BRDF Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Physically-Based Shading Models

- Phong model produces pretty pictures
- **Cons:** empirical (fudged?) \((\cos^\alpha \phi)\), plastic look
- Shaders can implement better lighting/shading models
- Trend towards Physically-based lighting models
- Physically-based?
  - Based on physics of light, interactions with actual surface
  - Use Optics/Physics theories
- Classic: Cook-Torrance shading model (TOGS 1982)
Cook-Torrance Shading Model

- Ambient and diffuse terms same as Phong
- New, better specular component than \( \cos^\alpha \phi \),

\[
\cos^\alpha \phi \rightarrow \frac{F(\phi, \eta)DG}{(n \cdot v)}
\]

- **Idea:** surfaces has small V-shaped microfacets (grooves)

- Many grooves at each surface point
- **Distribution term D:** Grooves facing a direction contribute
- E.g. half of grooves face 30 degrees, etc
- **F term:** what fraction of light bounces off, depends on material, angle
Self-Shadowing (G Term)

- Grooves on very rough surface may block other grooves (shadowing & masking)

\[ \cos^\alpha \phi \rightarrow \frac{F(\phi, \eta)DG}{(n \cdot v)} \]

**Shadowing**
(light blocked on way into groove)

**Masking**
(light blocked on way out of groove)
BV BRDF (Surface Material) Viewer

Tool to visualize distribution of light bounce
BRDF (Surface Material) Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Appearance Example: Weathering

- Analytic model for weathering of stone, metals

Weathered Stone

Metallic Patina
(Weathering Effect)
BRDF (Surface Material) Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Measuring BRDFs (Surface Material)

Murray-Coleman and Smith Gonioreflectometer. (Copied and Modified from [Ward92]).
Measured BRDF (Surface Material) Samples

- Mitsubishi Electric Research Lab (MERL)
  http://www.merl.com/brdf/
- Wojciech Matusik
- MIT PhD Thesis
- 100 Samples
BRDF (Surface Material) Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Time-varying BRDF (Surface Material)

- **BRDF**: How different materials reflect light
- **Time varying?**: how reflectance changes over time

![Banana images](a) ![Banana images](b) ![Banana images](c) ![Banana images](d)

![Leaf images](a) ![Leaf images](b) ![Leaf images](c) ![Leaf images](d)
References

Computer Graphics (4731)  
Lecture 17: Texturing

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second
- Many phenomena even more detailed
  - Clouds
  - Grass
  - Terrain
  - Skin
- **Images:** Computationally inexpensive way to add details

Image complexity does not affect the complexity of geometry processing (transformation, clipping...)
Textures in Games

- Mostly made of textures except foreground characters that require interaction
- Even details on foreground texture (e.g. clothes) is texture
Types of Texturing

1. geometric model
2. texture mapped
   Paste image (marble) onto polygon
Types of Texturing

3. Bump mapping
   Simulate surface roughness (dimples)

4. Environment mapping
   Picture of sky/environment over object
Texture Mapping

1. Define texture position on geometry

2. Projection

3. Texture lookup

4. Patch texel

3D geometry

2D projection of 3D geometry

2D image
Texture Representation

- **Bitmap (pixel map) textures:** images (jpg, bmp, etc) loaded
- **Procedural textures:** E.g. fractal picture generated in OpenGL program
- Textures applied in shaders

**Bitmap texture:**
- 2D image - 2D array `texture[height][width]`
- Each element (or **texel**) has coordinate `(s, t)`
- `s` and `t` normalized to `[0,1]` range
- Any `(s,t)` => [red, green, blue] color
Texture Mapping

- Map? Each (x, y, z) point on object, has corresponding (s, t) point in texture
  
  \[ s = s(x, y, z) \]
  
  \[ t = t(x, y, z) \]
6 Main Steps to Apply Texture

1. Create texture object
2. Specify the texture
   - Read or generate image
   - assign to texture (hardware) unit
   - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
   - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders
Step 1: Create Texture Object

- OpenGL has **texture objects** (multiple objects possible)
  - 1 object stores 1 texture image + texture parameters
- First set up texture object

```c
GLuint mytex[1];
glGenTextures(1, mytex); // Get texture identifier
glBindTexture(GL_TEXTURE_2D, mytex[0]); // Form new texture object
```

- Subsequent texture functions use this object
- Another call to `glBindTexture` with new name starts new texture object
Step 2: Specifying a Texture Image

- Define picture to paste onto geometry
- Define texture image as array of *texels* in CPU memory
  
  ```
  Glubyte my_texels[512][512][3];
  ```

- Read in scanned images (jpeg, png, bmp, etc files)
  - If uncompressed (e.g. bitmap): read from disk
  - If compressed (e.g. jpeg), use third party libraries (e.g. Qt, devil) to decompress + load

bmp, jpeg, png, etc
Step 2: Specifying a Texture Image

- Procedural texture: generate pattern in application code

- Enable texture mapping
  - `glEnable(GL_TEXTURE_2D)`
  - OpenGL supports 1-4 dimensional texture maps
Specify Image as a Texture

Tell OpenGL: this image is a texture!!

```c
glTexImage2D( target, level, components,
    w, h, border, format, type, texels );
```

- `target`: type of texture, e.g. `GL_TEXTURE_2D`
- `level`: used for mipmapping (0: highest resolution. More later)
- `components`: elements per texel
- `w, h`: width and height of `texels` in pixels
- `border`: used for smoothing (discussed later)
- `format, type`: describe texels
- `texels`: pointer to texel array

Example:

```c
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB,
    GL_UNSIGNED_BYTE, my_texels);
```
Fix texture size

- OpenGL textures must be power of 2
- If texture dimensions not power of 2, either
  1) Pad zeros
  2) Scale the Image
6 Main Steps. Where are we?

1. Create texture object
2. Specify the texture
   - Read or generate image
   - assign to texture (hardware) unit
   - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
   - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders
Step 3: Assign Object Corners to Texture Corners

- Each object corner \((x,y,z)\) => image corner \((s, t)\)
  - E.g. object \((200,348,100)\) => \((1,1)\) in image
- Programmer establishes this mapping
Step 3: Assigning Texture Coordinates

- After specifying corners, interior (s,t) ranges also mapped
- Example? Corners mapped below, abc subrange also mapped
Step 3: Code for Assigning Texture Coordinates

- **Example:** Map a picture to a quad
- For each quad corner (vertex), specify
  - Vertex \((x,y,z)\),
  - Corresponding corner of texture \((s,t)\)
- May generate array of vertices + array of texture coordinates

```
points[i] = point3(2,4,6);
tex_coord[i] = point2(0.0, 1.0);
```
Step 3: Code for Assigning Texture Coordinates

```c
void quad( int a, int b, int c, int d )
{
    quad_colors[Index] = colors[a];         // specify vertex color
    points[Index] = vertices[a];             // specify vertex position
    tex_coords[Index] = vec2( 0.0, 0.0 );    // specify corresponding texture corner
    index++;
    quad_colors[Index] = colors[b];
    points[Index] = vertices[b];
    tex_coords[Index] = vec2( 0.0, 1.0 );
    Index++;

    // other vertices
}
```

- **colors array**
  - Colors 1: `r g b`
  - Colors 2: `r g b`
  - Colors 3: `r g b`

- **points array**
  - Position 1: `a`
  - Position 2: `b`
  - Position 3: `c`

- **tex_coord array**
  - Tex0: `s`
  - Tex1: `s`
  - Tex2: `s`
Step 5: Passing Texture to Shader

- Pass vertex, texture coordinate data as vertex array
- Set texture unit

offset = 0;
GLuint vPosition = glGetUniformLocation(program, "vPosition");
glEnableVertexAttribArray(vPosition);
glVertexAttribPointer(vPosition, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(offset));

offset += sizeof(points);
GLuint vTexCoord = glGetUniformLocation(program, "vTexCoord");
glEnableVertexAttribArray(vTexCoord);
glVertexAttribPointer(vTexCoord, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(offset));

// Set the value of the fragment shader texture sampler variable
//   ("texture") to the appropriate texture unit.

glUniform1i(glGetUniformLocation(program, "texture"), 0);
Step 6: Apply Texture in Shader (Vertex Shader)

- Vertex shader receives data, output texture coordinates to fragment shader

```glsl
in vec4 vPosition; //vertex position in object coordinates
in vec4 vColor;  //vertex color from application
in vec2 vTexCoord; //texture coordinate from application

out vec4 color; //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated

texCoord = vTexCoord
color = vColor
gl_Position = modelview * projection * vPosition
```
Step 6: Apply Texture in Shader (Fragment Shader)

- Textures applied in fragment shader
- Samplers return a texture color from a texture object

```cpp
in vec4 color;  //color from rasterizer
in vec2 texCoord;  //texture coordinate from rasterizer
uniform sampler2D texture;  //texture object from application

void main() {
    gl_FragColor = color * texture2D( texture, texCoord );
}
```

Output color of fragment
Original color of object
Lookup color of texCoord (s,t) in texture
Map textures to surfaces

- Texture mapping is performed in rasterization

  - For each pixel, its texture coordinates \((s, t)\) interpolated based on corners’ texture coordinates (why not just interpolate the color?)
  - The interpolated texture \((s,t)\) coordinates are then used to perform texture lookup
Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join during fragment processing
  - Object geometry: geometry pipeline
  - Image: pixel pipeline
  - “complex” textures do not affect geometric complexity

![Diagram showing the OpenGL pipeline](image-url)
6 Main Steps to Apply Texture

1. Create texture object
2. Specify the texture
   - Read or generate image
   - assign to texture (hardware) unit
   - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
   - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders

still haven’t talked about setting texture parameters
Step 4: Specify Texture Parameters

- Texture parameters control how texture is applied
  - Wrapping parameters used if \( s, t \) outside \((0,1)\) range
    - Clamping: if \( s, t > 1 \) use 1, if \( s, t < 0 \) use 0
    - Wrapping: use \( s, t \) modulo 1

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
```
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
```
Step 4: Specify Texture Parameters

Mipmapped Textures

- **Mipmapping** pre-generates prefiltered (averaged) texture maps of decreasing resolutions
- Declare mipmap level during texture definition
  
  ```c
  glTexImage2D( GL_TEXTURE_*D, level, ...
  ```
References

- UIUC CS 319, Advanced Computer Graphics Course
- David Luebke, CS 446, U. of Virginia, slides
- Chapter 1-6 of RT Rendering
- Hanspeter Pfister, CS 175 Introduction to Computer Graphics, Harvard Extension School, Fall 2010 slides
- Christian Miller, CS 354, Computer Graphics, U. of Texas, Austin slides, Fall 2011
- Ulf Assarsson, TDA361/DIT220 - Computer graphics 2011, Chalmers Institute of Tech, Sweden