Texture mapping

Need more detail? Paste a picture!

- Texture: 1-D, 2-D and 3-D image
- Texture coordinates: Per vertex 1, 2 or 3 extra coordinates that point in the texture

H&B 18: 579-601

1D Texture mapping

1D texture mapping: Color scale for visualization

H&B 18-2:580-585

1D Texture mapping 2

Interpolation of colors gives poor results...

H&B 18-2:580-585

1D Texture mapping 3

Coloring with 1D texture-map: better result

H&B 18-2:580-585

2D Texture mapping

H&B 10-17:629-634
Forward mapping

Forward mapping: project texture to pixel

\[
\begin{align*}
(s(u), t(v)) & \\
(x, y) & \\
(s_0, t_0) & \\
(s_1, t_1) & \\
(s_2, t_2) &
\end{align*}
\]

Backward mapping

Backward mapping: lookup texture per pixel

\[
\begin{align*}
(s(u), t(v)) & \\
(x, y) & \\
(s_0, t_0) & \\
(s_1, t_1) & \\
(s_2, t_2) &
\end{align*}
\]

2D Texture mapping triangles

- Per vertex: specify position \((x, y)\) and texture coordinates \((s, t)\)
- Texture coordinates are interpolated during scan conversion and used to look-up the color.

3D Texture mapping

3D texture mapping:
- Texture map is volume (stack of bitmaps): memory intensive
- Per vertex \((s, t, r)\) coordinates
- Applications:
  - medical 3D images (plane through scan)
  - solid materials (wood)

Procedural texturing

- Instead of using a 1-, 2-, or 3-D image:
  Define function that returns color dependent on value of \(s, (s, t)\) or \((s, t, r)\)
- Simple: chessboard
- Advanced: wood, marble, etc.

Perlin Procedural texturing

Images Ken Buckner
OpenGL 1D Texture Mapping 1
First, make sure texture mapping is enabled:
```c
glEnable(GL_TEXTURE_1D);
```
Load the 1D texture map:
```c
GlTexImage1D(GL_TEXTURE_1D, // here comes a 1D map
0,             // 0: not part of larger array
GL_RGBA,       // we use RGBA colors
texColors,    // the number of colors (use power of 2)
0,             // no border
GL_RGBA,       // order of bytes in color
GL_UNSIGNED_BYTE, // components color: unsigned bytes
lineTexArray);  // array with color values
```

OpenGL 1D Texture Mapping 2
Use the 1D color map during rendering:
```c
glBegin(...);
glTexCoord1f(0.2); // one coordinate per vertex
 glVertex3fv(p1);
glTexCoord1f(0.8);
 glVertex3fv(p2);
...
glEnd();
```

OpenGL 2D Texture Mapping 1
First, make sure texture mapping is enabled:
```c
glEnable(GL_TEXTURE_1D);
```
Load the 2D texture map:
```c
GlTexImage2D(GL_TEXTURE_1D, // here comes a 2D map
0,             // 0: not part of larger array
GL_RGBA,       // we use RGBA colors
texwidth, // width of texture map (use power of 2)
texHeight, // height of map (use power of 2)
0,             // no border
GL_RGBA,       // order of bytes in color
GL_UNSIGNED_BYTE, // components color: unsigned bytes
surfTexArray);  // array with color values
```

OpenGL 2D Texture Mapping 2
Use the 2D color map during rendering:
```c
glBegin(...);
glTexCoord1f(0.2, 0.8); // two coordinates per vertex
 glVertex3fv(p1);
glTexCoord1f(0.8, 0.8);
 glVertex3fv(p2);
...
glEnd();
```

OpenGL texture parameters 1
Generic call:
```c
glTexparameter*(target, parameter, value(s));
```
* = iv or fv
```c
target = GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D
```
Some useful parameters:
```c
GL_TEXTURE_WRAP_S  
GL_TEXTURE_WRAP_T  
GL_TEXTURE_MIN_FILTER  
GL_TEXTURE_MAG_FILTER
```

OpenGL texture parameters 2
```c
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, V);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, V);
```
V = GL_REPEAT: Use only fractional part of texture coordinate
V = GL_CLAMP: Clamp texture coordinate to [0, 1] range

All coordinates: (s,t) texture coordinates
Quadrilateral
OpenGL texture parameters 3

- `glTexParameteri(GL_TEXTURE_2D, GL_WRAP_S, V);`
- `glTexParameteri(GL_TEXTURE_2D, GL_WRAP_T, V);`

V = GL_REPEAT: Use only fractional part of texture coordinate
V = GL_CLAMP: Clamp texture coordinate to [0, 1] range

GL_REPEAT, GL_REPEAT
GL_CLAMP, GL_CLAMP

OpenGL texture parameters 4

- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, V);`
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, V);`

Texture                 Pixel
V = GL_NEAREST
V = GL_LINEAR

Can be set independently for enlarged textures (MIN_FILTER) and shrunk textures (MAG_FILTER).

OpenGL texture color

- `glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, V);`

V = GL_MODULATE: Final color is product of current color and texture color
Default value.
Using texture on black object has no effect!
V = GL_REPLACE: Final color is texture color
Simplest to use.
V = GL_DECAL, GL_BLEND: If you want to do something special.

OpenGL named textures 1

Multiple textures simultaneously in one scene?
Using `glTexImage2D` multiple times per frame is expensive.
Solution: named textures.

Initially bind textures to symbolic id’s:

```c
myTextureInit(); // Taken care of in template assignment
{
  glBindTexture(GL_TEXTURE_2D, head_texture_id);
  glTexImage2D(..., head_texture);
  glBindTexture(GL_TEXTURE_2D, torso_texture_id);
  glTexImage2D(..., torso_texture);
  glBindTexture(GL_TEXTURE_2D, bricks_texture_id);
  glTexImage2D(..., bricks_texture);
}
```

OpenGL named textures 2

Multiple textures simultaneously in one scene?
Using `glTexImage2D` multiple times per frame is expensive.
Solution: named textures.

During drawing: refer to symbolic id’s:

```c
myDraw();
{
  glBindTexture(GL_TEXTURE_2D, head_texture_id);
  myDrawHead();
  glBindTexture(GL_TEXTURE_2D, torso_texture_id);
  myDrawTorso();
  glBindTexture(GL_TEXTURE_2D, bricks_texture_id);
  myDrawBricks();
}
```

Texture mapping hints

Use texture maps with dimensions that are a power of 2 (128×128, 128×256, 512×512, etc.);

Try to minimize distortion, i.e., aim at using dimensions in texture space that are proportional to dimensions in world space;

Use: `glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)`;

You can use multiple pictures in one texture map, and control via texture coordinates which picture you use.
Finally

• How about hidden surfaces?