What you will learn...

• Why textures?
• How to generate textures
• How to store textures
• How to apply textures
• GPU-based texture mapping
Why textures?

• Textures represent variations in reflected spectral light over a surface
• Textures are color patterns that represent:
  ▪ complex surface geometry
  ▪ complex surface composition
• Textures add realism to simple geometry
Example 1
Example 2
Texture Basics

Texture Space

(0,0)  (1,0)  (0,1)  (1,1)

Object Space

Screen Space

v

n

x_s

y_s

u
Texture Generation

Polygon to be textured

UV coordinates

Texture
For a triangle
Texture Mapping
Simple 1D Interpolation

• Coloring a triangle

\[ u = 0.5 \]
\[ u = 0.75 \]
\[ u = 0.5 \]
\[ u = 0.25 \]
Texturing Surfaces

Texture Space \((u,v)\) \rightarrow Object Space \((s,t)\) \rightarrow Screen Space \((x,y)\)

\( (u(s,t), v(s,t)) \)
Forward Texture Map

Texture Space

Object Space

Screen Space

(u,v) → (s,t) → (x,y)

((u(s,t),v(s,t)))
Backward Texture Map

Texture Space \( (u,v) \)  
Object Space \( (s,t) \)  
Screen Space \( (x,y) \)
Texture Mapping Triangles
Backward Texture Map

• Each vertex specified by: \((x, y, z, u, v)\)
Backward Texture Map

• At each vertex specify
  ▪ Position Coordinates \((x,y,z)\)
  ▪ Texture Coordinates \((u,v)\)

• Look up the color/texturing at scan conversion rasterization by interpolating the raster position into texture image space
3D Texture Map

• The texture map is a volume
  ▪ a stack of bitmaps - memory intensive

• Per vertex \((s, t, r)\) coordinates

• Applications:
  ▪ medical 3D images (plane through scan)
  ▪ solid materials (wood)
Procedural Texture Map

• Instead of using a 1-, 2-, or 3-D image, define a function that returns a color dependent on value of $s$, $(s,t)$ or $(s,t,r)$

• Simple example: a chessboard

• Advanced: wood, marble, etc.
Example: Perlin Texture

Images by Ken Buckner
Advanced Mapping
Intermediate surfaces

- Plane
- Cylinder
- Sphere
- Cube

Planar projector
Planar Orthographic

Orthographic projection onto XY plane:
\[ u = x, \ v = y \]

...onto YZ plane

...onto XZ plane
Cylindrical

- Convert rectangular coordinates \((x, y, z)\) to cylindrical \((r, \phi, h)\),
- Use only \((h, \phi)\) to index texture image
Spherical

- Convert rectangular coordinates \((x, y, z)\) to spherical \((\theta, \phi)\)
Parametric Surfaces

- Surface given by: $x = f(u, v), y = g(u, v), z = h(u, v)$
Summary:
Sampling
Sampling Texture

• A pixel maps to a non-rectangular region

• Map on center of pixels
Problems

• Over-sampling the image:
  ▪ same texel maps to several pixels

• Under-sampling the image:
  ▪ pixels only sparsely sample the texels
Solutions

• Nearest neighbor sampling
• Bilinear sampling
• MipMapping
Nearest Neighbor

• Pick the pixel with the nearest center

![Diagram showing Nearest pixel center and Texel coordinates (u,v)]
Bilinear Interpolation

- Linearly interpolate across u and v

Nearest pixel center

Texel coordinates (u,v)

Add an average of the surrounding pixels
Bilinear Interpolation

- Linear interpolate in both directions

\[ T(a, b) = T[i + \Delta_x, j + \Delta_y] \]
\[ = (1 - \Delta_x)(1 - \Delta_y)T[i, j] + \Delta_x(1 - \Delta_y)T[i + 1, j] \]
\[ + (1 - \Delta_x)\Delta_y T[i, j + 1] + \Delta_x\Delta_y T[i + 1, j + 1] \]
MipMaps

• Texture images must be a power of 2:
  ▪ 64, 128, 256, 512, 1024
• Resample the image at lower resolutions
• Create a pyramid of texture images
• Interpolate the texture between two adjacent layers
MipMaps

![MipMap Pyramid Diagram](image)

- 256 x 256
- 128 x 128
- 64 x 64
- 32 x 32
- 1 x 1
Bump Mapping

• Perturb the Normal Vector

\[ \tilde{Q}(u) = Q(u) + d(u)N(u) \]

\[ N(u) = \text{normal}[Q(u)] \]
Bump Mapping

Texture 1

Texture 2

Bump Map

Rendered Image
Environment Mapping
WebGL Textures
// Create a GL texture object “cubeTexture”
// Load the texture from the image file
function initTextures() {
    cubeTexture = gl.createTexture();
    cubeImage = new Image();
    cubeImage.onload = function() {
        handleTextureLoaded(cubeImage, cubeTexture);
    }
    cubeImage.src = "textureImage.png";
}
Bind Texture Buffers

// Callback function that is called after the texture image is loaded.
// Specifies that the new texture is the current texture by binding it to gl.TEXTURE_2D.
// Loaded image is passed into texImage2D() to write the image data into the texture.
function handleTextureLoaded(image, texture) {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
Power of 2 image sizes

1 x 1
32 x 32
64 x 64
128 x 128
256 x 256
NPOT: Not Power-of-Two

• Texture images with side lengths \((x,y)\)
  - power of two are ideal: e.g. 1, 2, 4, 8, 16, 32,

• Not all devices can support >4096.

• But, if the images are from outside, they may not be \(2^n\).

• How to solve this problem?
NPOT Solutions

• Best solution:
  ▪ Reduce the image into a square image with side lengths as a power of 2, i.e. $2^n$

• Second best solution:
  ▪ Could make the texture images the same as the native resolution of the monitor
NPOT Problems

• Cannot use MIPMAPPING
• Cannot REPEAT, tile or wrap
• MipMapping and UV repeat can be disabled with \texttt{gl.texParameteri()}
NPOT disabling MipMaps

// gl.NEAREST also allowed, instead of gl.LINEAR, as neither mipmap.
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

// Prevents s-coordinate wrapping (repeating).
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);

// Prevents t-coordinate wrapping (repeating).
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
Mapping Tex onto Faces

cubeVerticesTextureCoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVerticesTextureCoordBuffer);

var textureCoordinates = [
  // Front Face
  0.0, 0.0,
  1.0, 0.0,
  1.0, 1.0,
  0.0, 1.0, ...
];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoordinates), gl.STATIC_DRAW);
Updating the VS

• First, we need the location of the attribute “aTextureCoord” in the initShaders() function:

```javascript
textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");	gl.enableVertexAttribArray(textureCoordAttribute);
```
Update the VS

```xml
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;
  attribute vec2 aTextureCoord;
  uniform mat4 uMVMMatrix;
  uniform mat4 uPMatrix;
  varying highp vec2 vTextureCoord; // communicates with the FS
  void main(void) {
    gl_Position = uPMatrix * uMVMMatrix * vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
  }
</script>
```
Update the FS

```html
<script id="shader-fs" type="x-shader/x-fragment">

varying highp vec2 vTextureCoord;  // passed from VS
uniform sampler2D uSampler;  // holds the texture image

void main(void) {

  gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));

}

</script>
```
FS Note

• Instead of assigning a color value to the fragment color, the fragment color is computed by fetching the texel (that is, the pixel within the texture) that the sampler says best maps to the fragment position.
Finally, drawing

gl.activeTexture(gl.TEXTURE0);

```javascript
gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);
```

// note: GL provides 32 texture registers;
// the first of these is gl.TEXTURE0.