

Jedna bardzo ważna uwaga: jeśli nie używasz mipmap dla tekstury, konieczne jest, aby zmienić filtr minifikacji dla tej tekstury na `GL_LINEAR` lub, mniej prawdopodobne, `GL_NEAREST`. Domyślny filtr MIN wymaga mipmap, a jeśli mipmapy nie są dostępne, tekstura jest uważana za nieprawidłowo uformowaną, a OpenGL ją ignoruje! Pamiętaj, że jeśli nie utworzysz mipmap i nie zmienisz filtra minifikacji, twoja tekstura będzie po prostu ignorowana przez OpenGL.

Jest jeszcze jedna para parametrów tekstury, która kontroluje sposób traktowania współrzędnych tekstury poza zakresem od 0 do 1. Jak wspomniano powyżej, domyślnie powtarzana jest tekstura. Alternatywą jest „zacisnąć" teksturę. Oznacza to, że gdy określone są współrzędne tekstury poza zakresem od 0 do 1, wartości te są wymuszane w tym zakresie: Wartości mniejsze niż 0 są zastępowane przez 0, a wartości większe niż 1 są zastępowane przez 1. Wartości można zaciskać oddzielnie w s i t kierunkach za pomocą

```
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Przekazanie `GL_REPEAT` jako ostatniego parametru przywraca domyślne zachowanie. Gdy działa zacisk, współrzędne tekstury poza zakresem 0 do 1 zwracają ten sam kolor, co teksel, który leży wzdłuż zewnętrznej krawędzi obrazu. Oto jak wygląda efekt na dwóch teksturowanych kwadratach:

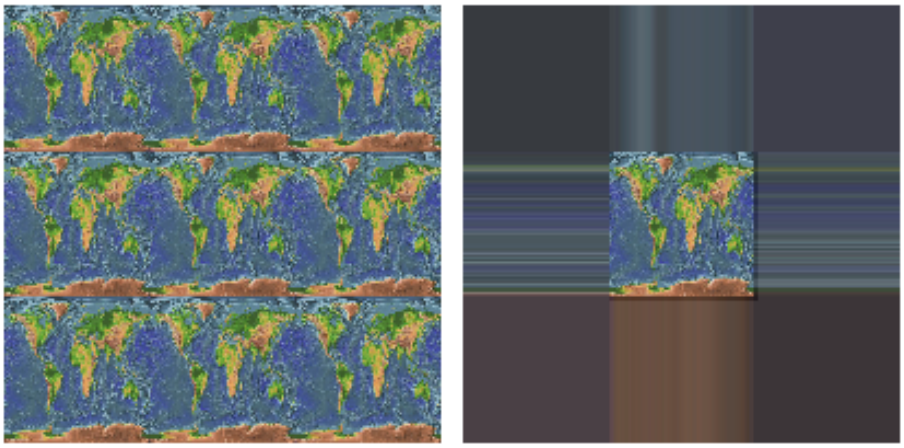

Figure 4: Powtarzanie lub "zacisk" tekstury

5 Transformacja tekstury

Gdy tekstura jest stosowana do prymitywu, współrzędne tekstury dla wierzchołka określają, który punkt tekstury jest odwzorowany na ten wierzchołek. Obrazy tekstur są 2D, ale OpenGL obsługuje również jednowymiarowe tekstury i trójwymiarowe tekstury. Oznacza to, że współrzędne tekstury nie mogą być ograniczone do dwóch współrzędnych. W rzeczywistości zbiór współrzędnych tekstury w OpenGL jest reprezentowany wewnętrznie w postaci jednorodnych współrzędnych, które określa się jako (s, t, r, q). Użyliśmy `glTexCoord2*` do określenia współrzędnych tekstury s i t, ale wywołanie `glTexCoord2f (s, t)` jest naprawdę rwnoznacznie do `glTexCoord4f(s, t,0,1)`.

Ponieważ współrzędne tekstury nie różnią się od współrzędnych wierzchołków, można je przekształcać dokładnie w ten sam sposób. OpenGL utrzymuje transformację tekstury jako część swojego stanu, wraz z transformacją modelu i projekcji. Bieżąca wartość każdej z trzech transformacji jest przechowywana jako macierz. Gdy tekstura jest stosowana do obiektu, współrzędne tekstury, które zostały określone dla jej wierzchołków, są przekształcane przez macierz tekstury. Przekształcone współrzędne tekstury są następnie używane do wybrania punktu w teksturze. Oczywiście domyślną transformacją tekstury jest transformacja tożsamości, która nie zmienia współrzędnych.

Macierz tekstur może reprezentować skalowanie, obrót, translację i kombinacje tych podstawowych transformacji. Aby określić transformację tekstury, musisz użyć `glMatrixMode()`, aby ustawić tryb macierzy na `GL_TEXTURE`. W tym trybie wywołania metod takich jak `glRotate *`, `glScale *` i `glLoadIdentity` są stosowane do macierzy tekstur. Na przykład, aby zainstalować transformację tekstury, która skaluje współrzędne tekstury o współczynnik dwa w każdym kierunku, można stosować:

```
glMatrixMode(GL_TEXTURE);
glLoadIdentity(); // Upewnij się, że zaczynamy od macierzy tożsamości.
glScalef(2,2,1);
glMatrixMode(GL_MODELVIEW); // Pozostaw tryb macierzy ustawiony na GL_MODELVIEW.
```

Ponieważ obraz leży w płaszczyźnie st, tylko dwa pierwsze parametry `glScalef` mają znaczenie. W przypadku obrotów należy użyć `(0,0,1)` jako osi obrotu, która obróci obraz w płaszczyźnie st.

Co to właściwie oznacza dla wyglądu tekstury na powierzchni? W przykładzie transformacja skalowania mnoży każdą współrzędną tekstury przez 2. Na przykład, jeśli do wierzchołka przypisano współrzędne tekstury 2D `(0.4,0.1)`, to po zastosowaniu transformacji tekstury ten wierzchołek zostanie odwzorowany na punkt $(y, t) = (0.8, 0.2)$ w teksturze. Współrzędne tekstury zmieniają się dwa razy szybciej na powierzchni, tak jak bez transformacji skalowania. Obszar na powierzchni, który będzie mapował do kwadratu o wymiarach 1 na 1 w obrazie tekstury bez transformacji, zamiast tego odwzoruje na obrazie kwadrat o wymiarach 2 na 2 - tak, że większy fragment obrazu będzie widoczny wewnątrz obszaru. Innymi słowy, **obraz tekstury zmniejszy się dwukrotnie na powierzchni**! Mówiąc ogólnie, **efekt transformacji tekstury na wygląd tekstury jest odwrotnością jej wpływu na współrzędne tekstury**. (Jest to dokładnie analogiczne do odwrotnej zależności między transformacją oglądania a transformacją modelowania.) Jeśli transformacja tekstury jest translacją w prawo, tekstura porusza się w lewo na powierzchni. Jeśli transformacja tekstury jest obróceniem w lewo, tekstura obraca się zgodnie z ruchem wskazówek zegara na powierzchni.

Wspomniamy tutaj o transformacjach tekstur, aby pokazać, jak OpenGL może wykorzystywać transformacje w innym kontekście. Czasami jednak warto przekształcić teksturę, aby lepiej pasowała do powierzchni. Aby uzyskać niezwykły efekt, możesz nawet animować transformację tekstury, aby obraz tekstury poruszył się po powierzchni.

6 Ładowanie tekstury z pamięci

OpenGL nie ma funkcji ładowania obrazów z pliku. Na razie zakładamy, że plik został już załadowany z pliku do pamięci komputera. W dalszej części wyjaśnię, jak to zrobić w języku C i Java.

Funkcją OpenGL do ładowania danych obrazu z pamięci komputera do tekstury 2D jest `glTexImage2D()`, która ma postać:

```
glTexImage2D(target, mipmapLevel, internalFormat, width, height, border,
                              format, dataType, pixels);
```

`target` powinno być `GL_TEXTURE_2D`. Poziom `mipmapLevel` powinien zwykle wynosić 0. Wartość 0 służy do ładowania głównej tekstury; większa wartość jest używana do załadowania pojedynczej mipmapy. `InternalFormat` mówi OpenGL, jak chcesz, aby dane tekstury były przechowywane w pamięci tekstur

OpenGL. Zwykle `GL_RGB` przechowuje 8-bitowy komponent czerwony / zielony / niebieski dla każdego piksela. Inną możliwością jest `GL_RGBA`, która dodaje komponent alfa. Szerokość i wysokość dają rozmiar obrazu; wartości powinny zwykle być potęgami dwóch. Wartość `border` powinna wynosić 0; jedyną inną możliwością jest 1, co oznacza, że wokół danych obrazu dodano jednopikselową ramkę. Ostatnie trzy parametry opisują dane obrazu. `Format` informuje, w jaki sposób oryginalne dane obrazu są reprezentowane w pamięci komputera, takie jak `GL_RGB` lub `GL_RGBA`. Typ danych to zazwyczaj `GL_UNSIGNED_BYTE`, co oznacza, że każdy komponent koloru jest reprezentowany jako wartość jednobajtowa w zakresie od 0 do 255. `Pixels` to wskaźnik do początku rzeczywistych danych kolorów dla pikseli. Dane pikselowe muszą być w określonym formacie, ale nie muszą nas tu dotyczyć, ponieważ zazwyczaj zajmują się nimi funkcje używane do odczytu obrazu z pliku. (W przypadku JOGL wskaźnik zostanie zastąpiony buforem.)

Wszystko to wygląda na dość skomplikowane, ale w praktyce wywołanie funkcji `glTexImage2D` generalnie przyjmuje następującą postać, z wyjątkiem możliwości zastąpienia `GL_RGB` przez `GL_RGBA`.

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
                    GL_RGB, GL_UNSIGNED_BYTE, pixels);
```

Wywołanie tej funkcji spowoduje załadowanie obrazu do tekstury, ale nie spowoduje użycia tekstury. W tym celu musisz również wywołać

```
glEnable (GL_TEXTURE_2D);
```

Jeśli chcesz użyć tekstury na niektórych obiektach, ale nie na innych, możesz włączyć `GL_TEXTURE_2D` przed narysowaniem obiektów, które chcesz teksturować, i wyłączyć je przed narysowaniem obiektów bez tekstur. Możesz także zmienić teksturę, która jest używana w dowolnym momencie, wywołując `glTexImage2D`.

7 Tekstura z bufora kolorów

Obrazy tekstur używane w programie OpenGL zwykle pochodzą ze źródła zewnętrznego, najczęściej pliku obrazu. Jednak OpenGL jest potężnym silnikiem do tworzenia obrazów. Czasami zamiast ładować plik obrazu, wygodnie jest utworzyć obraz OpenGL wewnętrznie, renderując go. Jest to możliwe, ponieważ OpenGL może odczytywać dane tekstury z własnego bufora kolorów, gdzie wykonuje swój rysunek. Aby utworzyć obraz tekstury przy użyciu OpenGL, wystarczy narysować obraz przy użyciu standardowych poleceń rysowania OpenGL, a następnie załadować ten obraz jako teksturę przy użyciu metody

```
glCopyTexImage2D( target, mipmapLevel, internalFormat,
                                    x, y, width, height, border );
```

W tej metodzie `target` będzie `GL_TEXTURE_2D`; `mipmapLevel` powinien wynosić zero; `InternalFormat` będzie zwykle `GL_RGB` lub `GL_RGBA`;]textttx i `y` określają

lewy dolny róg prostokąta, z którego będzie czytana tekstura; `width` i `height` są wielkościami tego prostokąta; a `border` powinna wynosić 0. Jak zwykle w przypadku tekstur, szerokość i wysokość zwykle powinny być potęgami dwóch. Wywołanie funkcji `glCopyTexImage2D` będzie zazwyczaj wyglądać tak

```
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, x, y, width, height, 0);
```

W rezultacie określony prostokąt z bufora kolorów zostanie skopiowany do pamięci tekstury i stanie się bieżącą teksturą 2D. Działa to w taki sam sposób, jak wywołanie funkcji `glTexImage2D()`, z wyjątkiem źródła danych obrazu.

8 Obiekty tekstury

Wszystko, co do tej pory, było już prawdziwe dla OpenGL 1.0. OpenGL 1.1 wprowadził nową funkcję zwaną obiektami tekstury, aby uczynić obsługę tekstur bardziej wydajną. Obiekty tekstury są używane, gdy trzeba pracować z kilkoma obrazami tekstury w tym samym programie. Zwykła metoda ładowania obrazów tekstur, `glTexImage2D`, przesyła dane z programu do karty graficznej. Jest to kosztowna operacja, a przełączanie między wieloma teksturami za pomocą tej metody może poważnie obniżyć wydajność programu. Obiekty tekstury oferują możliwość przechowywania danych tekstury dla wielu tekstur na karcie graficznej. Z obiektami tekstury możesz przełączać się z jednego obiektu tekstury na inny za pomocą pojedynczego, szybkiego polecenia OpenGL: Musisz tylko powiedzieć OpenGL, który obiekt tekstury chcesz użyć. (Oczywiście karta graficzna ma tylko ograniczoną ilość pamięci do przechowywania tekstur i nie masz gwarancji, że wszystkie obiekty tekstur będą przechowywane na karcie graficznej.)

Obiekty tekstur są zarządzane przez OpenGL i sprzęt graficzny. Obiekt tekstury jest identyfikowany przez liczbę całkowitą. Aby użyć obiektu tekstury, musisz uzyskać numer ID z OpenGL. Odbywa się to za pomocą funkcji `glGenTextures`:

```
void glGenTexures (int textureCount, int * array)
```

Ta funkcja może generować wiele identyfikatorów tekstur za pomocą jednego połączenia. Pierwszy parametr określa liczbę żądanych identyfikatorów. Drugi parametr mówi, gdzie zostaną zapisane wygenerowane identyfikatory. Powinna to być tablica, której długość wynosi co najmniej `textureCount`. Na przykład, jeśli planujesz użyć trzech obiektów tekstur, możesz wywołać

```
int idList [3];
glGenTextures (3, idList);
```

Następnie możesz użyć `idList[0], idList[1]` i `idList[2]`, aby odnieść się do tekstur.

Każdy obiekt tekstury ma swój własny stan, który zawiera wartości parametrów tekstury, takie jak `GL_TEXTURE_MIN_FILTER`, a także sam obraz tekstury. Aby pracować z określonym obiektem tekstury, musisz najpierw wywołać

```
glBindTexture (GL_TEXTURE_2D, texID)
```

gdzie `texID` to identyfikator tekstury zwrócony przez `glGenTextures`. Po tym wywołaniu każde użycie `glTexParameteri`, `glTexImage2D` lub `glCopyTexImage2D` zostanie zastosowane do obiektu tekstury z identyfikatorem ID `texID`.

Podobnie, gdy renderowany jest prymityw teksturowany, używana jest tekstura, która została ostatnio powiązana przy użyciu `glBindTexture`. Typowy wzór to ładowanie i konfigurowanie wielu tekstur podczas inicjowania programu:

```
glGenTextures( n, textureIdList );
for (i = 0; i < n; i++) {
    glBindTexture( textureIDList[i] );
      .
      .  // Load texture image number i
      .  // Configure texture image number i
      .
}
```

Następnie, podczas renderowania sceny, wywołujesz `glBindTexture` za każdym razem, gdy chcesz przełączyć się z jednego obrazu tekstury na inny obraz tekstury. Byłoby to znacznie bardziej wydajne niż wywoływanie `glTexImage2D` za każdym razem, gdy chcesz zmienić tekstury.

OpenGL 1.1 rezerwuje ID tekstury zero jako domyślny obiekt tekstury, który jest początkowo związany. Jest to obiekt tekstury, którego używasz, jeśli nigdy nie wywołujesz `glBindTexture`. Oznacza to, że możesz pisać programy, które używają tekstur, nie wspominając już o `glBindTexture`.

Literatura

Uwaga!

Tekstury OpenGL w języku C
teksturowanie obiektów elementarnych
https://drive.google.com/open?id=13ykFBEDMbbIqWAHj3ulPeSnvWPUzTJwkq2dQLZwYSBg
teksturowanie z użyciem kompresji oraz przezroczystości
https://drive.google.com/open?id=162RwFie9pVguUD1dMwZztugWiY_o_WaTRcLFdaq0_zE
W języku angielskim

- książka interakcyjna: Image Textures http://math.hws.edu/graphicsbook/c4/s3.html (rozdział 4.3)

9 Zadanie

Celem jest teksturowanie piramidy z użyciem dwóch sposobów ładowania tekstur: użycie tekstury z buforu kolorów (rysowanie w Panel); ładowanie tekstury z pliku (trzy pliki przykładowe do pobrania).