Laboratorium Grafiki Komputerowej i Animacji

Ćwiczenie VI

Biblioteka OpenGL - teksturowanie

Sławomir Samolej

Rzeszów, 1999

1. Wstęp

Podczas tworzenia skomplikowanych obiektów graficznych przydatnym mechanizmem staje się teksturowanie - możliwość "pokrywania" wielokątów zapisanymi cyfrowo obrazami. Teksturowanie pozwala na odwzorowywanie własności pewnego typu materiałów pokrywających powierzchnię obiektu bez konieczności czasochłonnego budowania tych powierzchni z prymitywów graficznych. W systemach komputerowych z wbudowaną akceleracją sprzętową obliczania tekstur teksturowanie umożliwia znaczne przyspieszenie pracy animacji, przy bardziej realistycznych efektach wizualnych. Zastosowanie odpowiednich tekstur wzmacnia złudzenie rzeczywistości wyświetlanej sceny.

Opracowanie dotyczy sposobu wykorzystania plików typu BMP (bitmapa) jako obrazów cyfrowych przeznaczonych do teksturowania obiektów tworzonych przy pomocy biblioteki OpenGL. Dodatkowo omówione są tak zwane listy wyświetlania OpenGL (ang. display lists), oraz sposób wykorzystywania zestawu obiektów geometrycznych wchodzących w skład biblioteki glu. Do opracowana dołączona jest dyskietka zawierająca omawiane w tekście przykłady.

2. Listy wyświetlania

Listy wyświetlania OpenGL zaprojektowane zostały po to, aby istniała możliwość zapamiętania w pamięci ciągu instrukcji OpenGL, a następnie szybkie ich odtworzenie. Część obliczeń znajdujących się wewnątrz listy wyświetlania wykonywana jest przed pojawieniem się sceny na ekranie, stąd wywołanie listy ma znamiona odtwarzania wcześniej obliczonych transformacji, bez konieczności obliczania ich w trakcie wyświetlania. Stopień optymalizacji list wyświetlania zależy od implementacji biblioteki OpenGL dla danego systemu operacyjnego. Standard OpenGL przewiduje jedynie, że zastosowanie list wyświetlania nie może spowolnić pracy programu. Sugeruje się używanie list wyświetlania w następujących przypadkach:

- **operacje macierzowe** - większość operacji macierzowych wymaga obliczenia odwrotności macierzy. Macierze i ich odwrotności mogą być przechowywane w listach wyświetlania;
- **operacje definiowania światła oraz właściwości materiałów** - umieszczenie komend definiujących właściwości materiałów w liście wyświetlania w scenach, w których często odbywa się redefinicja tych właściwości może spowodować wygenerowanie kodu programu, który będzie się wykonywał szybciej;
- **teksturowanie** - z reguły dane odczytywane z plików graficznych nie odpowiadają danym dostosowanym do toru graficznego komputera. W czasie kompilowania listy następuje transformacja danych graficznych, na takie jakie odpowiadają sprzętowi.

Listę wyświetlania OpenGL tworzy się umieszczając ciąg komend OpenGL pomiędzy wywołaniami funkcji glNewList() i glEndList():

```
glNewList(1, GL_COMPILE);
...
// Instrukcje OpenGL
...
glEndList();
```

Jako drugi parametr funkcji glNewList() można podać GL_COMPILE lub GL_COMPILE_AND_EXECUTE. Gdy podany jest pierwszy z nich zawartość listy jest kompilowana i zachowywana w pamięci. Podanie parametru GL_COMPILE_AND_EXECUTE powoduje automatyczne skompilowanie, zachowanie w pamięci, a następnie wykonanie komend

zawartych w liście. Lista identyfikowana jest w systemie poprzez jej numer podany jako pierwszy parametr wywołania funkcji glNewList(). Wykonanie listy identyfikowanej w systemie jako 1 można wywołać przy pomocy instrukcji:

```
glCallList(1);
```

Istnieje możliwość automatycznego generowania identyfikatorów list. Służyć do tego może dodatkowa funkcja **glGenLists()**. Wywołanie tej funkcji w sposób podany poniżej zapewnia wygenerowanie zawsze unikalnych w systemie identyfikatorów list.

```
GLuint      SkyTexture;
...
glNewList(SkyTexture = glGenLists(1), GL_COMPILE);
...
glEndList();
```

3. Teksturowanie

W standardzie OpenGL teksturowanie obiektów powinno się składać z następujących etapów:

- Przygotowania tekstury
- Ustalenia w jaki sposób tekstura odpowiadać będzie poszczególnym pikselom obrazu
- Uaktywnienia teksturowania w systemie OpenGL
- Utworzenia sceny zawierającej prymitywy i "rozciągnięte" na nich tekstury

Przygotowanie tekstury polega na włączeniu do pamięci programu tablicy pikseli reprezentującej jedno lub dwuwymiarowy obraz. Poniżej zaprezentowany zostanie sposób włączania plików DIB jako tekstur OpenGL. Budowa pliku typu BMP omówiona została w opracowaniu dotyczącym ćwiczenia II z przedmiotu Grafika Komputerowa i Animacja.

Rozmiar tablicy pikseli musi spełniać zależność: $(2^m+2b)\times(2^n+2d)$, gdzie m, n, b, d są liczbami całkowitymi. Liczby b i d określają szerokość dodatkowej ramki obrazu. Minimalny rozmiar obrazu musi wynosić 64×64 piksele. Należy pamiętać, że w plikach typu bitmapa dane mówiące o kolorze piksela zorganizowane są w kolejności BGR (składowe: niebieska-zielona-czerwona). Standard OpenGL oczekuje tablicy pikseli zorganizowanej w kolejności RGB (składowe: czerwona-zielona-niebieska). Wszystkie aplikacje znajdujące się na dołączonej do opracowania dyskietce zawierają zestaw gotowych funkcji służących do przetwarzania plików typu BMP w celu wyświetlania ich przy pomocy komend OpenGL. Prototypy i kody funkcji umieszczono w plikach bitmap.h i bitmap.c. Dwie podstawowe z funkcji to:

```
void * LoadDIBitmap(char *filename, BITMAPINFO **info);
```

oraz

```
GLubyte *ConvertRGB(BITMAPINFO *info, void *bits);
```

Pierwsza z nich ładuje do pamięci plik typu BMP o nazwie *filename* umieszczając nagłówek bitmapy w strukturze wskazywanej przez *info*. Wartością zwracaną jest wskaźnik na zaalokowaną tablicę pikseli. Druga funkcja dokonuje automatycznej konwersji formatu danych opisujących kolor piksela z BGR na RGB. Jako parametry przyjmuje ona wskaźnik na strukturę zawierającą nagłówek pliku BMP oraz wskaźnik do tablicy pikseli. Wartością zwracaną jest nowa tablica pikseli po konwersji.

Włączenie teksturowania do sceny poprzedzone musi być modyfikacją ustawień systemowych OpenGL. Podstawową funkcją ustalającą parametry teksturowania jest **glTexEnv*()**. Wywołanie tej funkcji w sposób:

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

ustala sposób określania koloru tekstur na scenie. Ostatni parametr wywołania funkcji może przyjmować wartości: **GL_MODULATE**, **GL_DECAL** oraz **GL_BLEND**. Jeśli wynosi on GL_DECAL to kolor pikseli tekstury pokrywa w całości teksturowany obiekt i nie jest możliwe uzyskanie efektów związanych z oświetleniem. Parametr GL_MODULATE powoduje, że kolor pokrytego teksturą obiektu filtrowany jest przez kolor pokrywającej go tekstury. Parametr GL_BLENT ustala, że podstawowe znaczenie w wyświetlaniu koloru teksturowanego obiektu ma zdefiniowany kolor prymitywu. Jeśli nie zostanie zdefiniowany stopień przeźroczystości prymitywu to tekstura nie będzie wcale widoczna. Podanie parametrów GL_BLEND i GL_MODULATE pozwala na uzyskiwanie efektów związanych z oświetleniem. System OpenGL musi zostać poinformowany o wprowadzeniu teksturowania. Wywołanie funkcji glEnable() z parametrami **GL_TEXTURE_2D** lub **GL_TEXTURE_1D** uaktywnia obliczanie tekstur w systemie. Fragment kodu:

```
glDisable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_1D);
```

włącza w systemie teksturowanie obiektów przy pomocy jednowymiarowych tekstur.

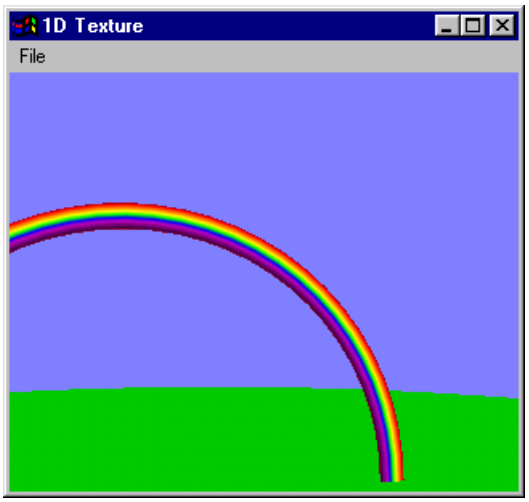
Do zdefiniowania w OpenGL tekstury jednowymiarowej służy funkcja o prototypie:

```
void glTexImage1D( GLenum target, GLint level, GLint components,
                   GLsizei width, GLint border, GLenum format,
                   GLenum type, const GLvoid *pixels );
```

Funkcja pobiera osiem argumentów. Argument *target* podaje jakiego typu tekstura ma zostać zdefiniowana i musi on przyjmować wartość GL_TEXTURE_1D. Argument *level* wskazuje poziom oddawania szczegółów zawartych w teksturze i zwykle ustawiany jest na 0. Inne wartości tego argumentu służą do zdefiniowania mniej szczegółowych postaci tekstur. Argument *components* wskazuje ile wartości liczbowych w tablicy pikseli opisuje jeden piksel ekranowy. Dla trybu opisu koloru RGB wartość powinna wynosić 3 dla trybu RGBA - 4. Wartość 1 oznacza, że kolory tekstury opisywane będą w trybie indeksowym (przy pomocy palety kolorów). Parametry *width* i *border* opisują rozmiary obrazu. Wartość *border* kontroluje ilość pikseli tworzących ramkę wokół obrazu i może przyjmować wartości 0, 1, 2. Parametr *width* określa długość tekstury i musi być to liczbą będąca potęgą liczby 2. Argument *format* oznacza typ opisywania kolorów pikseli i może przyjmować wartości GL_COLOR_INDEX (gdy kolory opisywane są w trybie indeksowym), GL_LUMINANCE, GL_RGB (gdy kolor opisują trzy liczby), GL_RGBA (gdy kolor opisują cztery liczby). Parametr *type* określa typ danych przechowujących informacje o kolorach pikseli (dla plików typu bitmapa parametr ten powinien wynosić GL_UNSIGNED_BYTE). Ostatni parametr wywołania funkcji jest wskaźnikiem do tablicy pikseli. Podany poniżej fragment kodu pochodzi z programu TEXT1D z załączonej do opracowania dyskietki. Zawiera on funkcję definiującą jednowymiarową teksturę, służącą później do wyrysowania tęczy w programie (rys. 3.1).

```
void
LoadAllTextures(void)
{
  static unsigned char roygbiv_image[8][3] =
```

```
  {
    { 0x3f, 0x00, 0x3f },     /* Dark Violet (for 8 colors...) */
    { 0x7f, 0x00, 0x7f },     /* Violet */
    { 0xbf, 0x00, 0xbf },     /* Indigo */
    { 0x00, 0x00, 0xff },     /* Blue */
    { 0x00, 0xff, 0x00 },     /* Green */
    { 0xff, 0xff, 0x00 },     /* Yellow */
    { 0xff, 0x7f, 0x00 },     /* Orange */
    { 0xff, 0x00, 0x00 }      /* Red */
  };

  glNewList(RainbowTexture = glGenLists(1), GL_COMPILE);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage1D(  GL_TEXTURE_1D, 0, 3, 8,
                   0, GL_RGB, GL_UNSIGNED_BYTE, roygbiv_image);
  glEndList();
}
```



Rys. 3.1 Okno programu TEX1D

W pokazanym fragmencie kodu tworzona jest lista wyświetlania zawierająca obraz przeznaczony do teksturowania oraz wymagane funkcje definiujące w jaki sposób zachowywać się ma utworzona tekstura jeśli rozpinana będzie ona na obiektach mających inne rozmiary niż rozmiary obrazu. Do określania właściwości tekstur w OpenGL służy funkcja **glTexParameter*()**. Pierwszy argument funkcji wskazuje jakiego typu tekstura będzie modyfikowana (GL_TEXTURE_1D - jednowymiarowa lub GL_TEXTURE_2D - dwuwymiarowa). Argument drugi wyznacza własność teksturowania, która będzie modyfikowana. Zaś argument trzeci opisuje sposób modyfikowania własności. Jeśli drugim parametrem wywołania funkcji jest GL_TEXTURE_MAG_FILTER to funkcja opisuje w jaki sposób zachowywać się ma bitmapa, gdy "rozciągana" będzie na obiekcie o rozmiarach większych niż rozmiary obrazu. W przypadku, gdy drugi parametr przyjmuje wartość GL_TEXTURE_MIN_FILTER, funkcja określa sposób "rozciągania" bitmapy na obiekcie mniejszym niż rozmiar obrazu. Jeśli trzeci parametr wywołania funkcji przyjmuje wartość GL_LINEAR, to przed wyrysowaniem tekstury jest ona liniowo interpolowana. Gdy trzecim parametrem wywołania funkcji jest GL_NEAREST rozszerzanie lub zmniejszanie tekstury polega na powielaniu wybranych pikseli.

Do zdefiniowania dwuwymiarowej tekstury w standardzie OpenGL służy funkcja:

```
void glTexImage2D(GLenum target, GLint level, GLint components,
```

```
                GLsizei width,    GLsizei height, GLint border,
                GLenum format, GLenum type, const GLvoid *pixels
                );
```

W porównaniu z funkcją glTexImage1D() ma ona o jeden argument więcej (*height*) określający wysokość przetwarzanego obrazu. Parametry *height* i *width* muszą być liczbami będącymi potęgami liczby 2. Podany poniżej fragment kodu pochodzi z programu TEX2D i zawiera funkcję, która definiuje dwuwymiarową teksturę OpenGL na podstawie wczytanego pliku typu bitmapa:

```
void
LoadAllTextures(void)
{
  BITMAPINFO      *info;                        /* Bitmap information */
  void            *bits;                        /* Bitmap pixel bits */
  GLubyte   *rgb;                  /* Bitmap RGB pixels */


 /*
  * Try loading the bitmap and converting it to RGB...
  */

  bits = LoadDIBitmap("sky.bmp", &info);
  if (bits == NULL)
    return;

  rgb = ConvertRGB(info, bits);
  if (rgb == NULL)
  {
    free(info);
    free(bits);

    return;
  };

  glNewList(SkyTexture = glGenLists(1), GL_COMPILE); //glCallLists
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

   /*
    * Define the 2D texture image.
    */

    glPixelStorei(GL_UNPACK_ALIGNMENT, 4);      /* Force 4-byte alignment */
    glPixelStorei(GL_UNPACK_ROW_LENGTH, 0);
    glPixelStorei(GL_UNPACK_SKIP_ROWS, 0);
    glPixelStorei(GL_UNPACK_SKIP_PIXELS, 0);

    glTexImage2D( GL_TEXTURE_2D, 0, 3,
                  info->bmiHeader.biWidth,
                  info->bmiHeader.biHeight, 0,
                  GL_RGB, GL_UNSIGNED_BYTE, rgb
                );
  glEndList();

 /*
  * Free the bitmap and RGB images, then return 0 (no errors).
  */

  free(rgb);
```

```
  free(info);
  free(bits);
}
```

Program TEX2D stosuje załadowaną bitmapę do wyrysowania nieba (rys. 3.2). Pokazany wyżej fragment kodu zawiera dodatkowe wywołania funkcji glTexParameter*(). Omówione one zostaną w dalszej części opracowania. Przed zdefiniowaniem tekstury wywołano dodatkowo kilkakrotnie komendę **glPixeStore*()**. Określa ona w jaki sposób dane graficzne o teksturze będą odczytywane bądź przechowywane w pamięci. Znaczenie poszczególnych parametrów wyjaśnia tabela poniżej.

Prototyp funkcji:
`void glPixelStore{if}(GLenum pname, TYPE param)`

Parametry:		
pname	GL_UNPACK_ALIGNMENT	ustala w jaki sposób uporządkowane zostaną dane w tablicy pikseli oznaczające kolejne wiersze obrazu (Podanie jako drugi parametr wywołania funkcji wartości 4 zapewnia prawidłową interpretację ułożenia pikseli w pochodzących z plików typu BMP)
	GL_UNPACK_LENGTH	definiuje ilość pikseli w rzędzie. Jeśli drugi parametr wywołania funkcji wynosi 0 ilość pikseli w rzędzie liczona jest automatycznie na podstawie rozmiarów tablicy pikseli
	GL_UNPACK_SKIP_ROWS	definiuje ile pikseli opuścić w kierunku pionowym; podanie jako drugi argument liczby 0 powoduje, że mapa bitowa będzie odtwarzana od pierwszego górnego wiersza
	GL_UNPACK_SKIP_PIXELS	definiuje ile pikseli opuścić w kierunku poziomym; podanie jako drugi parametr liczby 0 powoduje, że mapa bitowa będzie odtwarzana od pierwszej (lewej) kolumny

Wywołania funkcji glPixelStore*() zaprezentowane w podanym wcześniej fragmencie kodu zapewniają w systemie Windows prawidłową interpretację i organizację map bitowych przeznaczonych do teksturowania obiektów OpenGL. Komenda glPixelStore*() z parametrami GL_UNPACK_SKIP_ROWS i GL_UNPACK_SKIP_PIXELS pozwala na ustalenie wycinka z obrazu, który ma być wyświetlony.

Rys 3.2 Okno programu TEX2D

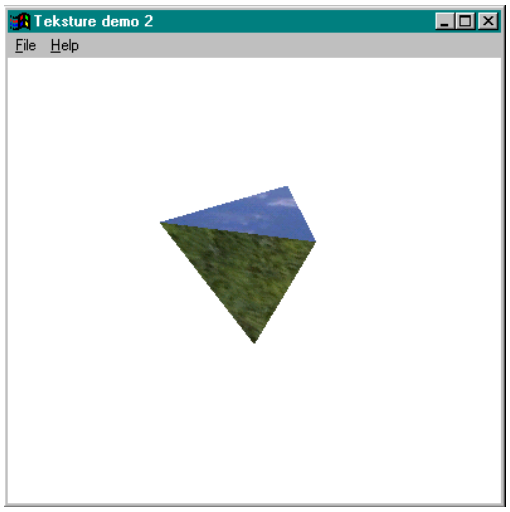
Podany poniżej fragment kodu pokazuje sposób "rozpięcia" jednowymiarowej bitmapy na ciągu prymitywów tworzących łuk tęczy (program TEX1D):

```
    glEnable(GL_TEXTURE_1D);
    glCallList(RainbowTexture);
    glBegin(GL_QUAD_STRIP);
      for (th = 0.0; th <= M_PI; th += (0.03125 * M_PI))
      {
        x = cos(th) * 50.0;
        y = sin(th) * 50.0;
        z = -50.0;
        glTexCoord1f(0.0);
        glVertex3f(x, y, z);

        x = cos(th) * 55.0;
        y = sin(th) * 55.0;
        z = -50.0;
        glTexCoord1f(1.0);
        glVertex3f(x, y, z);
      }
   glEnd();
```

Do umiejscowienia tekstury na poszczególnych wielokątach wykorzystuje się funkcję **glTexCoord*()**. Wartość 0.0 reprezentuje piksel w mapie bitowej położony najbardziej z lewej strony, natomiast wartość 1.0 reprezentuje prawy skrajny piksel obrazu. Umiejscawianie bitmapy na prymitywie polega kolejno na podaniu współrzędnych tekstury, które mają być przywiązane do wierzchołka (funkcja glTexCoord*()), a następnie współrzędnych samego wierzchołka.

Przykład rozłożenia dwuwymiarowej tekstury na obiekt pochodzący z programu CZWOR1 (rys. 3.3) znajduje się poniżej:

```
glCallList(SkyTexture);
glBegin(GL_TRIANGLES);
      glNormal3f(0.0f, -1.0f, 0.0f);
      glTexCoord2f(0.5f, 0.5f);          glVertex3f(0.0f,-24.48f, 34.66f);
      glTexCoord2f(0.0f, 1.0f);          glVertex3f(-30.0f, -24.48f, -17.4f);
      glTexCoord2f(1.0f, 1.0f);          glVertex3f(30.0f,-24.48f,-17.4f);
glEnd();
```



Rys 3.3 Okno programu CZWOR1

Utworzona w OpenGL dwuwymiarowa tekstura ma zawsze wymiary 1.0 na 1.0. Oznacza to przykładowo, że jeśli w funkcji glCoord2f() podamy jako parametry liczby 0.0 i 1.0 to odwołujemy się do lewego dolnego rogu obrazu graficznego. Jeśli liczby podawane jako współrzędne tekstur przekraczają zakres 0.0 - 1.0, to wyświetlanie bitmap zależy od ustawień podanych w wywołaniu funkcji glTexParameter*() wywołanej z pierwszym parametrem GL_TEXTURE_2D i z drugim parametrem GL_TEXTURE_WRAP_S lub GL_TEXTURE_WRAP_T (litera S odnosi się do współrzędnej poziomej tekstury, litera T odnosi się do współrzędnej pionowej tekstury). Jeśli przykładowo wywołanie funkcji wygląda:

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

to trzeci parametr (GL_CLAMP) decyduje, że rozmiary tekstury ograniczone są do wielkości od 0.0 do 1.0 w kierunku poziomym i tekstura nie jest powielana w celu pokrycia obiektu. Wywołanie tej samej funkcji w sposób:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

zapewnia, że w przypadku podania w wywołaniu funkcji glCoord2f() parametrów większych od 1.0 rozłożenie tekstury na obiekcie polegać będzie na odpowiednim powieleniu mapy bitowej wzdłuż osi poziomej (Podanie tej samej komendy z parametrem GL_TEXTURE_WRAP_T decydowało będzie o sposobie powielania tekstury wzdłuż osi poziomej tekstury). Przykładowe wywołania funkcji glTexParameter*() z omówionymi powyżej parametrami znajdują się w prezentowanym wcześniej fragmencie kodu pochodzącym z programu TEX2D.

Standard OpenGL umożliwia również wygenerowanie zestawu tekstur na podstawie jednej bitmapy zawierających zróżnicowany poziom wyświetlania detali obrazu. Jeśli na podstawie jednego obrazu zostaje stworzony zestaw tekstur, to OpenGL automatycznie wybiera, którą z nich umieścić na scenie (zależy to od położenia obiektu na scenie). Podane poniżej dwa fragmenty kodu pochodzą z plików *texture.c* wchodzących w skład projektów aplikacji dołączonych do opracowania. Pokazano sposób wykorzystania funkcji **gluBuild1DMipmaps()** i **gluBuild2DMipmaps()** do automatycznego wygenerowania zestawu tekstur na podstawie jednego obrazu (standard OpenGL umożliwia wygenerowanie do 4 tekstur na podstawie jednego obrazu).

```
// Utwórz jednowymiarowe mipmapy (tekstury o różnych rozdzielczościach)
```

```
  gluBuild1DMipmaps(GL_TEXTURE_1D, 3, info->bmiHeader.biHeight,
                    GL_RGB, GL_UNSIGNED_BYTE, rgb);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_1D,GL_TEXTURE_MIN_FILTER,
             GL_NEAREST_MIPMAP_NEAREST);

// Utwórz dwuwymiarowe mipmapy (tekstury o różnych rozdzielczościach)
  gluBuild2DMipmaps(GL_TEXTURE_2D, 3, info->bmiHeader.biWidth,
                    info->bmiHeader.biHeight, GL_RGB, GL_UNSIGNED_BYTE, rgb);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                  GL_NEAREST_MIPMAP_NEAREST);
```

Towarzyszące tworzeniu zestawu tekstur wywołania funkcji glTexParameteri() z parametrami jak powyżej zapewniają optymalne zarządzanie teksturami w systemie (odpowiedni dobór tekstury do położenia obiektu).

4. Kwadrygi OpenGL - sfery, cylindry, dyski

Biblioteka OpenGL, w celu ułatwienia konstruowania bardziej złożonych scen zawiera zdefiniowane proste trójwymiarowe obiekty zwane kwadrygami. Definicje obiektów są składnikami biblioteki GLU (OpenGL Utility Library). Zestaw komend biblioteki GLU pozwala na tworzenie stożków, cylindrów, dysków i sfer. Każda kwadryga rysowana na ekranie posiada zestaw ustawień z nią związanych. Funkcja **gluNewQuadric()** tworzy strukturę danych identyfikującą kwadrygę w systemie oraz zawierającą zestaw zmiennych definiujących jej właściwości (sposób rysowania na ekranie, orientacja, tryb oświetlania, tryb teksturowania, funkcje odwołujące się do obiektu):

```
GLUquadricObj *obj;
obj= glNewQuadric();
```

Struktura nie zawiera informacji o kształcie kwadrygi.

Po utworzeniu w pamięci kwadrygi można dostosować jej kształt i właściwości do własnych potrzeb. Służą do tego następujące funkcje:

```
void gluQuadricDrawStyle( GLUquadricObj * qobj, GLenum drawStyle );
void gluQuadricNormals( GLUquadricObj *qobj, GLenum normals );
void gluQuadricOrientation( GLUquadricObj * quadObject, GLenum orientation );
void gluQuadricTexture( GLUquadricObj * quadObject, GLboolean textureCoords );
```

Pierwszym parametrem wywołania wszystkich wymienionych funkcji jest wskaźnik do struktury identyfikującej utworzoną kwadrygę. Funkcja gluQuadricDrawStyle() ustala typ prymitywów graficznych, przy pomocy których tworzony będzie obiekt. Domyślnym ustawieniem jest tworzenie obiektów z pasów złożonych z wielokątów - prymitywy OpenGL typu STRIP. Zestawienie dostępnych parametrów funkcji przedstawia tabela poniżej:

Parametr drawStyle	Opis
GLU_FILL	Kwadryga rysowana jest przy pomocy wypełnionych pasów wielokątów.
GLU_LINE	Kwadryga rysowana jest w postaci siatki składającej się z linii
GLU_SILHOUETTE	Kwadryga rysowana jest w postaci siatki złożonej z linii, widoczne są tylko zewnętrzne krawędzie
GLU_POINT	Kwadryga rysowana jest w postaci chmury wierzchołków

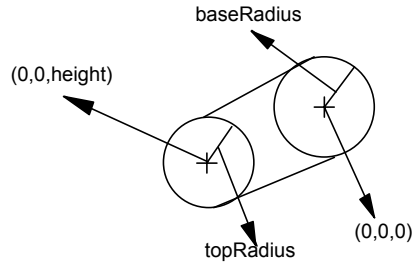
Normalne do kwadryg generowane są zwykle w sposób automatyczny. Funkcja gluQuadricNormals() steruje sposobem wyliczania normalnych. Dostępne warianty obliczania reakcji na padające światło dla kwadryg prezentuje tabela poniżej:

Parametr normals	Opis
GLU_NONE	Normalne nie są generowane
GLU_FLAT	Normalne generowane są jako prostopadłe do wielokątów tworzących powierzchnię
GLU_SMOOTH	Normalne generowane są dla każdego wierzchołka osobno w celu nadania "gładkości" powierzchniom tworzącym obiekt

Do ustalenia zwrotu wygenerowanych normalnych służy funkcja gluQuadricOrientation(). Jeśli normalne skierowane mają być na zewnątrz to drugim parametrem wywołania funkcji powinna być stała GLU_OUTSIDE. W przeciwnym wypadku - GLU_INSIDE.

Możliwe jest także automatyczne wygenerowanie koordynat tekstur pokrywających kwadrygi. Funkcja gluQuadricTexture() uaktywnia (GL_TRUE) lub zabrania (GL_FALSE) automatyczną generację współrzędnych tekstur przeznaczonych do pokrywania kwadryg.

Cylindry tworzy się przy pomocy funkcji **gluCylinder()**. Utworzony obiekt jest w rzeczywistości tubą, której oś przebiega wzdłuż osi z układu współrzędnych. Końce tuby nie są nigdy wypełniane (rys 4.1).



Rys. 4.1 Interpretacja paramertów funkcji gluCylinder()

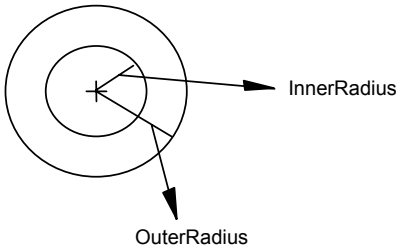
Prototyp funkcji ma postać:

```
void gluCylinder(
GLUquadricObj * qobj,
GLdouble baseRadius,
GLdouble topRadius,
GLdouble height,
GLint slices,
GLint stacks );
```

Parametry *baseRadius* i *topRadius* definiują promienie dolnej i górnej podstawy cylindra. Parametr height definiuje wysokość cylindra. Argumenty *slices* i *stacks* decydują o ilości wielokątów, z których składa się obiekt. Zwykle do utworzenia "gładkiego" walca wystarcza ustawienie parametru *slices* na 20. Dla uzyskania wysokiej jakości efektów oświetlenia obiektu parametr *stacks* powinien być zbliżony do parametru *slices*. Dla zwykłych zastosowań ustala się go na 2. Obiekty typu cylinder mogą także posłużyć do przybliżania brył o przekroju wielokąta jak na przykład ołówek. Zdefiniowany jak powyżej cylinder można w prosty sposób przekształcić w **stożek**. Wystarczy tylko nadać jednemu z promieni walca wartość 0. Nakładanie tekstury na cylinder odbywa się od punktu o współrzędnych (0, radius,0).

Dyski w bibliotece GLU mają postać płaskich kół z możliwością zdefiniowania w nich kolistych otworów o środku pokrywającym się ze środkiem pierwotnego koła (rys. 4.2). Prototyp funkcji definiującej dysk ma postać:

```
void gluDisk(
GLUquadricObj * qobj,
GLdouble innerRadius,
GLdouble outerRadius,
GLint slices,
GLint loops );
```



Rys. 4.2 Interpretacja parametrów funkcji gluDisk()

Parametry *innerRadius* i *outerRadius* decydują o rozmiarze dysku i ewentualnym rozmiarze otworu wewnątrz niego (rys. 4.2). Jeśli parametr *innerRadius* ustalony jest na 0, to dysk rysowany jest jako wypełnione koło. Parametr *slices* decyduje z ilu wycinków koła przybliżonych wielokątami składać się będzie rysowany obiekt (Do uzyskania efektu gładkości zwykle wystarcza wartość 20). Parametr *loops* decyduje o ilości koncentrycznych pierścieni dzielących dysk (znajdujących się pomiędzy wewnętrzną a zewnętrzną średnicą dysku). Zwykle parametr ten ustala się na 1 dla kół lub na 2 dla kół z wyciętym otworem wewnątrz. Dla osiągnięcia specjalnych efektów z oświetleniem liczbę tę należy powiększyć.

Istnieje możliwość zdefiniowania **wycinka dysku**. Dodatkowymi parametrami nowej funkcji są kąt początkowy (*startAngle*) i kąt końcowy (*sweepAngle*) liczone zgodnie z kierunkiem obrotu wskazówek zegara od góry dysku:

```
void gluPartialDisk(
GLUquadricObj * qobj,
GLdouble innerRadius,
GLdouble outerRadius,
GLint slices,
GLint loops,
GLdouble startAngle,
GLdouble sweepAngle );
```

Do tworzenia sfer służy funkcja o prototypie:

```
void gluSphere(
GLUquadricObj * qobj,
GLdouble radius,
GLint slices,
GLint stacks );
```

Parametr *radius* decyduje o promieniu sfery. Parametry slices (ilość południków sfery) i stacks (ilość równoleżników sfery) decydują o ilości wielokątów, z których zbudowany będzie obiekt.

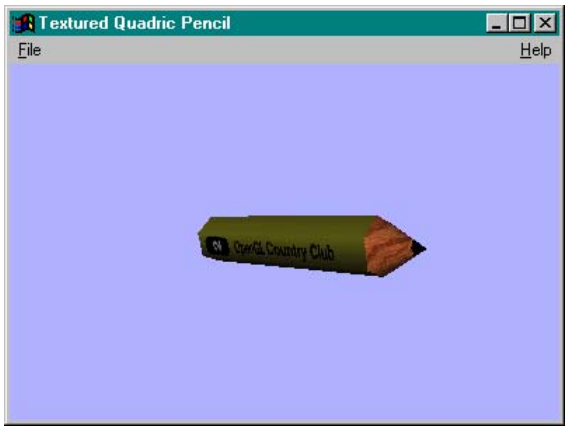
N załączonej do opracowania dyskietce znajduje się program PENCIL. Wykorzystuje on trzy kwadrygi typu walec do zdefiniowania ołówka. Ołówek pokryty jest bitmapami jak na rysunku 4.3.


Rys 4.3 Bitmapy wykorzystane do animacji ołówka

Początek ołówka jest stożkiem stworzony w sposób omówiony wcześniej w opracowaniu. Trzon ołówka zbudowano z cylindra. Końcówka ołówka zbudowana została również z cylindra ale o zerowej wysokości i promieniu jednej z podstaw równym 0. Taki sposób stworzenia ostatniej ściany obiektu wynikał z własności teksturowania cylindra (nie potrzeba było tworzyć nowej bitmapy, wystarczyło wykorzystać tę, która posłużyła do pokrycia początku ołówka - stożka). Podany fragment kodu prezentuje sposób stworzenia ołówka i pokrycia go teksturą:

```
gluQuadricNormals(PencilObj, GLU_FLAT);
// Ścina boczna ołówka
glCallList(PencilTexture);

glPushMatrix();
      glTranslatef(0.0, 0.0, -20.0);

      gluCylinder(PencilObj, 5.0, 5.0, 40.0, 6, 2);
glPopMatrix();

// Koniec i początek ołówka:
gluQuadricNormals(PencilObj, GLU_SMOOTH);
glCallList(LeadTexture);

glPushMatrix();
      glTranslatef(0.0, 0.0, 20.0);

      gluCylinder(PencilObj, 5.0, 0.0, 7.5, 6, 2);
glPopMatrix();

glPushMatrix();
      glTranslatef(0.0, 0.0, -20.0);
      gluCylinder(PencilObj, 5.0, 0.0, 0.0, 6, 2);
glPopMatrix();
```

Rysunek 4.4 prezentuje okno programu PENCIL.



Rys. 4.4 Przykładowe okno programu PENCIL

Bibliografia:

[1] R. S. Wright Jr., M. Sweet, *OpenGL Superbible*, The Waite Group Press, 1996
[2] *OpenGL Programming Guide*, Addison-Wessley, 1993
[3] E. Angel, *Interactive Computer Graphics*, Addison-Wesley, 1997
[4] R. Fosner, *OpenGL Programming for Windows and Windows NT*, Addison-Wesley, 1997
[5] R. Leniowski, *Wykłady z przedmiotu Grafika Komputerowa i Animacja*
[6] *Guide to OpenGL® on Windows® From Silicon Graphics®*, 1997