Ismerd meg!

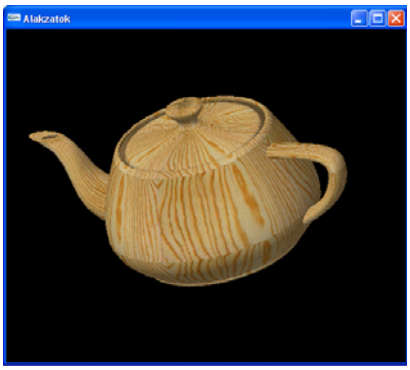
Számítógépes grafika

XXI. rész

Textúrák OpenGL-ben

A fotorealisztikus képek létrehozásának fontos eszköze a textúrázás. A valós tárgyak felülete mintázott, a mintákat pedig textúrák segítségével tudjuk reprodukálni.

A textúrákat képek segítségével tudjuk az OpenGL-nek átadni, általában bitmap (BMP) vagy JPG képeket szoktunk használni, de bármilyen formátumú kép felhasználható, sőt lehetőségünk van mozgóképek (pl. AVI) átadására is, és ebből textúrát készíteni (például egy szobabelső modellezésekor a tévéképernyőre textúraként feltehetünk egy filmet).



1. ábra
A utahi teáskanna textúrás képe

A textúra – amint a kép is – geometriailag egy téglalap alakú terület, amelyet tetszőleges alakú poligonokra, poligonhálókra rá tud húzni a rendszer. A ráhelyezést a modelltérben adjuk meg, így a textúrákra is hatnak a transzformációk (pl. perspektíva, forgatás stb.). A ráhelyezést úgy tudjuk megadni, hogy a vertex koordináták mellett megadjuk a textúra koordinátákat is.

Textúrázáshoz a következő lépéseket kell megtenni:

- engedélyezni kell a textúraleképzést;
- létre kell hozni egy textúraobjektumot és hozzá kell rendelni a textúrát;
- szűrővel meg kell adni, hogy a textúrát hogyan alkalmazza a rendszer;
- meg kell rajzolni a textúrázandó objektumot és meg kell adni a textúra koordinátákat.

A textúraleképzés engedélyezését a `glEnable()` paranccsal tehetjük meg, ha a `GL_TEXTURE_1D`, `GL_TEXTURE_2D` vagy `GL_TEXTURE_3D` szimbolikus konstansokkal hívjuk meg. Általában 2D textúrákat használunk, az 1D textúrák a vonalstílusok, a 3D textúrák pedig egymás mögé helyezett 2D textúrák, amelyek segítségével mélységet tudunk érzékeltetni. Ezeket használja a CT (*Computed Tomography*), MRI (*Magnetic*

Resonance Imaging), vagy 3D textúrák segítségével jelenítjük meg a kőzetrétegeket, bányákat, barlangokat stb.

Itt a 2D textúrákat mutatjuk be, a parancsokat értelemszerűen kell használni 1D és 3D textúrák esetében (2D helyett 1D vagy 3D írandó).

A textúrát egy pixeles kép alapján készíthetjük el, szükségünk van tehát egy olyan rutinra, amely beolvas pl. egy BMP állományt és feldolgozza azt (kiolvassa és a rendelkezésünkre bocsátja a pixeladatokat).

Kétdimenziós textúrát hoz létre a

```
void glTexImage2D(GLenum target, GLint level,
    GLint internalformat, GLsizei width, GLsizei height,
GLint border, GLenum format, GLenum type, const GLvoid *pixels);
```

parancs. A `target` paraméter értéke `GL_TEXTURE_2D` vagy `GL_PROXY_TEXTURE_2D` lehet. A `level` paramétert akkor kell használni, ha a textúrát több felbontásban is szeretnénk tárolni, különben értéke 0. Az `internalformat` a textúrában használatos színkomponenseket határozza meg. Értéke 1, 2, 3, 4 vagy a következő szimbolikus konstansok valamelyike lehet: `GL_ALPHA`, `GL_ALPHA4`, `GL_ALPHA8`, `GL_ALPHA12`, `GL_ALPHA16`, `GL_LUMINANCE`, `GL_LUMINANCE4`, `GL_LUMINANCE8`, `GL_LUMINANCE12`, `GL_LUMINANCE16`, `GL_LUMINANCE_ALPHA`, `GL_LUMINANCE4_ALPHA4`, `GL_LUMINANCE6_ALPHA2`, `GL_LUMINANCE8_ALPHA8`, `GL_LUMINANCE12_ALPHA4`, `GL_LUMINANCE12_ALPHA12`, `GL_LUMINANCE16_ALPHA16`, `GL_INTENSITY`, `GL_INTENSITY4`, `GL_INTENSITY8`, `GL_INTENSITY12`, `GL_INTENSITY16`, `GL_R3_G3_B2`, `GL_RGB`, `GL_RGB4`, `GL_RGB5`, `GL_RGB8`, `GL_RGB10`, `GL_RGB12`, `GL_RGB16`, `GL_RGBA`, `GL_RGBA2`, `GL_RGBA4`, `GL_RGB5_A1`, `GL_RGBA8`, `GL_RGB10_A2`, `GL_RGBA12`, vagy `GL_RGBA16`. A `width` és a `height` a textúra szélességét és magasságát jelentik, a `border` a textára határának szélességét adja meg, értéke 0 vagy 1 lehet. A `width` és a `height` értékek 2^w+2·`border`, 2^h+2·`border` alakúak kell hogy legyenek, ahol w és h természetes számok. A `format` a pixel-adatok formátuma (`GL_COLOR_INDEX`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_RGB`, `GL_RGBA`, `GL_LUMINANCE`, vagy `GL_LUMINANCE_ALPHA`), a `type` a pixel-adatok típusa (`GL_UNSIGNED_BYTE`, `GL_BYTE`, `GL_BITMAP`, `GL_UNSIGNED_SHORT`, `GL_SHORT`, `GL_UNSIGNED_INT`, `GL_INT`, vagy `GL_FLOAT`), a `pixels` pedig a képadatokra mutató pointer.

A képbuffer tartalmából is létrehozható textúra a

```
void glCopyTexImage2D(GLenum target, GLint level,
 GLenum internalformat, GLint x, GLint y, GLsizei width,
              GLsizei height, GLint border);
```

parancs segítségével. A paraméterek ugyanazok, mint az előbb bemutatottak, az `x` és az `y` a kimásolandó pixeltömb bal alsó sarkának a koordinátái.

A textúrákhoz pozitív egész neveket rendelhetünk. A

```
void glGenTextures(GLsizei n, GLuint *textures);
```

paranccsal `n` darab használaton kívüli nevet generálhatunk, ezeket a `textures` tömbben tárolja az OpenGL.

Egy tetszőleges nevet lekérdezhetünk a

```
GLboolean glIsTexture(GLuint name);
```

paranccsal. A parancs `GL_TRUE`-t térít vissza, ha a `name` egy létező textúra név, különben `GL_FALSE`-t.

A textúraneveket hozzá kell rendelni a textúrákhoz. Ezt tehejük meg a

```
void glBindTexture(GLenum target, GLuint name);
```
parancs segítségével. A `target` értéke `GL_TEXTURE_1D`, `GL_TEXTURE_2D` vagy `GL_TEXTURE_3D` lehet, a `name` pedig a textúra neve. Amikor először hívjuk meg a parancsot, akkor leköti a nevet egy alapértelmezett textúrával, majd a textúra létrehozása után feltölti a lefoglalt részt a valós adatokkal. Ha nem először hívjuk meg a parancsot, akkor aktuálissá teszi a `name`-mel hivatkozott textúrát az összes többi közül. Ha 0-val hívjuk meg a parancsot, az alapértelmezett textúra lesz az aktuális.

Textúraneveket a

```
void glDeleteTextures(GLsizei n, const GLuint* names);
```

paranccsal törölhetünk.

OpenGL-ben (a GLU szintjén) lehetőség van *mip-map-technika* megvalósítására is. Az

```
int gluBuild1DMipmaps(GLenum target, GLint components,
 GLint width, GLenum format, GLenum type, void *data);
```

illetve

```
int gluBuild2DMipmaps(GLenum target, GLint components,
GLint width, GLint height, GLenum format, GLenum type,
                      void *data);
```

parancsok segítségével 1D illetve 2D *mip-maps* textúraképek generálhatók.

A *mip-map-technika*nak a lényege, hogy amikor a betöltött bitmap képből a textúrát generálja a rendszer, az OpenGL több változatot is elkészít belőle (a `gluScaleImage` segítségével), különféle részletességi szinttel és ezek közül fog válogatni a távolság függvényében.

Képzeljünk el, mondjuk egy focipályát, amely meglehetősen nagy. Az egyik sarkában álló játékos nem fogja látni külön-külön a pálya másik végében az összes fűszálat. Felesleges tehát nagy felbontású, részletes textúrát használni ott, csak nagyjából látszanak a képek és csak a renderelést lassítják a méretük miatt. Viszont a hozzá közel lévő részeken élesnek kell lenni a képnek. A mip-map-technika a távolság arányában többféle részletességi szintű textúrát alkalmaz.

A `target` paraméter az első parancs esetében `GL_TEXTURE_1D`, a másodikéban pedig `GL_TEXTURE_2D`. A `components` a színkomponensek számát jelenti (1, 2, 3 vagy 4), a `width`, illetve a `height` a kép szélessége és magassága, a `format` a pixeladatok formátuma (`GL_COLOR_INDEX`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_RGB`, `GL_RGBA`, `GL_LUMINANCE`, vagy `GL_LUMINANCE_ALPHA`), a `type` a pixeladatok típusa (`GL_UNSIGNED_BYTE`, `GL_BYTE`, `GL_BITMAP`, `GL_UNSIGNED_SHORT`, `GL_SHORT`, `GL_UNSIGNED_INT`, `GL_INT`, vagy `GL_FLOAT`), a `data` pedig a képadatokra mutató pointer.

A téglalap alakú textúrákat az OpenGL ráfeszíti a nem téglalap alakú objektumokra, és ezeket együtt is transzformálja a színtér objektumaival, ezért gyakran megtörténik, hogy egy képpixelnek nem csak egy textúrapixel fog megfelelni. Ezekben az esetekben *filterezni* kell a textúrát. A *textúrafilterezés* lényege, hogy megadhatjuk, kicsinyítésnél (távol van) és nagyításnál (közel került) hogyan viselkedjenek a textúrák, mennyivel kell nagyítani vagy kicsinyíteni a textúra pixeleket, hogy ráférjenek az objektum képének a pixeleire. Ugyancsak ezen paraméterek segítségével adhatjuk meg, hogy a textúra ismétlődjön a felületen (a felületet kitöltjük a textúraképpel úgy, hogy egymás mellé másoljuk többször ugyanazt a képet), vagy csak egyszerűen feszüljön rá a felületre. A

```
void glTexParameter{i f}{# v}(GLenum target,
                GLenum pname, T param);
```

parancs segítségével a textúrafilterezéshez szükséges szűrőket adhatjuk meg. A `target` értéke `GL_TEXTURE_1D`, `GL_TEXTURE_2D` vagy `GL_TEXTURE_3D` lehet, a `pname` éstékei a `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAG_FILTER`, `GL_TEXTURE_WRAP_S`, `GL_TEXTURE_WRAP_T` lehetnek.

Az első a kicsinyítéshez, a második a nagyításhoz, a harmadik és negyedik pedig a textúra *s* és *t* koordinátája szerinti ismétléshez szükséges. Kicsinyítés esetén összesen hatféle textúrafilterezésre van lehetőségünk (`param`), ezek közül a `GL_NEAREST` a legrosszabb minőségű, de a leggyorsabb, míg a `GL_LINEAR` a legjobb minőségű. A mipmap-technika igyekszik egyensúlyt találni a kettő között. Ennek négy változata van, `GL_xx_MIPMAP_xx` alakban, ahol `xx` vagy `LINEAR` vagy `NEAREST`.

Nagyítás esetén a `GL_NEAREST` és a `GL_LINEAR` közül választhatunk.

A másik kettő esetében pedig a `GL_CLAMP` vagy a `GL_REPEAT` a lehetőségek.

Textúrák használata esetén azt is el tudjuk érni, hogy az objektum színét keverjük a textúra színével. Erre a

```
void glTexEnv{i f}{# v}(GLenum target, GLenum pname,
                        GLfloat param);
```

parancsot használjuk. A `target` értéke `GL_TEXTURE_ENV` lehet, a `pname` értéke `GL_TEXTURE_ENV_MODE` vagy `GL_TEXTURE_ENV_COLOR` lehet.

Ha az érték `GL_TEXTURE_ENV_MODE`, a `param` értékei `GL_MODULATE`, `GL_DECAL`, `GL_BLEND`, vagy `GL_REPLACE` lehetnek, ezek írják elő, hogy a rendszer a textúrát hogyan kombinálja a feldolgozandó pixel színével.

Ha a `pname` értéke `GL_TEXTURE_ENV_COLOR`, akkor a `param` az RGBA komponenseket tartalmazó tömb címe lesz, de ezeket csak akkor fogja használni a rendszer, ha értelmezett a `GL_BLEND` is.

Ha egy objektumot textúrázni akarunk, a vertexek mellett meg kell adnunk a csúcspontok textúrakoordinátáit is, amelyek azt mondják meg, hogy az adott vertexhez melyik textúra pixel tartozik. A textúrának 1, 2, 3 vagy 4 koordinátája lehet, ezeket az *s*, *t*, *r*, *q* betűkkel jelöljük. A *q* koordináta értéke általában 1 (homogén koordináta), a többi alapértelmezett értéke 0. Az aktuális textúrakoordinátákat a

```
void glTexCoord{1 2 3 4}{s i f d}{# v}(T coords);
```

parancs segítségével adhatjuk meg.

A megadott textúrakoordinátákat a rendszer megszorozza a textúramátrixszal.

A következő példaprogram bemutatja, hogyan tudunk betölteni és használni két textúrát.

```
1.   #include <stdlib.h>
2.   #include <GL/glut.h>
3.   #include "RgbImage.h" // BMP állomány beolvasása
4.
5.   static GLuint textureName[2]; // a textúrák
6.
7.   // textúra beolvasása és inicializálása
8.   void LoadTextureFromFile(char* filename)
9.   {
10.    glClearColor (0.0, 0.0, 0.0, 0.0);
11.    glShadeModel(GL_FLAT);
12.    glEnable(GL_DEPTH_TEST);
13.    RgbImage image(filename);
14.    glTexParameteri(GL_TEXTURE_2D,
15.                    GL_TEXTURE_WRAP_S,
```

```
16.                     GL_REPEAT);
17.    glTexParameteri(GL_TEXTURE_2D,
18.                    GL_TEXTURE_WRAP_T,
19.                    GL_REPEAT);
20.    glTexParameteri(GL_TEXTURE_2D,
21.                    GL_TEXTURE_MAG_FILTER,
22.                    GL_NEAREST);
23.    glTexParameteri(GL_TEXTURE_2D,
24.                    GL_TEXTURE_MIN_FILTER,
25.                    GL_NEAREST);
26.    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
27.                      image.GetNumCols(),
28.                      image.GetNumRows(),
29.                      GL_RGB, GL_UNSIGNED_BYTE,
30.                      image.ImageData());
31.  }
32.
33.  // a textúrák inicializálása
34.  void InitTexture(char* filenames[])
35.  {
36.    glGenTextures(2, textureName);
37.    for(int i=0; i<2; ++i)
38.    {
39.      glBindTexture(GL_TEXTURE_2D, textureName[i]);
40.      LoadTextureFromFile(filenames[i]);
41.    }
42.  }
43.
44.  void display()
45.  {
46.    glClear(GL_COLOR_BUFFER_BIT |
47.            GL_DEPTH_BUFFER_BIT);
48.    glEnable(GL_TEXTURE_2D);
49.    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
50.              GL_MODULATE);
51.    glRotatef(31, 1, 1, 0);
52.    // a textúrák használata
53.    glBindTexture(GL_TEXTURE_2D, textureName[0]);
54.    glBegin(GL_QUADS);
55.      glTexCoord2f(0.0, 0.0);
56.      glVertex3f(-2.0, -2.0, -0.5);
57.      glTexCoord2f(0.0, 1.0);
58.      glVertex3f(-2.0, 2.0, -0.5);
59.      glTexCoord2f(1.0, 1.0);
60.      glVertex3f(2.0, 2.0, -0.5);
61.      glTexCoord2f(1.0, 0.0);
62.      glVertex3f(2.0, -2.0, -0.5);
63.    glEnd();
64.    glBindTexture(GL_TEXTURE_2D, textureName[1]);
65.    GLUquadricObj* sphere;
66.    sphere = gluNewQuadric();
67.    gluQuadricOrientation(sphere, GLU_OUTSIDE);
68.    gluQuadricNormals(sphere, GLU_SMOOTH);
69.    gluQuadricTexture(sphere, GL_TRUE);
70.    gluSphere(sphere, 0.5, 20, 20);
71.    gluDeleteQuadric(sphere);
```

```
72.    glFlush();
73.    glDisable(GL_TEXTURE_2D);
74.  }
75.
76.  void ResizeWindow(int w, int h)
77.  {
78.    float viewWidth = 2.2;
79.    float viewHeight = 2.2;
80.    glViewport(0, 0, w, h);
81.    h = (h==0)?1:h;
82.    w = (w==0)?1:w;
83.    glMatrixMode(GL_PROJECTION);
84.    glLoadIdentity();
85.    if(h < w) viewWidth *= (float)w/(float)h;
86.    else viewHeight *= (float)h/(float)w;
87.    glOrtho(-viewWidth, viewWidth, -viewHeight,
88.             viewHeight, -2.0, 2.0);
89.    glMatrixMode(GL_MODELVIEW);
90.    glLoadIdentity();
91.  }
92.
93.  void keyboard (unsigned char key, int x, int y)
94.  {
95.    switch (key)
96.    {
97.      case 27:
98.        exit(0);
99.        break;
100.     default:
101.       break;
102.   }
103. }
104.
105. char* filenameArray[2] =
106. {
107.   "fu.bmp",
108.   "fa.bmp",
109. };
110.
111. int main(int argc, char** argv)
112. {
113.   glutInit(&argc, argv);
114.   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
115.                       GLUT_DEPTH);
116.   glutInitWindowSize(500, 400);
117.   glutInitWindowPosition(100, 100);
118.   glutCreateWindow("2 textura");
119.   InitTexture(filenameArray);
120.   glutDisplayFunc(display);
121.   glutReshapeFunc(ResizeWindow);
122.   glutKeyboardFunc(keyboard);
123.   glutMainLoop();
124.   return 0;
125. }
```