Material Properties

- Depend on lighting factors:
  - $k_a$ - Ambient light coefficient
  - $k_d$ - Diffuse light coefficient
  - $k_s$ - Specular light coefficient
  - $n$ - Surface normal

```gl
glMaterial[fi]( face, property, value);
glMaterial[fi]( face, property, *value);
```

- Define the surface properties of a primitive
- Separate materials allowed for front and back
- `face` is GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK

Material Properties

- Properties and default values:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_AMBIENT</td>
<td>Ambient reflectance, RGBA</td>
</tr>
<tr>
<td>GL_DIFFUSE</td>
<td>Diffuse reflectance, RGBA</td>
</tr>
<tr>
<td>GL_SPECULAR</td>
<td>Specular reflectance, RGBA</td>
</tr>
<tr>
<td>GL_EMISSION</td>
<td>Emitted light intensity</td>
</tr>
<tr>
<td>GL_SHININESS</td>
<td>Specular exponent, [0,128]</td>
</tr>
<tr>
<td>GL_AMBIENT_AND_DIFFUSE</td>
<td>Equivalent to two calls to glMaterial</td>
</tr>
</tbody>
</table>

Materials in OpenGL

- A simpler interface:
  ```gl
  glColorMaterial( face, mode );
  ```
- Specifies that material parameters track the current color.
- Must invoke glEnable(GL_COLOR_MATERIAL)
- Allows a subset of material parameters to be changed for each vertex using only glColor, without calling glMaterial
- If only such a subset of parameters is to be specified for each vertex, glColorMaterial is easier to use than glMaterial

```gl
glColorMaterial( face, mode );
```

- `face`
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK
- `mode` - Specifies which of several material parameters track the current color
  - GL_EMISSION
  - GL_AMBIENT
  - GL_DIFFUSE
  - GL_SPECULAR
  - GL_AMBIENT_AND_DIFFUSE (default)
### Texture Mapping

#### glColorMaterial Example

- When you need to change a single material parameter for most vertices in your scene:

```gl
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
/* now glColor* changes diffuse reflection */
gColor3f(0.2, 0.5, 0.8);
/* draw some objects here */
gColorMaterial(GL_FRONT, GL_SPECULAR);
/* glColor* no longer changes diffuse reflection, *
 now glColor* changes specular reflection */
gColor3f(0.5, 0.0, 0.2);
/* draw other objects here */
gDisable(GL_COLOR_MATERIAL);
```

#### Texture Mapping

- Developed in 1974 by Ed Catmull, currently president of Pixar

- Goal: To make Phong shading less plastic looking

#### Texture Mapping

- Apply a 1D, 2D, or 3D image to geometric primitives
- Uses of texturing:
  - Simulating materials
  - Reducing geometric complexity
  - Image warping
  - Reflections
- 1D textures are images with width but no height, or vice-versa
  - One pixel wide or high
- 2D textures have width and height
- 3D textures describe volumes
  - Used, e.g., in CAT/MRI scans
  - Huge - a 256x256x256 grayscale-alpha texture uses 32MB of memory

#### Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
  - "Complex" textures do not affect geometric complexity
Texture Example

• The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective.

Applying Textures

• Three steps
  - Specify texture
    • Read or generate image
    • Assign to texture
    • Enable texturing
  - Assign texture coordinates to vertices
  - Specify texture parameters
    • Wrapping, filtering, etc.

Texture Mapping - Uses

• Surface color - most common
• Normal vector perturbations (bump mapping)
• Reflection
  - Illumination
  - Specular
  - Diffuse
  - Transparency
  - Environment mapping

What is a texture?

• Two Types:
  - Image file
  - Mathematically generated, e.g., checker board pattern
• Contents:
  - Made up of texels
  - Photos shot straight on
  - E.g., wood grain, marble, patterns

Direct Mapping

• Simplest form of mapping
• Texture coordinates equal polygon coordinates. \( u = x \) and \( v = y \)
• How to fill? Polygon filling
  - Same scanline algorithm as CGI
  - Color data pulled from texture array
Texture Mapping - Direct Mapping

- Problems:
  - Causes texture to be clipped, or entire polygon not filled.
- Solution:
  - Need an algorithm that stretches & shrinks texture over a polygon. Affine mapping comes to the rescue.

Texture Mapping - Affine Mapping

- Stretching and shrinking of the texture is based on the premise of sampling.
- Texture coordinates become necessary
  - Mapping of texture coordinates to polygon coordinates
- Additional calculations required
  - $dxdy, du, dv$ all need to be calculated for each edge. $dudx, dvdx$ for each scanline.
Texture Mapping - Affine Mapping

- Affine mapping can have different problems due to sampling:
  - Small textures on large models/polygons tend to become blocky.
  - Large textures on small models/polygons tend to become grainy.

Texture Mapping - Bump Mapping

- Adds roughness to surfaces
- Quick way to add detail to an object
- Polygon remains physically flat, but appears bumpy

Jim Blinn
Bump Mapping Theory

- If your eyes see light and dark → bumps
- Flat surfaces reflect more light
- Bumpy surfaces reflect less

Surfaces angled upward tend to be brightly lit
Downward inclined surfaces tend to be darker

These are the same image!

Texture Mapping - Bump Mapping

- Perturbing surface normal
- Texture map represents displacements from the normal
- Use perturbed normal in illumination model

Bump Mapping – Height Field Method

- Extension of Phong Shading technique
  - Surface normal interpolated over polygon & that vector is used to calculate brightness of pixel
  - Alter normal vector slightly based on info in bump map
  - Adjust normal vector to change brightness of pixel

Convert bumps on bump map into little vectors (one vector per pixel).

\[
x_{\text{gradient}} = \text{pixel}(x-1, y) - \text{pixel}(x+1, y)
y_{\text{gradient}} = \text{pixel}(x, y-1) - \text{pixel}(x, y+1)
\]

Adjust normal vector of polygon at that point.

\[
\text{New Normal} = \text{Normal} + (U \times x_{\text{gradient}}) + (V \times y_{\text{gradient}})
\]

Calculate brightness of polygon at that point using Phong illumination model.
Landscape Examples

Left – Perspective view of the area, around Pasadena, CA, computer-generated from co-registered 30 m LANDSAT Thematic Mapper imagery and voxel format terrain elevation data. Courtesy of Image Data Corp. (now Core Software Technology ©).

Bump Mapping - Issues

• Silhouette
• Proper scaling
• Aliasing

Texture Mapping

• Bump Mapping

Texture Mapping - Environment mapping

• Create an image, representing the reflection of the world onto an object
• Use surrounding sphere or box, image is texture map indexed by direction of reflection ray
• Poor-man’s ray tracing - cheaper
• Can also be used to do illumination (image-based lighting)

Bump Mapping

• Applet
http://users.interfriends.net/maurid/BumpMapping.htm
Debevec Film: Rendering with Natural Light (http://www.debevec.org/RNL/)

Texture Mapping - Environment mapping

- Not associated with a particular object but with an imaginary surface surrounding the scene
  - Specular Reflection - indexed by reflected ray
  - Diffuse - by surface normal
  - Transparency - refracted ray direction

Texture mapping - Illumination Mapping

Texture mapping - Reflection Mapping

Texture mapping - 3D Textures

Film: 2000 Proceedings, Image-Based: Relief Texture Mapping

Genetic Texture - Parent with 19 Random Variations

[Sims91]

Advantages of texture mapping
- Easy way to add complexity to a scene
- Hardware support

Issues:
- Aliasing - because of holes or overlaps, interpolation and perspective projection
- Limited resolution (zoom in DOOM effect)
- Static image

Aliased image

Anti-aliased image
Texture Mapping – Level of Detail (MIPMAPS)

- Mipmaps
  - Pre-calculate your texture map at many resolutions (or layers)
  - Store all "texture layers" in a single image.
  - Use "appropriate" layer when performing rendering, interpolating between levels as required:
  - MIP is an acronym for the Latin phrase multum in parvo, which can be translated as "much on a small object", some also say it stands for Multiple Image Pattern

Recent Texture Research

- 2000 Proceedings
  - Image-Based: Relief Texture Mapping (uses OpenGL!)
- 2001 Proceedings
  - Illumination & Texture Photorealism: Texture Mapping Progressive Meshes
  - Procedural Modeling: Feature Based Cellular Texturing
  - Interaction of Light and Matter: Polynomial Texture Maps
  - Point-Based: Surface Splatting
- 2003 Proceedings
  - GraphCut Textures
  - Wang Tiles
  - Synthesis of Progressively Variant Textures on Arbitrary Surfaces
  - View Dependent Displacement Mapping

Applying Textures

- Three steps
  - Specify texture
    - Read or generate image
    - Assign to texture
    - Enable texturing
    - Assign texture coordinates to vertices
    - Specify texture parameters
      - Wrapping, filtering, etc.

Specify Texture Image

- Define a texture image from an array of pixels in memory

```c
glTexImage1D( target, level, components, w, border, format, type, *pixels );
glTexImage2D( target, level, components, w, h, border, format, type, *pixels );
```

- Dimensions of image must be powers of 2
- Pixel colors are processed by pixel pipeline
  - Pixel scales, biases and lookups can be done
Specify Texture Image

- Pixel data formats are the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Values are</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_RGB</td>
<td>red intensities</td>
</tr>
<tr>
<td>GL_BLUE</td>
<td>blue intensities</td>
</tr>
<tr>
<td>GL_GREEN</td>
<td>green intensities</td>
</tr>
<tr>
<td>GL_ALPHA</td>
<td>alpha intensities</td>
</tr>
<tr>
<td>GL_LUMINANCE</td>
<td>grayscale colors</td>
</tr>
<tr>
<td>GL_LUMINANCE_ALPHA</td>
<td>alpha and grayscale colors</td>
</tr>
<tr>
<td>GL_COLOR_INDEX</td>
<td>color indices</td>
</tr>
<tr>
<td>GL_LUMINANCE_INDEX</td>
<td>grayscale colors</td>
</tr>
</tbody>
</table>

Converting A Texture Image

- If dimensions of image are not power of 2, must convert

\[
gluScaleImage(\ format, \ w_in, \ h_in, \ type_in, \ *data_in, \ w_out, \ h_out, \ type_out, \ *data_out);\]

- Parameters:
  - \*in are for source image
  - \*out are for destination image

- Image interpolated and filtered during scaling

Specifying a Texture: Other Methods

- Use current frame buffer as source of texture image
  - `glCopyTexImage1D(...)`
  - `glCopyTexImage2D(...)`

- Modify part of a defined texture
  - `glTexSubImage1D(...)`
  - `glTexSubImage2D(...)`
  - `glTexSubImage3D(...)`

- Do both with `glCopyTexSubImage2D(...)` , etc.

Texture Objects

- Problem: OpenGL works with a current texture
- What if we want more than one?
- Solution: texture objects
- Like display lists for texture images
  - One image per texture object
  - May be shared by several graphics contexts
- Create texture objects with texture data and state

- Example - using three textures:

```c
GLint textures[3];
// presumably, pixels0, pixels1, and pixels2 have
// been declared elsewhere
glEnable( GL_TEXTURE_2D );
glGenTextures( 3, textures );
glBindTexture( GL_TEXTURE_2D, textures[0] );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, nCols, nRows, 0,
GL_RGB, GL_UNSIGNED_BYTE, pixels0 );
glBindTexture( GL_TEXTURE_2D, textures[1] );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, nCols, nRows, 0,
GL_RGB, GL_UNSIGNED_BYTE, pixels1 );
glBindTexture( GL_TEXTURE_2D, textures[2] );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, nCols, nRows, 0,
GL_RGB, GL_UNSIGNED_BYTE, pixels2 );
```
Texture Objects

• Once we have created the bindings, we can switch between textures easily:

```c
// use texture #0 now
glBindTexture( GL_TEXTURE_2D, textures[0] );
...
// switch to texture #1
glBindTexture( GL_TEXTURE_2D, textures[1] );
...
// use texture #2 now
glBindTexture( GL_TEXTURE_2D, textures[2] );
...```

Mapping a Texture

• Based on parametric texture coordinates
• `glTexCoord*()` specified at each vertex

![Mapping a Texture Diagram](image)

Applying a Texture

• We apply a texture by interspersing calls to `glTexCoord*()` with calls to `glVertex*()`:

```c
glBegin( GL_QUADS );
glTexCoord2i( tx1, ty1 ); glVertex3i( ix1, iy1, iz1 );
glTexCoord2i( tx2, ty2 ); glVertex3i( ix2, iy2, iz2 );
glTexCoord2i( tx3, ty3 ); glVertex3i( ix3, iy3, iz3 );
...
glEnd();
```

Texture Environments

• A texture environment specifies how texture values are interpreted when a fragment is textured

```c
glTexImage[fi]( target, pname, params );
glTexImage[fi]v( target, pname, *params );
```

• `target` - a texture environment, `GL_TEXTURE_ENV`.
• `pname` - texture environment parameter name
  - `GL_TEXTURE_ENV_MODE` - use a texture function
  - `GL_TEXTURE_ENV_COLOR`
• `*params` - a pointer to an array of parameters:
  - symbolic constant `GL_MODULATE`, `GL_DECAL`, `GL_BLEND`, `GL_REPLACE`
  - RGBA color

![Texture Environments Diagram](image)
Texture Functions

- Controls how texture is applied
  
  \texttt{glTexEnv[fi]}(GL\_TEXTURE\_ENV, prop, param);
  \texttt{glTexEnv[fi]}v(GL\_TEXTURE\_ENV, prop, *param);

- \texttt{GL\_TEXTURE\_ENV\_MODE} modes
  - \texttt{GL\_MODULATE}
  - \texttt{GL\_BLEND}
  - \texttt{GL\_DECAL}
  - \texttt{GL\_REPLACE}

- Set blend color with \texttt{GL\_TEXTURE\_ENV\_COLOR}

Texture Coordinate Generation

- \texttt{glTexGen[dfi]}( coord, pname, params );
  \texttt{glTexGen[dfi]}v( coord, pname, *params );

  - coord - texture coordinate
  - GL\_OBJECT\_LINEAR, GL\_EYE\_LINEAR, or GL\_SPHERE\_MAP
  - GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_PLANE, or
    GL\_EYE\_PLANE

- params - a pointer to an array of texture generation parameters
  - If pname is \texttt{GL\_TEXTURE\_GEN\_MODE}:
    - GL\_OBJECT\_LINEAR, GL\_EYE\_LINEAR, or GL\_SPHERE\_MAP
  - Otherwise, the coefficients for the texture-coordinate generation function specified by pname.

Texture Application Methods

- Filter Modes
  - Minification or magnification
  - Special mipmap minification filters

- Wrap Modes
  - Clamping or repeating

- Texture Functions
  - How to mix primitive’s color with texture’s color
    - Blend, modulate or replace texels
**Filter and Repetition Parameters**

```c
void glTexParameter[fi](target, pname, params);
```

- **target** - the target texture:
  - GL_TEXTURE_1D or GL_TEXTURE_2D.
- **pname** - symbolic name of a texture parameter:
  - GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_TEXTURE_BORDER_COLOR.
- **params** - pointer to an array where the value or values of `pname` are stored.

**Texture Filtering Modes**

- To texture parameter options for a currently bound texture:
  - **GL_TEXTURE_WRAP_S** - Determines what should be done if the horizontal texture coordinates ever go beyond the [0,1] range. GL_CLAMP clamps the texture, and GL_REPEAT tiles it.
  - **GL_TEXTURE_WRAP_T** - Determines what should be done if the vertical texture coordinates ever go beyond the [0,1] range. GL_CLAMP clamps the texture, and GL_REPEAT tiles it.
  - **GL_TEXTURE_MAG_FILTER** - Determines what kind of stretching should be done if a texture-mapped polygon is rendered larger than the texture mapped to it (pixel-wise). GL_NEAREST uses simple "blocky" stretching for textures, whereas GL_LINEAR uses linear-interpolated stretching.
  - **GL_TEXTURE_MIN_FILTER** - Determines what kind of stretching should be done if a texture-mapped polygon is rendered smaller than the texture mapped to it (pixel-wise). GL_NEAREST uses simple "blocky" stretching for textures, whereas GL_LINEAR uses linear-interpolated stretching.

**Wrapping Mode**

- Example:

```c
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
```

**Filtering**

- Using GL_NEAREST for both the GL_TEXTURE_MAG_FILTER and the GL_TEXTURE_MIN_FILTER. First, the original 32x32 texture; then, the texture scaled to 66%; then, the texture scaled to 133%. Notice how the textures appear "blocky", especially in the smaller textures.

- Using GL_LINEAR for both the GL_TEXTURE_MAG_FILTER and the GL_TEXTURE_MIN_FILTER. Notice how the textures appear much smoother thanks to the linear interpolation.

**Mipmapped Textures**

- Mipmapping allows for prefiltered texture maps of decreasing resolutions.
- Lessens interpolation errors for smaller textured objects.
- Declare mipmap level during texture definition:

```c
glTexImage*D(GL_TEXTURE_*D, level, …)
```

- GLU mipmap builder routines:

```c
gluBuild*DMipmaps(…)
```

- OpenGL 1.2 introduces advanced LOD controls.
Perspective Correction Hint

- How texture coordinate and color values are interpolated across a primitive
  - either linearly in screen space (simple calculation)
  - or using depth/perspective values (slower)
- Most textures require perspective-correct interpolation to look correct
- Noticeable for polygons "on edge"

```gl
glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint);
```
- where `hint` is one of
  - GL_DONT_CARE
  - GL_NICEST
  - GL_FASTEST

Texture Residency

- Working set of textures
  - high-performance, usually hardware accelerated
  - textures must be in texture objects
  - a texture in the working set is resident
  - for residency of current texture, check `GL_TEXTURE_RESIDENT` state
- If too many textures, not all are resident
  - can set priority to have some kicked out first
  - establish 0.0 to 1.0 priorities for texture objects

Is There Room for a Texture?

- Query largest dimension of texture image
  - typically largest square texture
  - doesn't consider internal format size

```gl
glGetIntegerv(GL_MAX_TEXTURE_SIZE, &size);
```
- Texture proxy
  - will memory accommodate requested texture size?
  - no image specified; placeholder
  - if texture won't fit, texture state variables set to 0
    - doesn't know about other textures
    - only considers whether this one texture will fit all of memory