Texture Mapping

- Enhancing rendering realism by adding surface textures to faces of mesh objects.
- Texture image: a function $f(s,t) : [0,1] \times [0,1] \rightarrow D$
  where $D$ is the color domain. It is often represented as a 2D image (bitmap), but can also be computed through a procedure or a mathematical function.
- Bitmap texture:
  - **Texel**: each pixel unit of the texture image.
  - Can be either a 2D array of texels or 1D array texels in a row-major order.
- Procedural texture: The texture function $f(s,t)$ is computed through a procedure (e.g. fake sphere).
A texel in a $n \times m$ texture image:

$\text{texture}[i][j]$ or $\text{texture}[j \cdot n + i]$

$(i = 0, 1, \ldots, n-1; j = 0, 1, \ldots, m-1)$

Access $f(s, t)$ $(0 \leq s, t \leq 1)$

- Nearest neighbor:
  $f(s, t) = \text{texture}[(\text{int})(s \cdot (n-1) + 0.5)][(\text{int})(t \cdot (m-1) + 0.5)]$

- Bilinear interpolation

\[
\begin{align*}
&u = s \cdot (n-1) - (\text{int})(s \cdot (n-1)), \quad v = t \cdot (m-1) - (\text{int})(t \cdot (m-1)) \\
&C_q = (1-u) \cdot C_a + u \cdot C_r, \quad C_s = (1-u) \cdot C_c + u \cdot C_d, \\
&C_p = (1-v) \cdot C_a + v \cdot C_r
\end{align*}
\]
Mapping texture to surface

- Each surface point has a corresponding texture coordinate, (s,t), within the texture image.
- In OpenGL, each polygon vertex can have a texture coordinate specified by:
  \[
  \text{glTexCoord2f}(s,t)
  \]
- Texture coordinate is a state variable (similar to normal)

```c
glBegin(GL_QUADS);
  glTexCoord2f(0.0, 0.0); glVertex3f(1.0, 2.5, 1.5);
  glTexCoord2f(0.0, 0.6); glVertex3f(1.0, 3.7, 1.5);
  glTexCoord2f(0.8, 0.6); glVertex3f(2.0, 3.7, 1.5);
  glTexCoord2f(0.8, 0.0); glVertex3f(2.0, 2.5, 1.5);
glEnd();
```

Texture Definition and Rendering

- A texture is defined by:
  \[
  \text{glTexImage2D}(GL\_TEXTURE\_2D, \text{level}, \text{components}, \text{width}, \text{height}, \text{border}, \text{format}, \text{type}, \text{image})
  \]
  Where "image" is a 1D array of size "width \times height \times components" and type "type", storing the texture image with color format "format".
- For each surface point to be drawn to a pixel, its texture coordinate, (s,t), is first computed, and corresponding color in the texture image, f(s,t), will then be found, and drawn to the pixel.
- Computing texture coordinate: bilinear interpolation - only work well with parallel projection.
Perspective Interpolation

- With perspective projection, equal-distance steps on screen space do not correspond to equal-distance steps in texture space -- Bilinear interpolation can lead to texture distortion.
- Perspective interpolation: given line segment $AB$ and its perspective projection $ab$. Let the 4th components of $a$'s and $b$'s homogeneous coordinates be $a_4$ and $b_4$. Then the texture coordinates at a point $p(f) = (1-f)\cdot a + f \cdot b$ will be:

$$s_p = \frac{(1-f) \cdot a_4 + f \cdot b_4}{(1-f) \cdot a_4 + f \cdot b_4 + 1}$$

$$t_p = \frac{(1-f) \cdot a_4 + f \cdot b_4}{(1-f) \cdot a_4 + f \cdot b_4 + 1}$$
Interpolating z-values

\[
Z_t = Z_1 + t(Z_2 - Z_1)
\]

\[
\frac{1}{Z_t} = \frac{1}{Z_1} + \left( \frac{1}{Z_2} - \frac{1}{Z_1} \right) s
\]

Interpolating attribute values

\[
I_t = I_1 + t(I_2 - I_1)
\]

\[
\frac{I_t}{Z_t} = \frac{I_1}{Z_1} + s \left( \frac{I_2}{Z_2} - \frac{I_1}{Z_1} \right)
\]
Texture modulation - how texture colors are used?

- **Replace (Glowing)**: use texture color to directly replace the pixel color without computing shading.
  
  ```c
  glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
  ```

- **Modulate (Painting)**: Multiple the texture colors to the diffuse and ambient components in shading formula.
  
  ```c
  glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  I = texture(s,t)[I_a\rho_n + I_d\rho_d L] + I_g\rho_g P'
  ```

- There several other modulation modes, such as:
  
  - GL_BLEND,
  - GL_ADD,
  - GL_DECAL

Texture parameters

- Many texture mapping parameters can be set by:
  
  ```c
  glTexParameter (GL_TEXTURE_2D, pname, param)
  ```

- **pname = GL_TEXTURE_MIN_FILTER**: minifying function options, determining how a pixel color is computed if the pixel is mapped to an area greater than one texel. (LINEAR, NEAREST, MIPMAP, etc).

- **pname = GL_TEXTURE_MAG_FILTER**: magnification function options, determining how a pixel color is computed if the pixel is mapped to an area smaller than one texel. (NEAREST, LINEAR, etc).

- **pname = GL_TEXTURE_WRAP_S or GL_TEXTURE_WRAP_T**: setting wrapping parameters in s or t direction. (CLAMP, REPEAT, etc.)

- **pname = GL_TEXTURE_BORDER_COLOR**, etc.
Texture objects

- A texture object stores texture data so that it can be used later. Many texture objects can be generated, each has its own texture data and parameters.
- Generating n names (system generated positive integers) in nameArray for texture objects:
  \[ \text{glGenTextures}(n, \text{nameArray}) \]
- \( \text{glBindTexture}(\text{GL\_TEXTURE\_2D}, \text{name}) \):
  - Creates a texture object with the given name, and binds the texture object to the next texture data to be created using \( \text{glTexImage2D} \)
  - Make this texture object current, so that subsequent texture operations are applied to this texture object.
  - Can be called multiple times, but only the first time creates the texture object.

Other OpenGL texture functions

- Texture image files: texture file reading functions are needed to read texture images from image files. It depends on image file format, e.g. jpeg, gif, ppm, bmp, etc.
- \( \text{glEnable}(\text{GL\_TEXTURE\_2D}) \) : Enable 2D texture mapping
- \( \text{glBindTexture}(\text{GL\_TEXTURE\_2D}, \text{name}) \):
  (1) Assign a name to the next texture to be created using \( \text{glTexImage2D} \); (2) bind the named texture to the current 2D texture (make active).
- \( \text{GlHint}(\text{GL\_PERSPECTIVE\_CORRECTION\_HINT}, \text{GL\_NICEST}) \) : requests the use of perspective interpolation in texture rendering.
- Example: rotating cube with 6 texture images.
Example: rotating cube with 6 texture images

Automatically generated texture coordinates

```c
glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable( GL_TEXTURE_GEN_S);
glEnable( GL_TEXTURE_GEN_T);
glEnable( GL_TEXTURE_2D);
```
Portable Pixmap (PPM) image files

- PPM file has the simplest file format, sometimes also called raw format.
- The file has a header followed by image data.
  - Any line starting with a "#" is a comment line
  - First line is always: "P6"
  - Second line has two numbers: the width and height of the image.
  - The next line has one single integer number: the maximum value of any RGB color component.
  - The image data is in binary form, containing the byte stream of the pixel values stored in a row by row order. Each pixel has three bytes (its RGB values).
  - fread() and fwrite() are the best ways to read and write the binary portion of the image data.

OpenGL Quadrics

- Quadrics: mathematical surfaces represented by 2nd degree polynomial functions: cylinder, cone, sphere, etc.
  \[ a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5xz + a_6yz + a_7x + a_8y + a_9z + a_{10} \]
- OpenGL quadrics (quadric object): quadrics with certain drawing properties. A quadrics (with drawing properties) can be used to draw various types of quadric surfaces such as cylinder, cone and sphere.
- Using quadrics: a quadric object is first created, assigned drawing properties, and then applied to various quadric surfaces for rendering.
  \[ GLUquadricObj *gluNewQuadric(); \]
- Drawing properties: Draw style; Normal; Orientation; Texture coordinates.
Drawing properties

- Draw style: filled, wireframe, etc.
  `gluQuadricDrawStyle (GLUquadricObj *quadObj, GLenum style);`
  style: GLU_FILL (default), GLU_LINE, GLU_POINT, GLU_SILHOUETTE.

- Normal: how normals are generated.
  `gluQuadricNormals (GLUquadricObj *quadObj, GLenum normalMode);`
  normalMode: GLU_NONE (no normal), GLU_FLAT (normals for polygons only), GLU_SMOOTH (normals for vertices).

Drawing properties (2)

- Orientation: Normal pointing to outside (default) or inside.
  `gluQuadricOrientation (GLUquadricObj *quadObj, GLenum orientation);`
  orientation: GLU_OUTSIDE (default), GLU_INSIDE.

- Texture Coordinates: generate texture coordinates or not.
  `gluQuadricTexture (GLUquadricObj *quadObj, GLboolean useTextureCoords);`
  useTextureCoords: GL_TRUE, GL_FALSE (default).

- Cleaning up:
  `gluDeleteQuadric (GLUquadricObj *quadObj);`
Quadric surfaces

- Disk: a flat circle with possibly a hole in the middle.
  
  \[ \text{gluDisk (quadObj, innerRadius, outerRadius, slices, loops)} \]
  
  slices, loops: number of subdivisions for polygon approximation.

- Partial disk:
  
  \[ \text{gluPartialDisk (quadObj, innerRadius, outerRadius, slices, loops, startAngle, sweepAngle)} \]

- Sphere:
  
  \[ \text{gluSphere (quadObj, radius, slices, stacks)} \]
  
  slices, stacks: for polygon approximation.
  
  Texture coordinates:
  
  \( s \): rotation angle around Z-axis
  
  \( t \): Z coordinate

Quadric surfaces (2)

- Cylinders (and cones):
  
  \[ \text{gluCylinder (quadObj, baseRadius, topRadius, height, slices, stacks)} \]
  
  when either baseRadius or topRadius is 0, it becomes a cone.
  
  Texture coordinates:
  
  \( s \): angle
  
  \( t \): height (Z-coordinates).
Drawing quadrics

- A quadric object is first created, assigned drawing properties, and then applied to various quadric surfaces for rendering.
  - GLUquadricObj *gluNewQuadric();
- Draw style: GLU_FILL (default), GLU_LINE, etc.
  - gluQuadricDrawStyle (GLUquadricObj *quadObj, Glenum style);
- Normal: GLU_NONE, GLU_FLAT, GLU_SMOOTH
  - gluQuadricNormals (GLUquadricObj *quadObj, GLUenum normalMode);
- Orientation: GLU_OUTSIDE (default) or GLU_INSIDE.
  - gluQuadricOrientation (GLUquadricObj *quadObj, GLUenum orientation);
- Cleaning up: gluDeleteQuadric (GLUquadricObj *quadObj);

Quadric surface texture mapping

- Setting quadrics texture coordinates: GL_TRUE or GL_FALSE
  - gluQuadricTexture (GLUquadricObj *quadObj, GLboolean useTextureCoords);
- Sphere: gluSphere (quadObj, radius, slices, stacks)
  - slices, stacks: for polygon approximation.
  - texture coordinates automatically generated.
- Cylinders (and cones):
  - gluCylinder (quadObj, baseRadius, topRadius, height, slices, stacks)
  - texture coordinates automatically generated
1D and 3D textures

- **1D texture**: can be considered a 2D texture with a height 1. It is often used for drawing color bands or curves.

  ```c
  unsigned char texture[128];
  glTexImage1D(GL_TEXTURE_1D, level, components, width, border, format, type, texture);
  ```

- **3D texture**: can be considered a stack (depth) of 2D textures. It is often used for medical imaging visualization, solid texturing, and volume rendering.

  ```c
  unsigned char texture[width*height*depth*3];
  glTexImage3D(GL_TEXTURE_3D, level, components, width, height, depth, border, format, type, texture);
  ```

3D texture mapping examples
Curved surface texture mapping

- Assuming a curved surface is represented by a parametric surface: \( S(u,v) = (x(u,v), y(u,v), z(u,v)) \), and approximated by a polygon mesh.
- The main task is to find the proper texture coordinates for the vertices of the polygon mesh.

![Texture function diagram]

Cylindrical surfaces

- Parametric representation:
  \[
  \begin{align*}
  x(\theta, h) &= r \cos(\theta) \\
  y(\theta, h) &= r \sin(\theta) \\
  z(\theta, h) &= h
  \end{align*}
  \]

- Mapping function:
  \[
  [0,1] \times [0,1] \rightarrow [\theta_a, \theta_b] \times [h_a, h_b]
  \]
  \[
  \begin{align*}
  s &= (\theta - \theta_a) / (\theta_b - \theta_a) \\
  t &= (h - h_a) / (h_b - h_a)
  \end{align*}
  \]
Surfaces of revolution

- Parametric representation
  \[
  \begin{align*}
  x(\theta, v) &= X(v) \cos(\theta) \\
  y(\theta, v) &= X(v) \sin(\theta) \\
  z(\theta, v) &= Z(v)
  \end{align*}
  \]

- Mapping function:
  \[
  \begin{align*}
  s &= (\theta - \theta_a)/(\theta_b - \theta_a) \\
  t &= (v - v_a)/(v_b - v_a)
  \end{align*}
  \]

- Using enclosing cylinder: the texture is first mapped to the enclosing cylinder, and then projected back to the surface along the cylinder’s normal direction.

Spherical surfaces

- Parametric representation
  \[
  \begin{align*}
  x(\theta, \phi) &= r \cos(\phi) \cos(\theta) \\
  y(\theta, \phi) &= r \cos(\phi) \sin(\theta) \\
  z(\theta, \phi) &= r \sin(\phi)
  \end{align*}
  \]

- Linear mapping:
  \[
  \begin{align*}
  s &= (\theta - \theta_a)/(\theta_b - \theta_a) \\
  t &= (\phi - \phi_a)/(\phi_b - \phi_a)
  \end{align*}
  \]

- Triangular mapping:
  \[
  \begin{align*}
  (\theta, 0^\circ) &\rightarrow (0.5, 1.0) \\
  (\theta, 0) &\rightarrow (0.0, 0.0) \\
  (\theta, 0) &\rightarrow (1.0, 0.0)
  \end{align*}
  \]
Bump Mapping

- Bump mapping simulates surface roughness by perturbing normal vectors. It works better than mapping a texture of rough surface image (Fig 8.49).
- Normal perturbation: Given a surface \( P(u,v) \), with normal vectors: \( n(u,v) = P_u \times P_v / |P_u \times P_v| \), the perturbed surface is:
  \[
P'(u,v) = P(u,v) + \text{texture}(s(u),t(v)) \cdot n(u,v)
\]
  \[
n'(u,v) = n + \text{texture}_s \cdot (P_u \times n) + \text{texture}_t \cdot (n \times P_v)
\]
  where \( \text{texture}(s,t) \) is a scalar valued perturbation function.
- The rendering process will use the original surface vertices (i.e. \( P(u,v) \)) and the perturbed normals (i.e. \( n(u,v) \)).
The perturbation function can be a mathematical function (e.g., $\sin(au)\sin(bv)$) or a sampled gray level image.
Environmental mapping

- Allowing surrounding/background scene to be seen through reflections on object surfaces (Fig 8.63).
- An environmental texture is mapped to the interior surface of a surrounding sphere or cube, and reflected back to object surfaces (Fig 8.64).

- The environment map is reflected onto the object surfaces:
  Trace a ray from eye to a point P on the object surface, determine the reflection ray, and then trace the reflection ray until it hits the surrounding sphere or cube. The color from environment map at this hit-point will be placed at point P.

Environment mapping – OpenGL solution

- A simplified solution using automatically generated texture coordinates.

```c
// Build the environment map as a texture image
// Generate texture coordinates automatically
glTexGenf (GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGenf (GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
 glEnable (GL_TEXTURE_GEN_S);
 glEnable (GL_TEXTURE_GEN_T);

// bind the environment texture and draw the object
```
Using different resolutions of a texture image for different levels of detail rendering.
- Level-0: the original texture (primary texture). The width and height need to be a power of 2 (e.g. 64x64)
- Level-1: half size of level-0 in both s and t directions. (32x32)
- Level-2: half size of level-1 (16x16)
- Level-3 (8x8); level 4 (4x4); level 5 (2x2); level 6 (1x1).

Defining mipmap:

- `glTexImage2D (GL_TEXTURE_2D, level, GL_RGB, ....)`
  where level = 0, 1, 2, ....

- A GLU mipmap function: generating a series of mipmaps automatically from a primary texture image.
  `gluBuild2DMipmaps (GL_TEXTURE_2D, ....)`
Mipmap filters

- Mipmap minification filters need to be applied when using mipmap textures. E.g.:
  - `GL_NEAREST_MIPMAP_LINEAR`: using nearest mipmap to the polygon resolution, and using `GL_LINEAR` filter.
  - `GL_LINEAR_MIPMAP_NEAREST`: using linear interpolation between the two mipmaps closest to the polygon resolution, and the `GL_NEAREST` filter.

- Example:
  
  ```
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
  gluBuild2DMipmaps (GL_TEXTURE_2D, 3, 64, 64, GL_RGB, GL_UNSIGNED_BYTE, texImage);
  ```