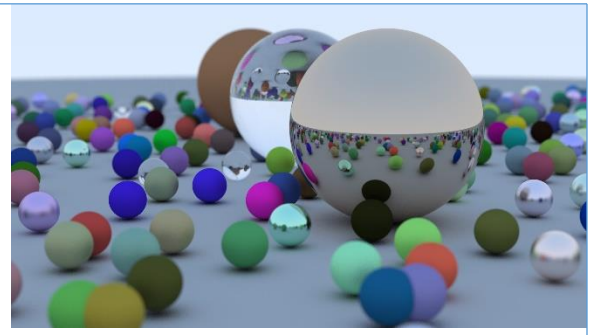
Comp4422



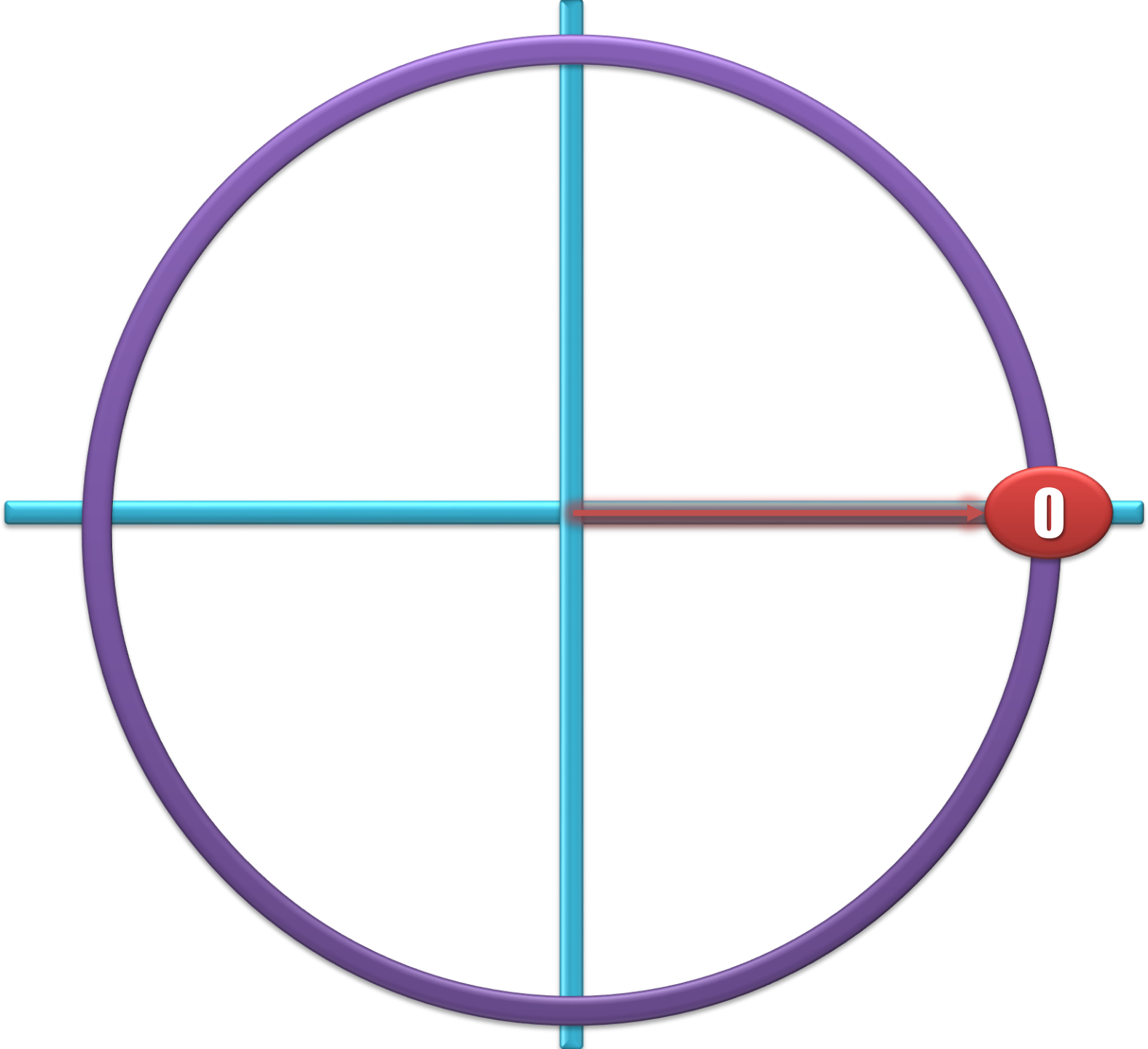
Computer Graphics

Lecture 06:

Texture Mapping

What you will learn…

- Why textures in computer graphics
- How to generate textures
- How to store textures
- How to apply textures
- GPU-based texture mapping

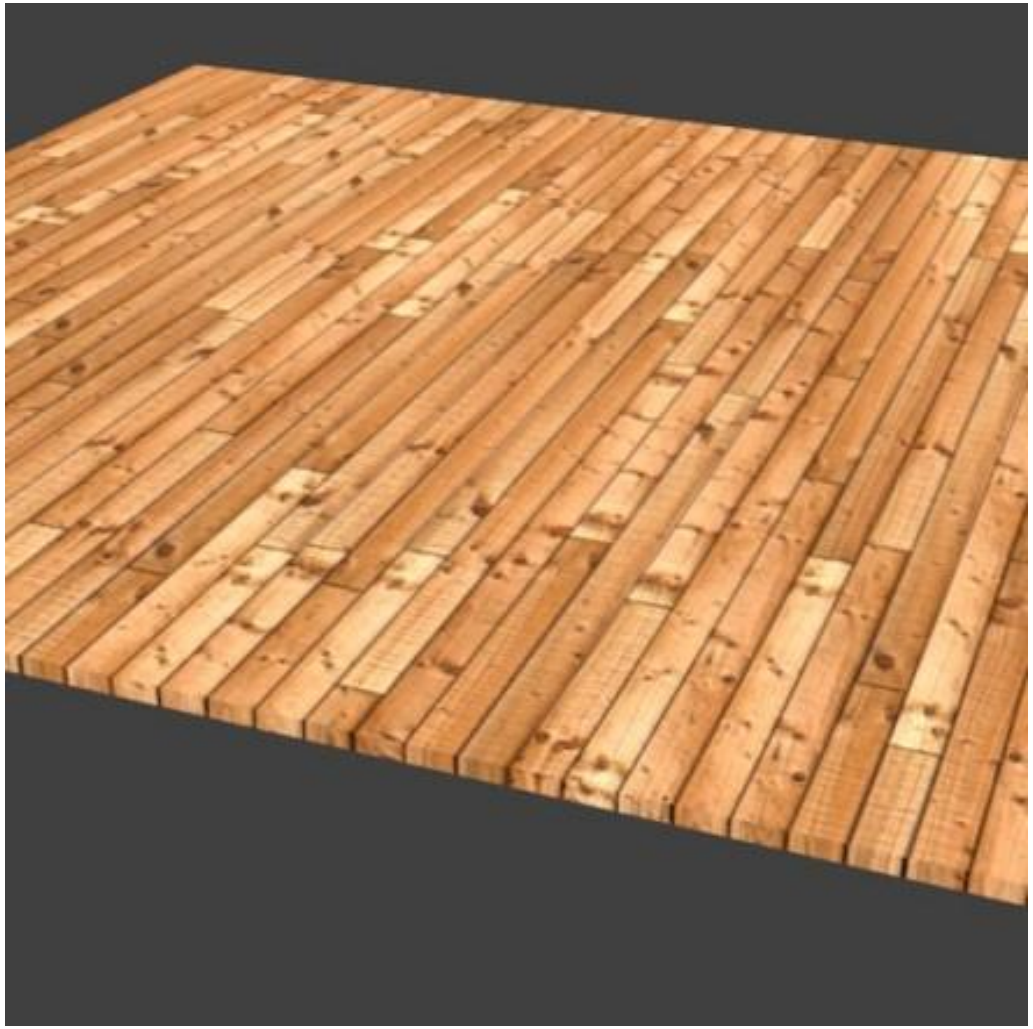

0

Why textures?

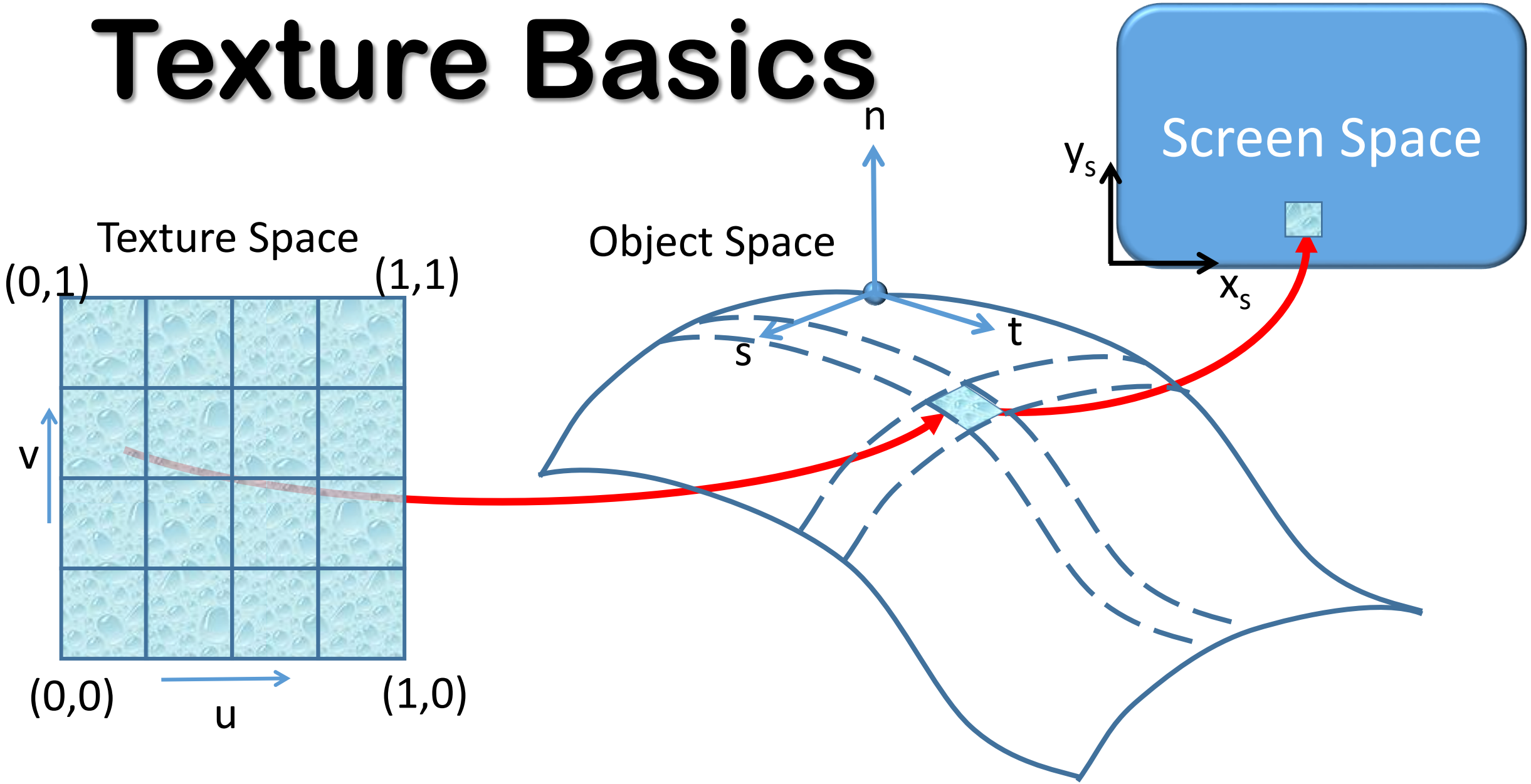
- Textures represent variations in reflected spectral light over a surface
- Textures are color patterns that represent
  - complex surface geometry
  - complex surface composition
- Textures add realism to simple geometry

Example 1

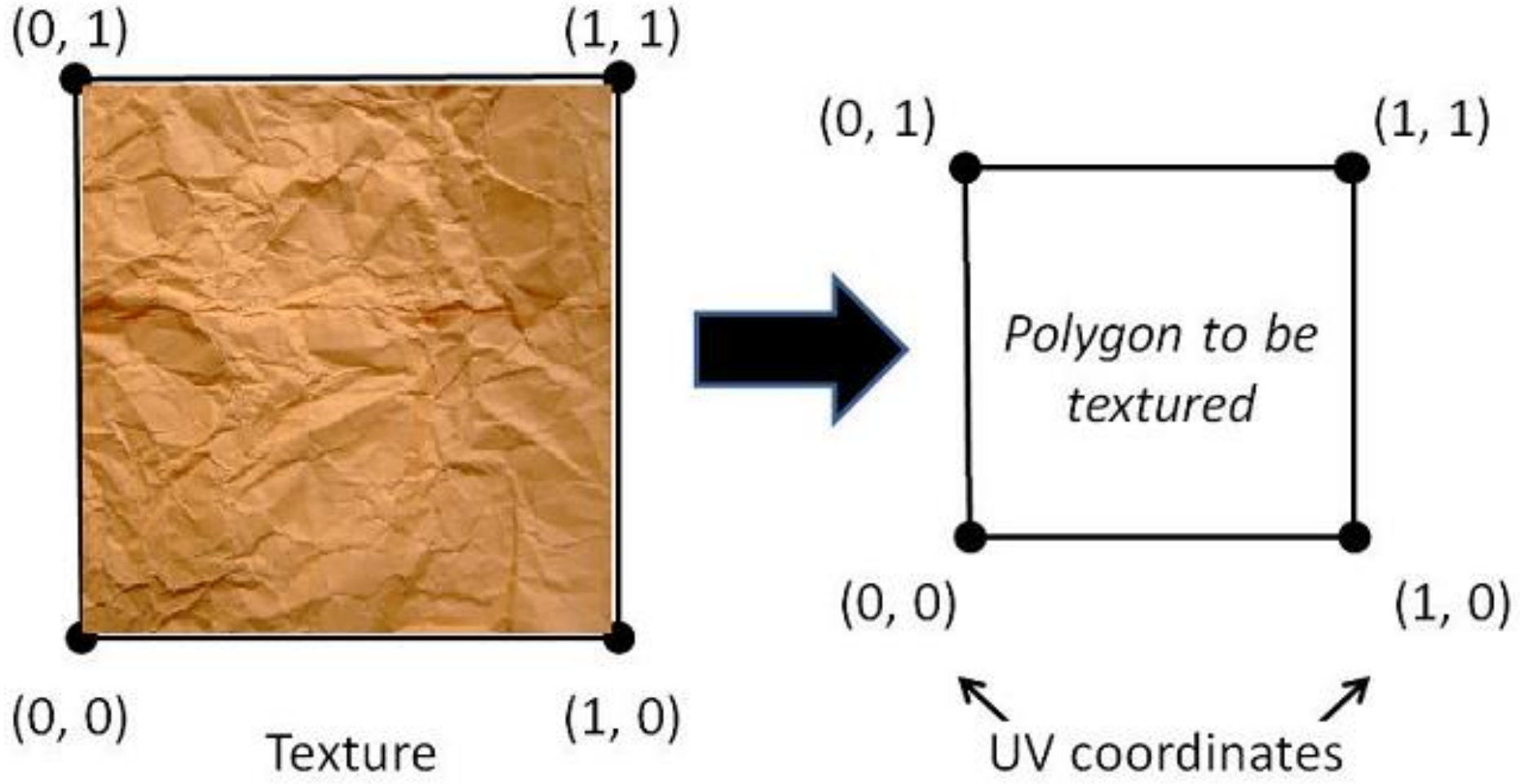
Example 2



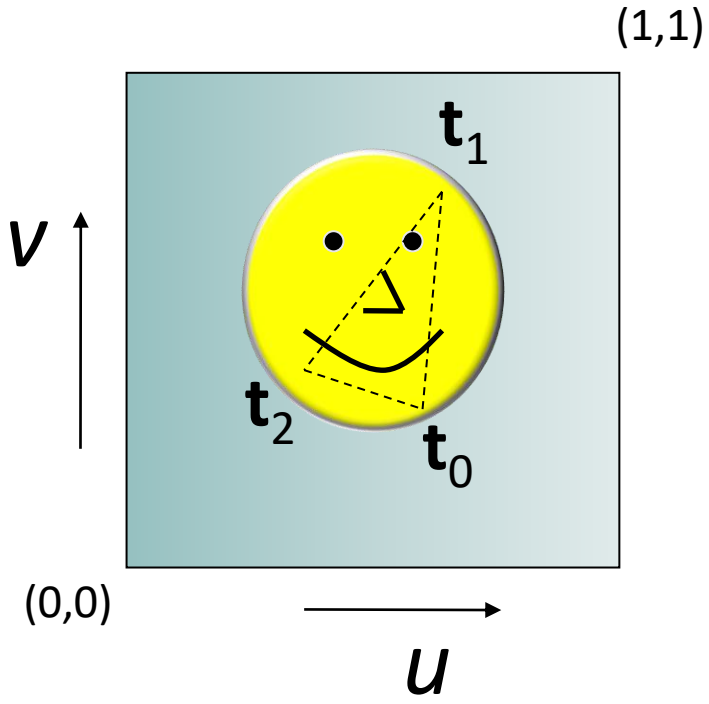
Texture Basics

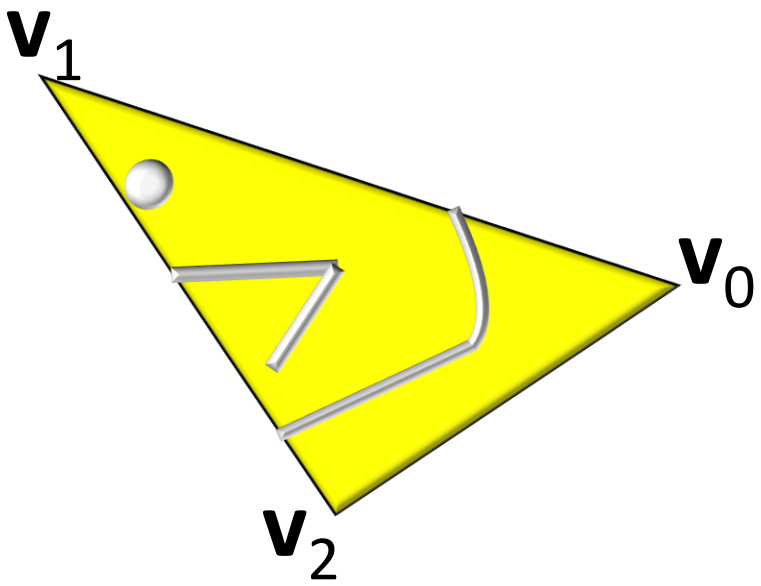


Texture Generation



For a triangle

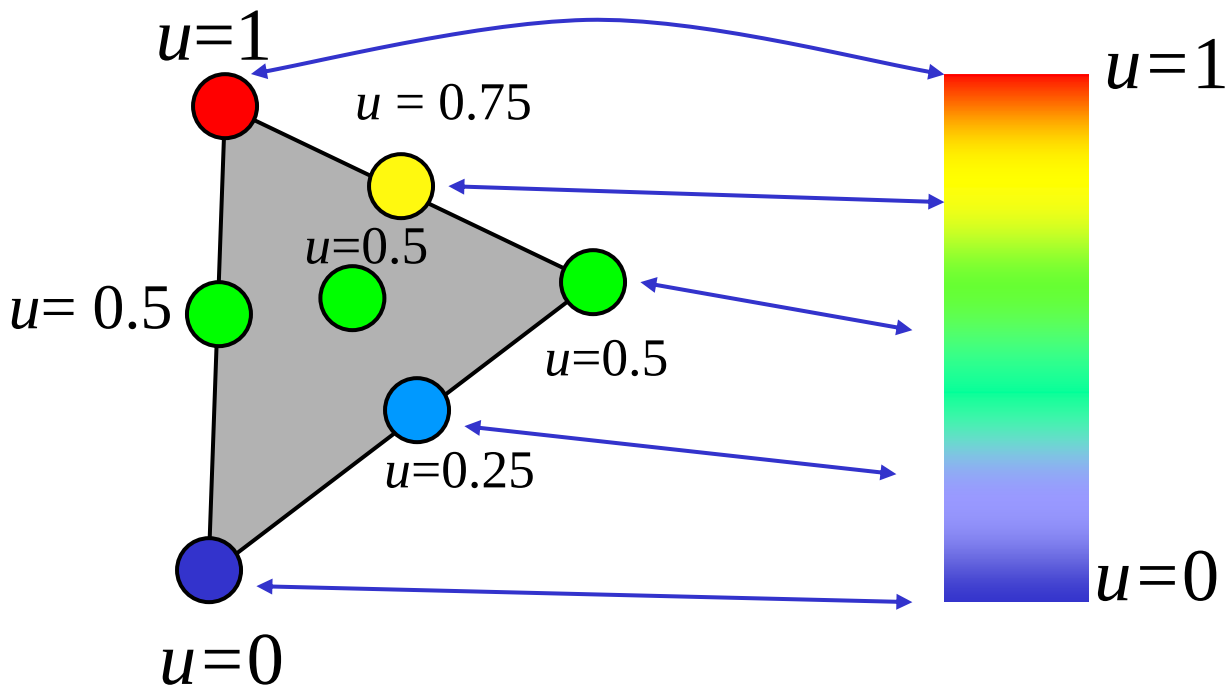




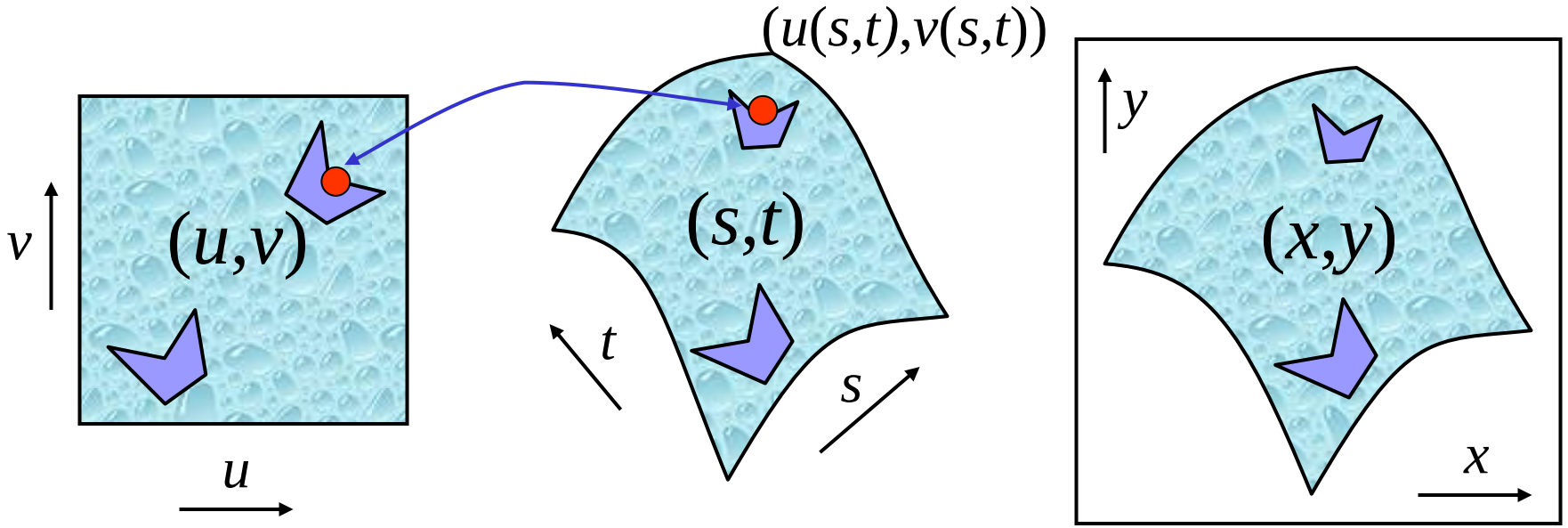
Texture Mapping

Simple 1D Interpolation

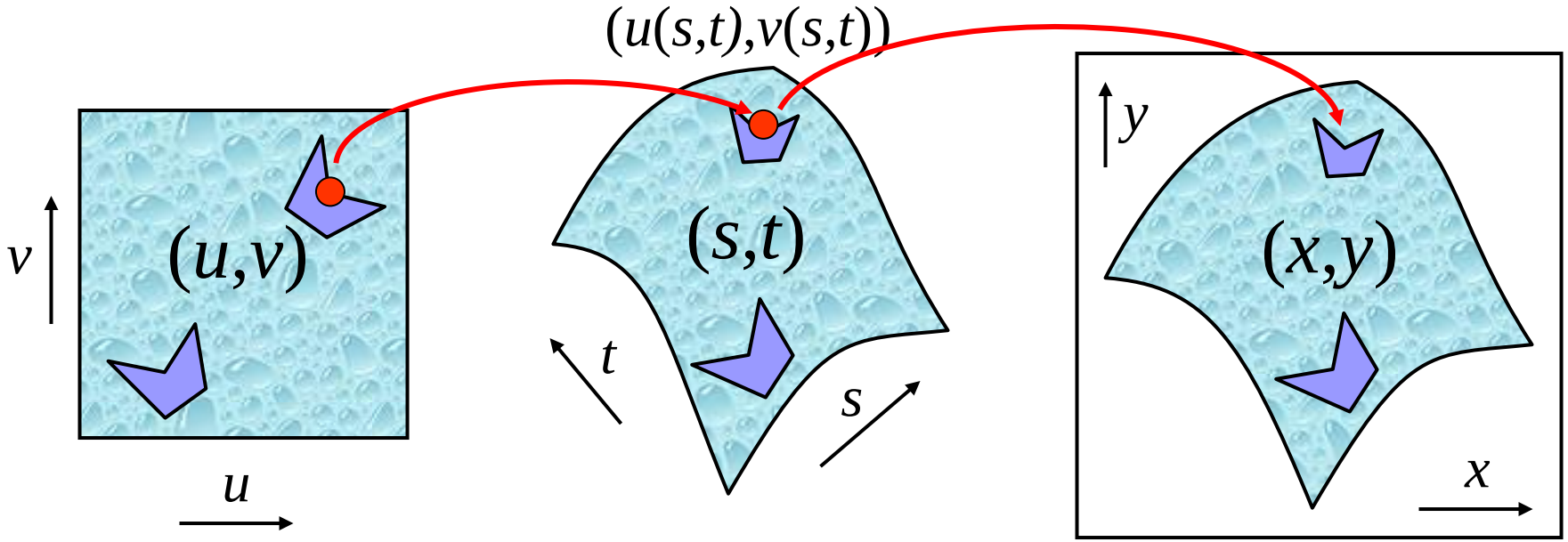
- Coloring a triangle



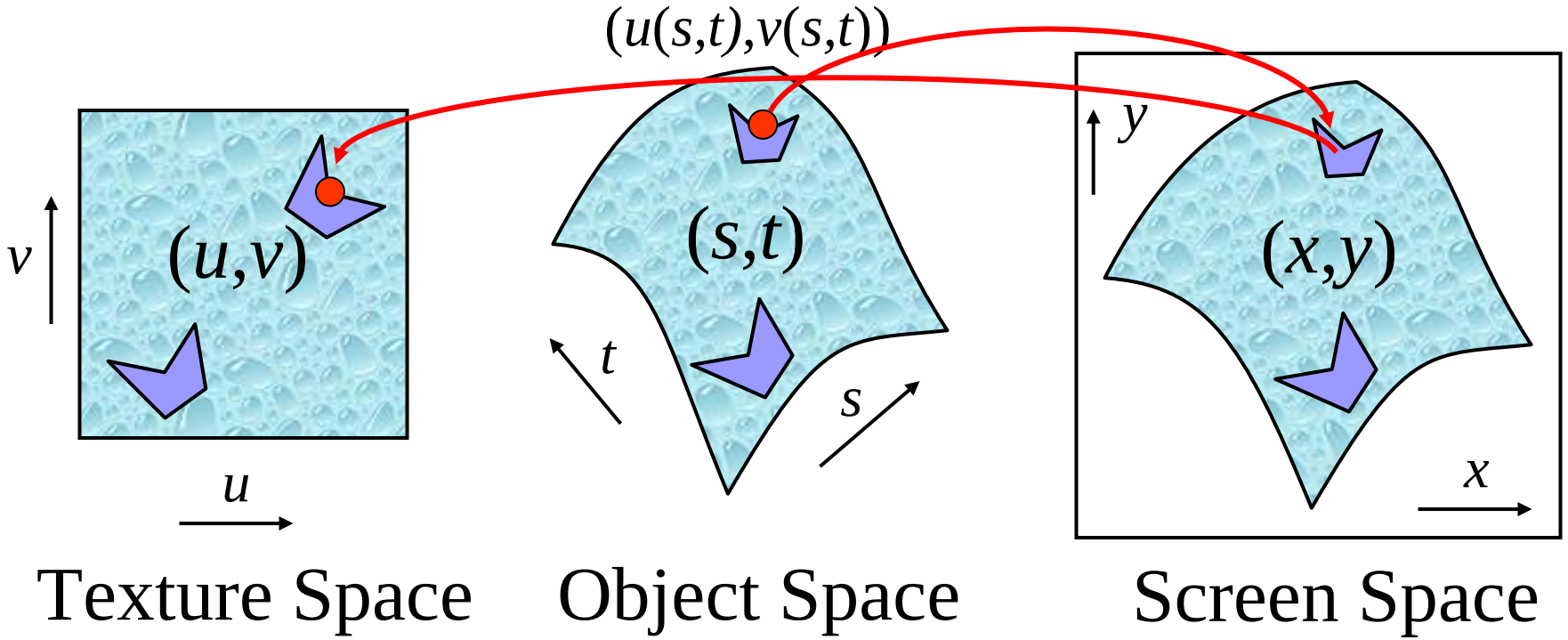
Texturing Surfaces



Forward Texture Map



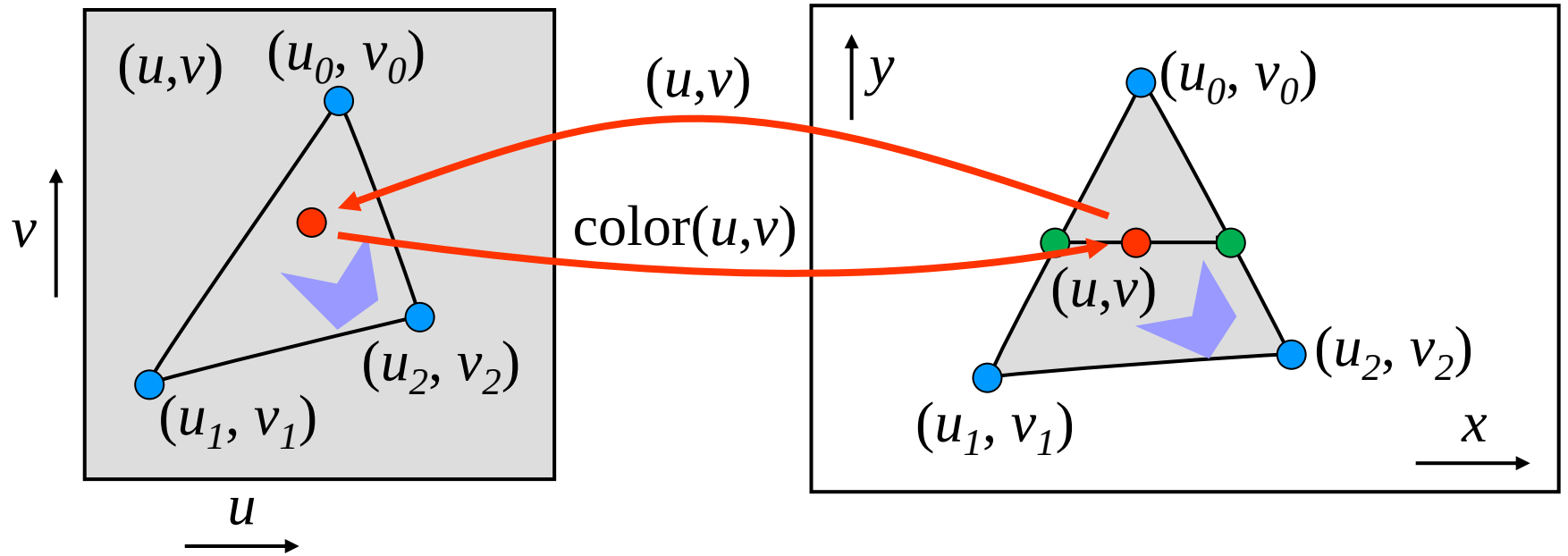
Backward Texture Map



Texture Mapping Triangles

Backward Texture Map

- Each vertex: (x,y,z,u,v)



Backward Texture Map

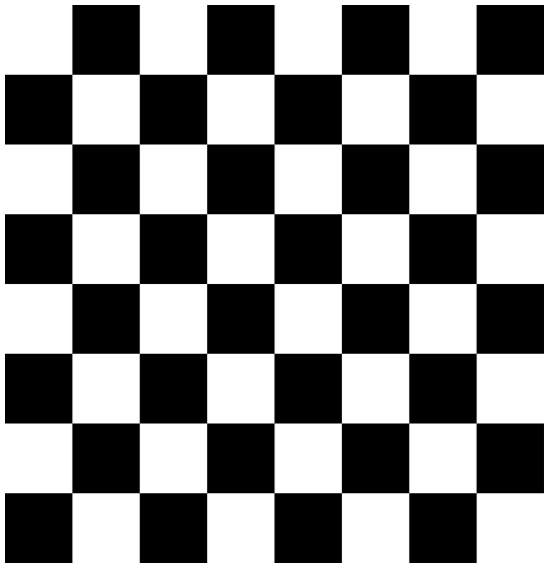
- At each vertex specify
  - Position Coordinates (x,y,z)
  - Texture Coordinates (u,v)
- Look up the color/texture at scan conversion/rasterization by interpolating the raster position into texture image space

3D Texture Map

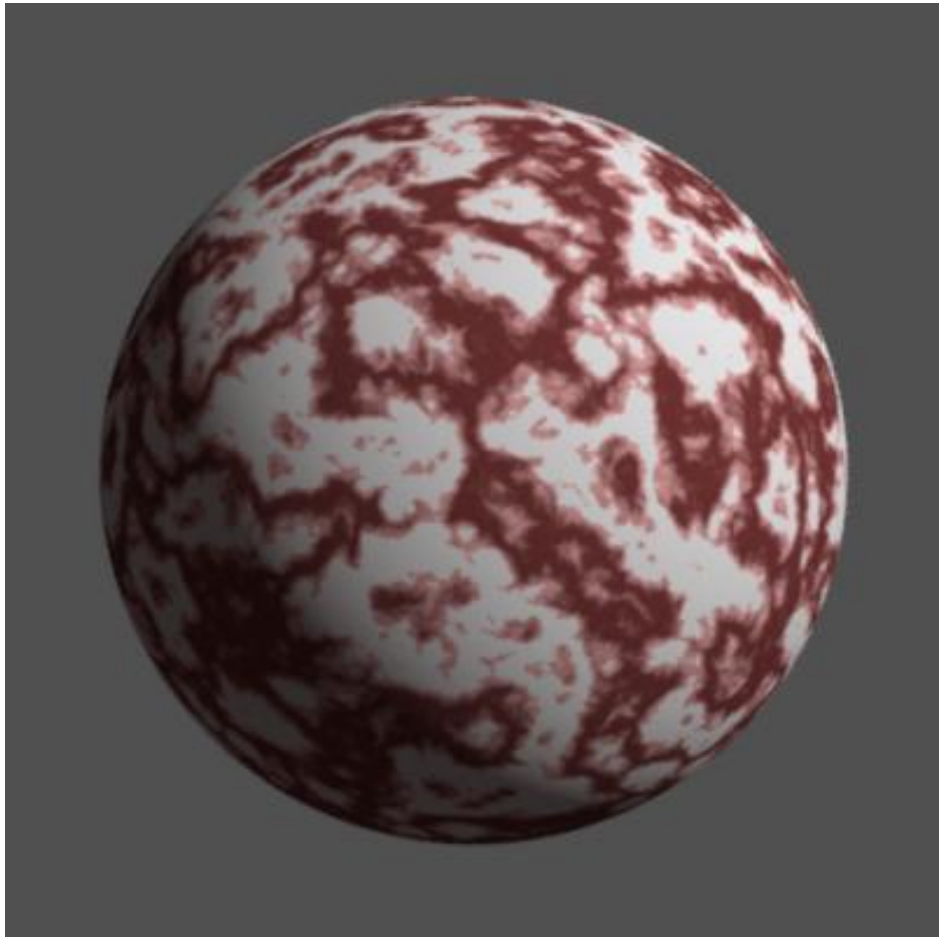
- The texture map is a volume
  - a stack of bitmaps - memory intensive
- Per vertex (*s*,*t*,*r*) coordinates
- Applications:
  - medical 3D images (plane through scan)
  - solid materials (wood)

Procedural Texture Map

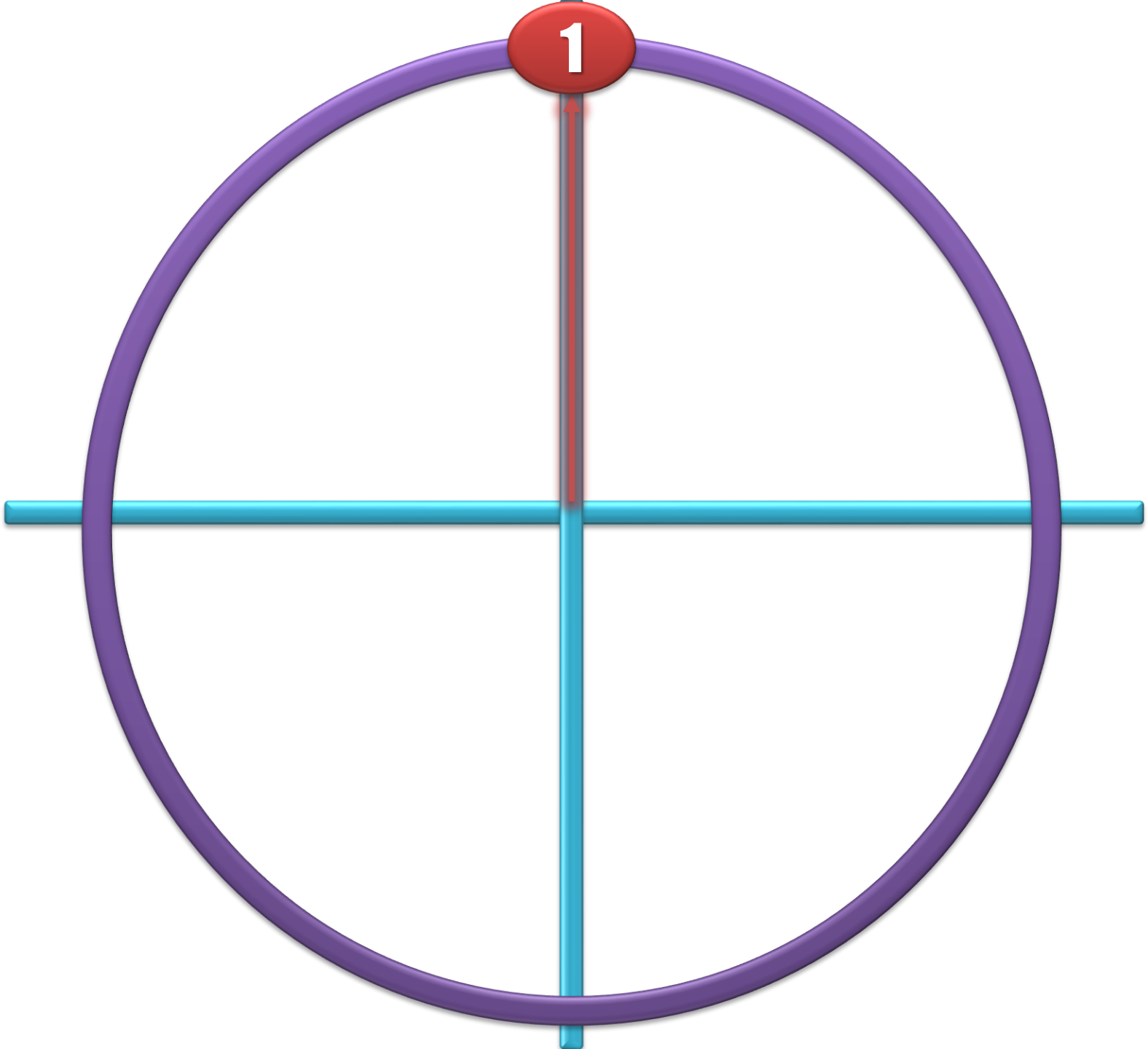
- Instead of using a 1-, 2-, or 3-D image:
  - Define function that returns color dependent on value of s, (s,t) or (s,t,r)
- Simple example: a chessboard
- Advanced: wood, marble, etc.



Example: Perlin Texture



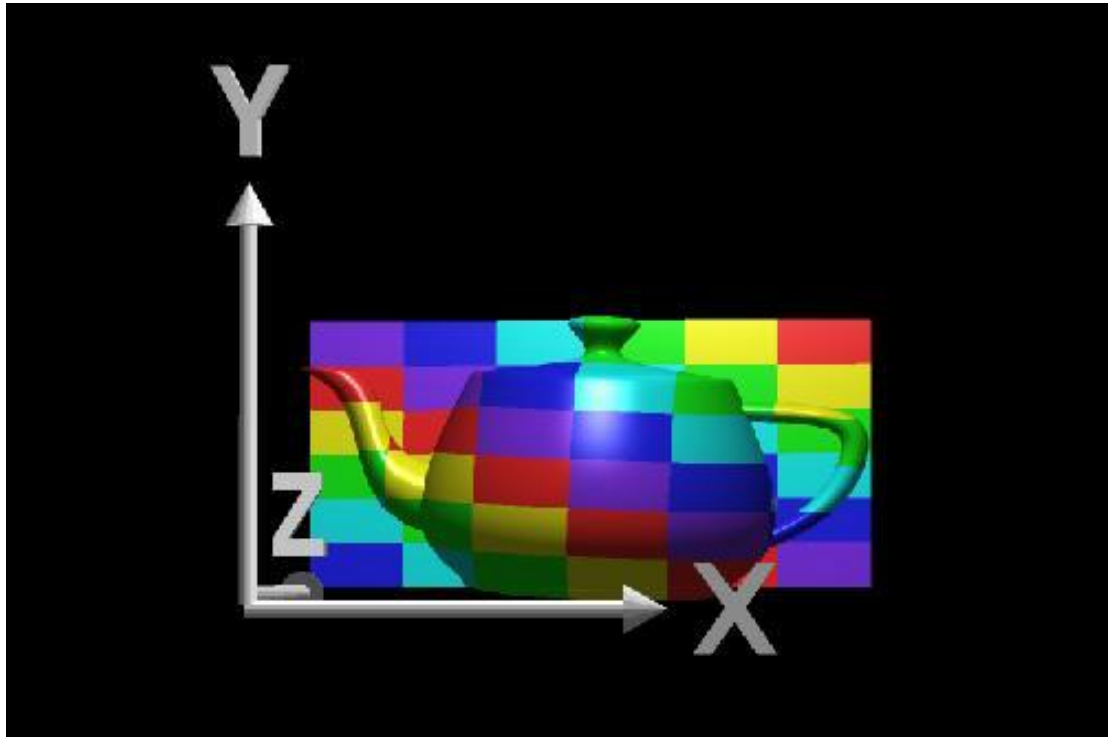
Images by Ken Buckner


1

Advanced Mapping

Intermediate surfaces

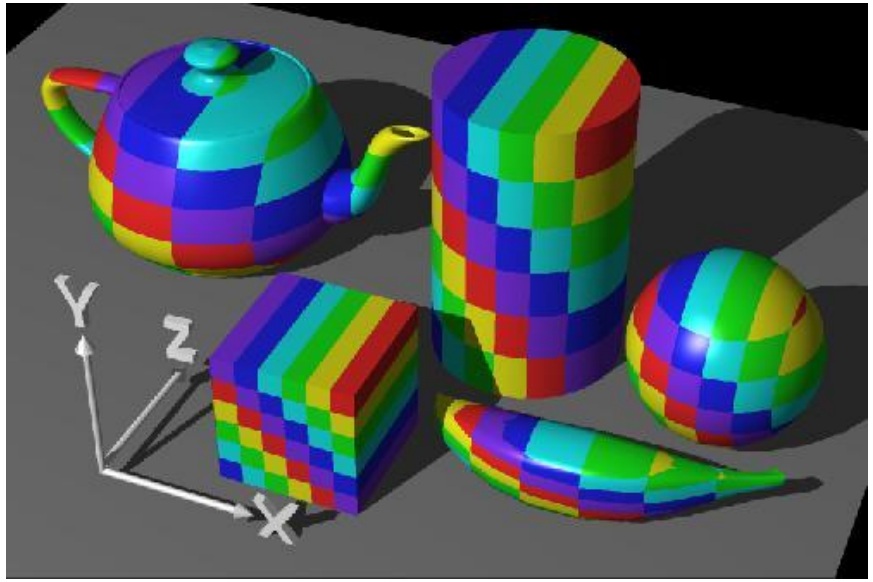
- Plane
- Cylinder
- Sphere
- Cube



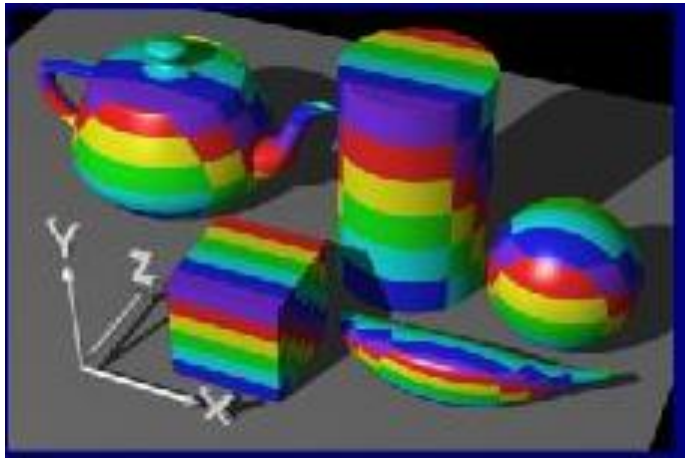
Planar projector

Planar Orthographic

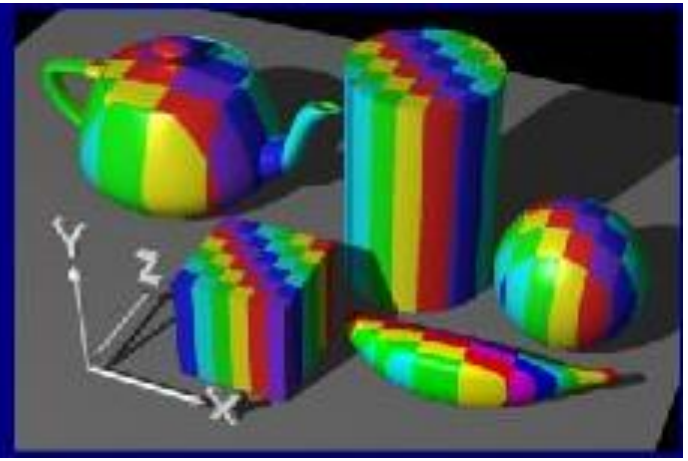
Orthographic projection
onto XY plane:
$u = x,\ v = y$



...onto YZ plane



...onto XZ plane

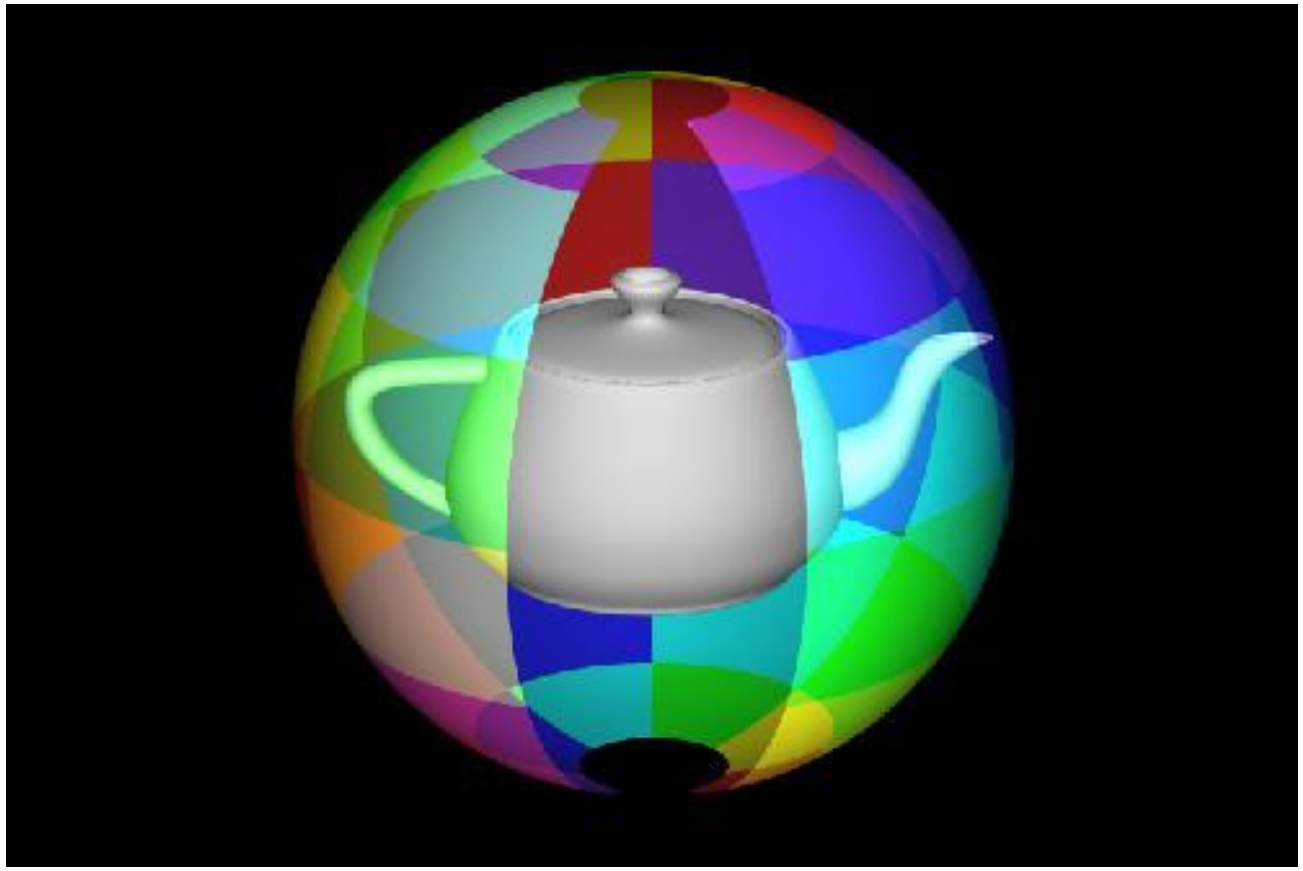


Cylindrical

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

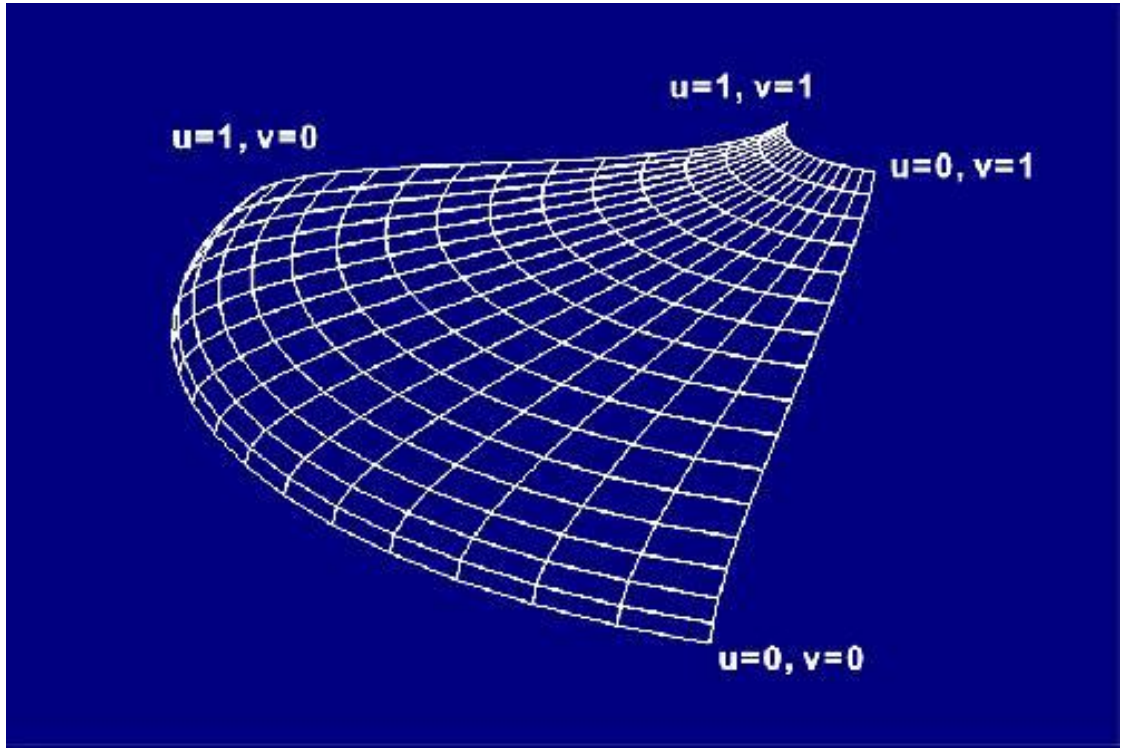
Spherical

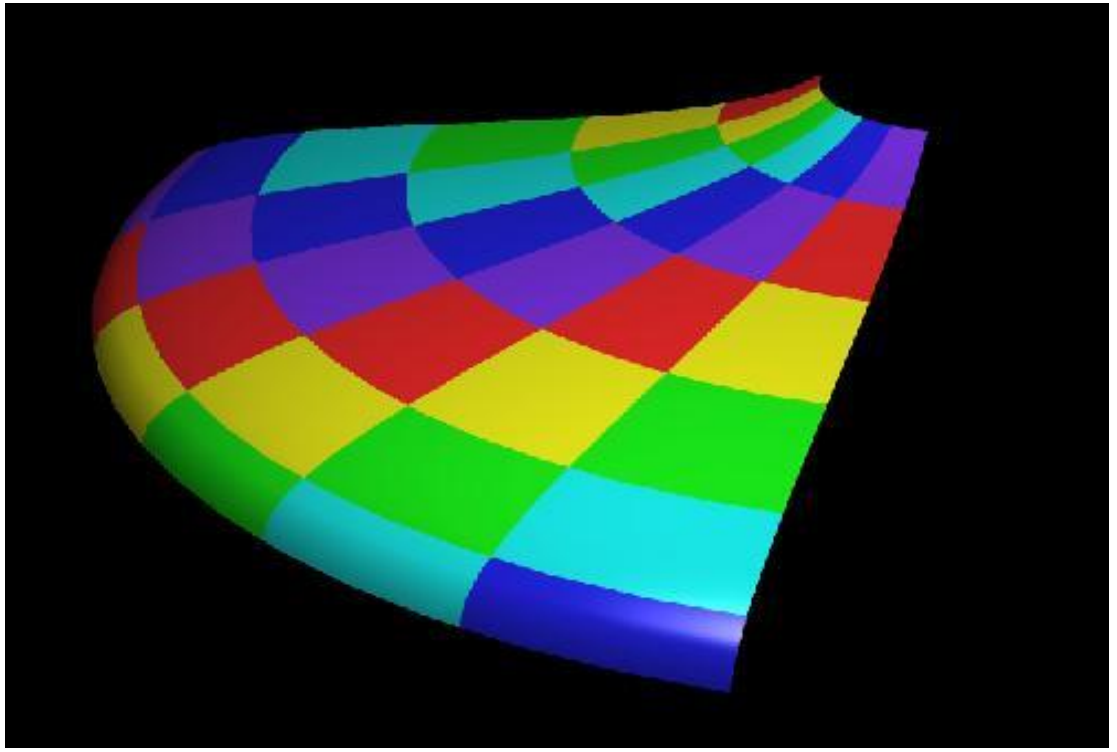
- Convert rectangular coordinates (x, y, z) to spherical (θ, ϕ)



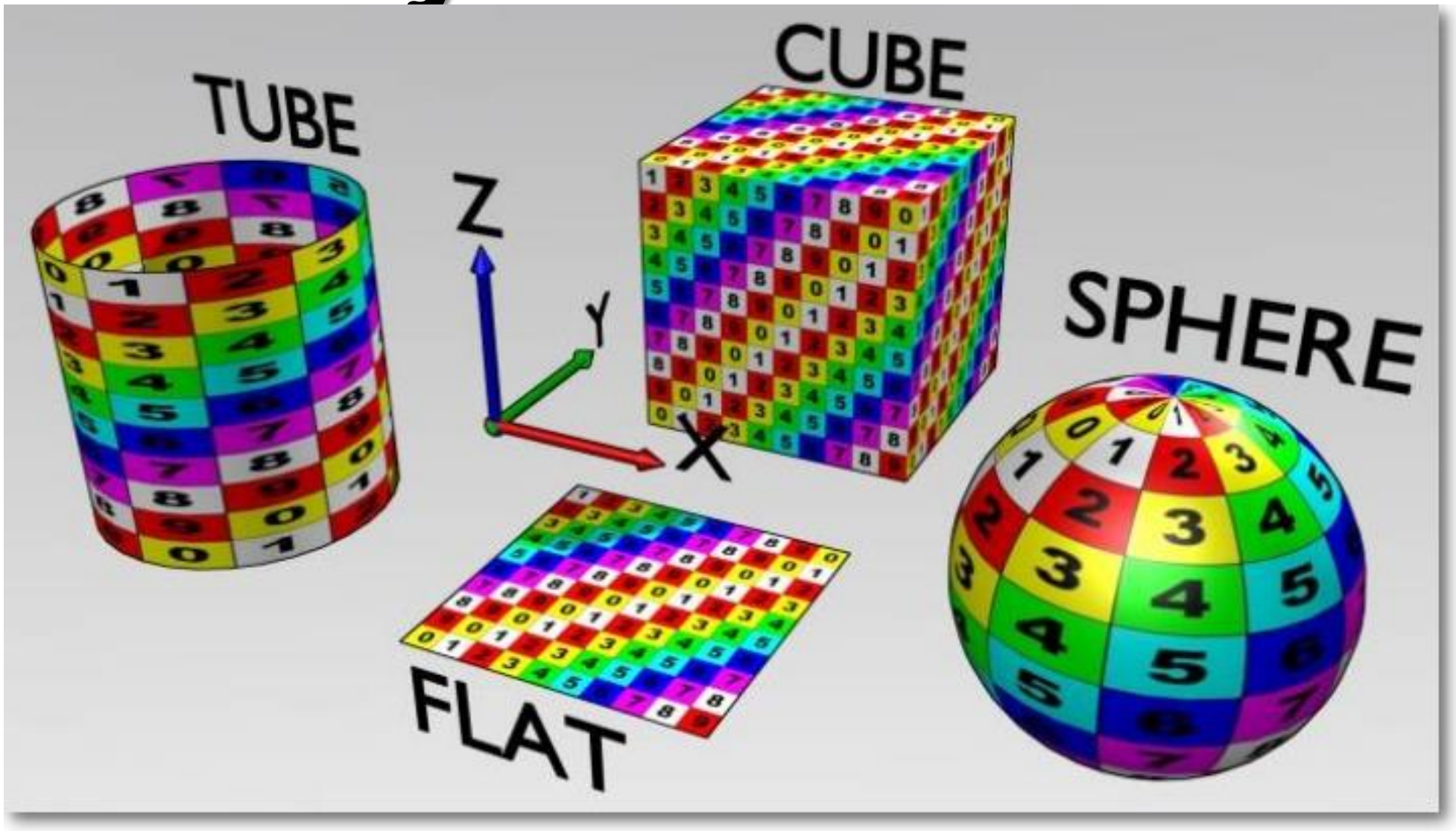
Parametric Surfaces

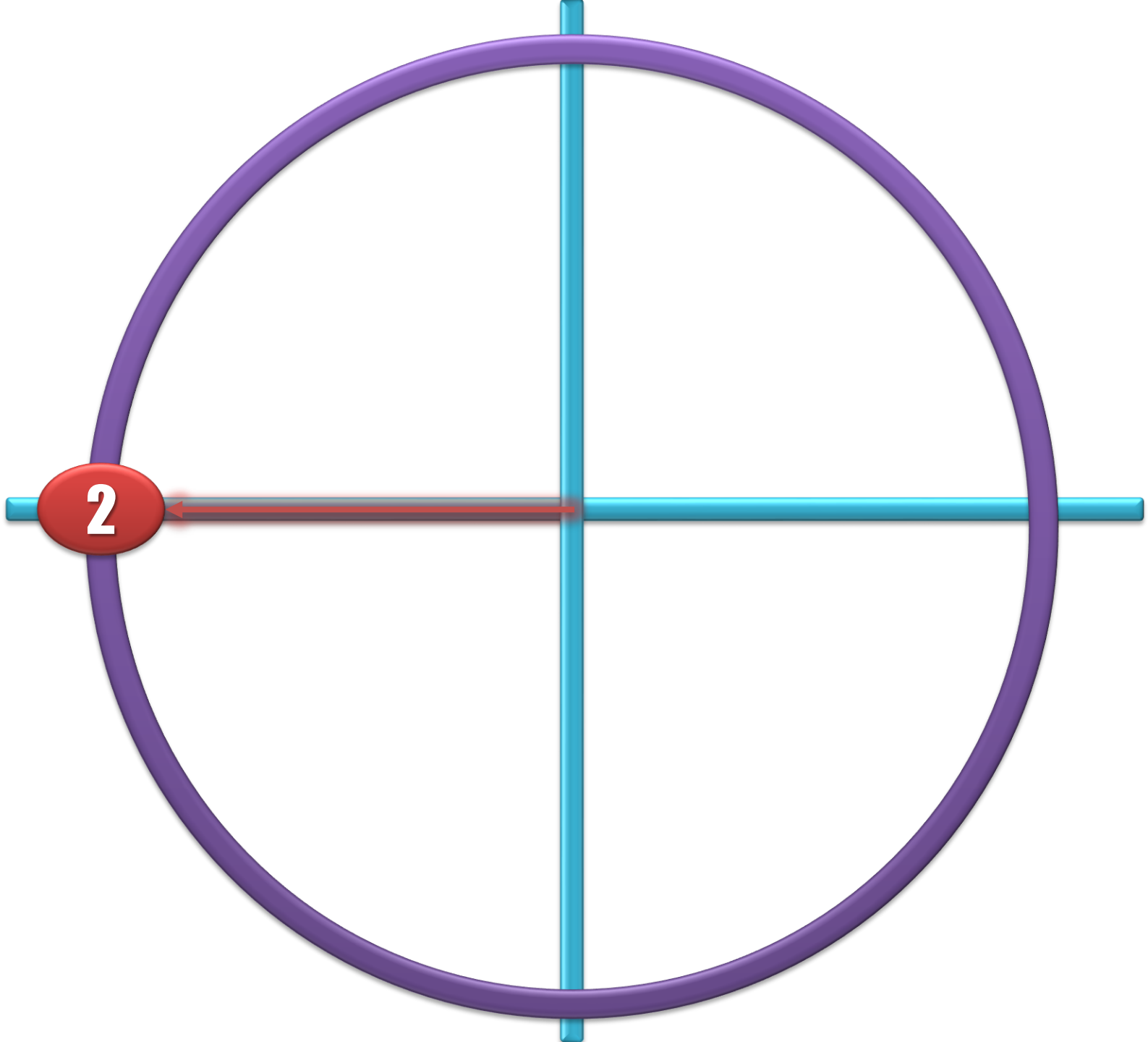
- Surface given by: $x = f(u, v)$, $y = g(u, v)$, $z = h(u, v)$





Summary:




2

Sampling

Sampling Texture

- A pixel maps to a non-rectangular region
- Perform map on center of pixels

Problems

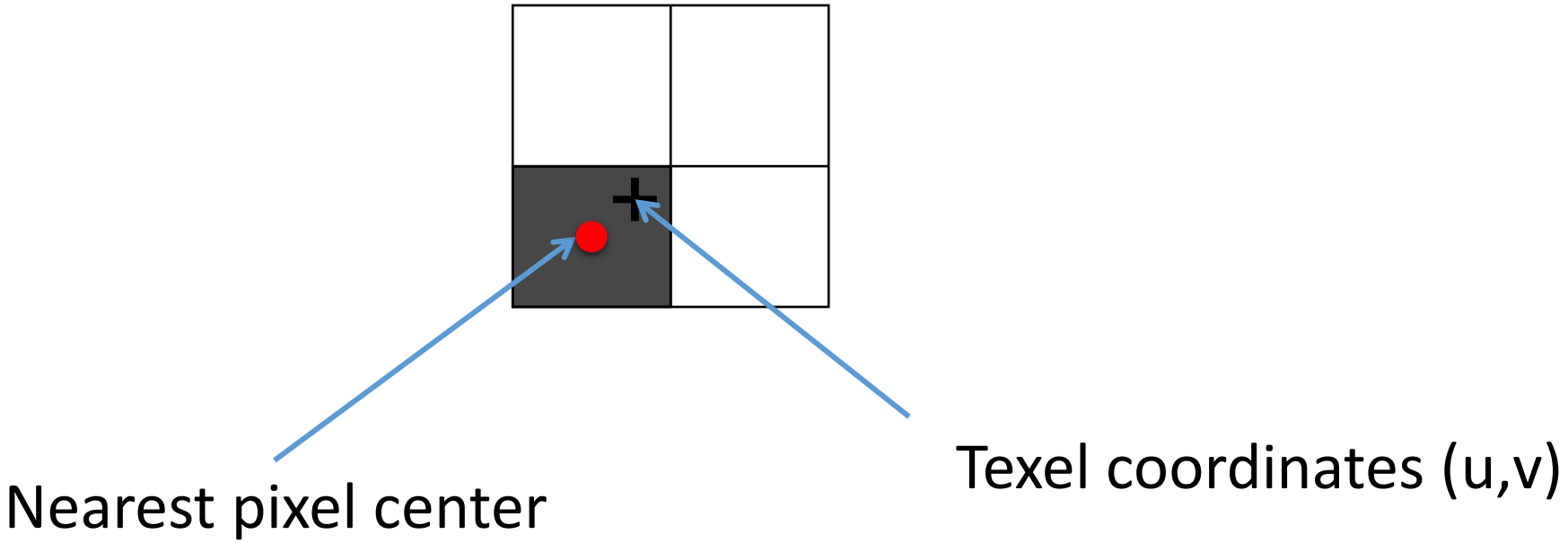
- Over-sampling the image:
  - same texel maps to several pixels
- Under-sampling the image:
  - pixels only sparsely sample the texels

Solutions

- Nearest neighbor sampling
- Bilinear sampling
- MipMapping

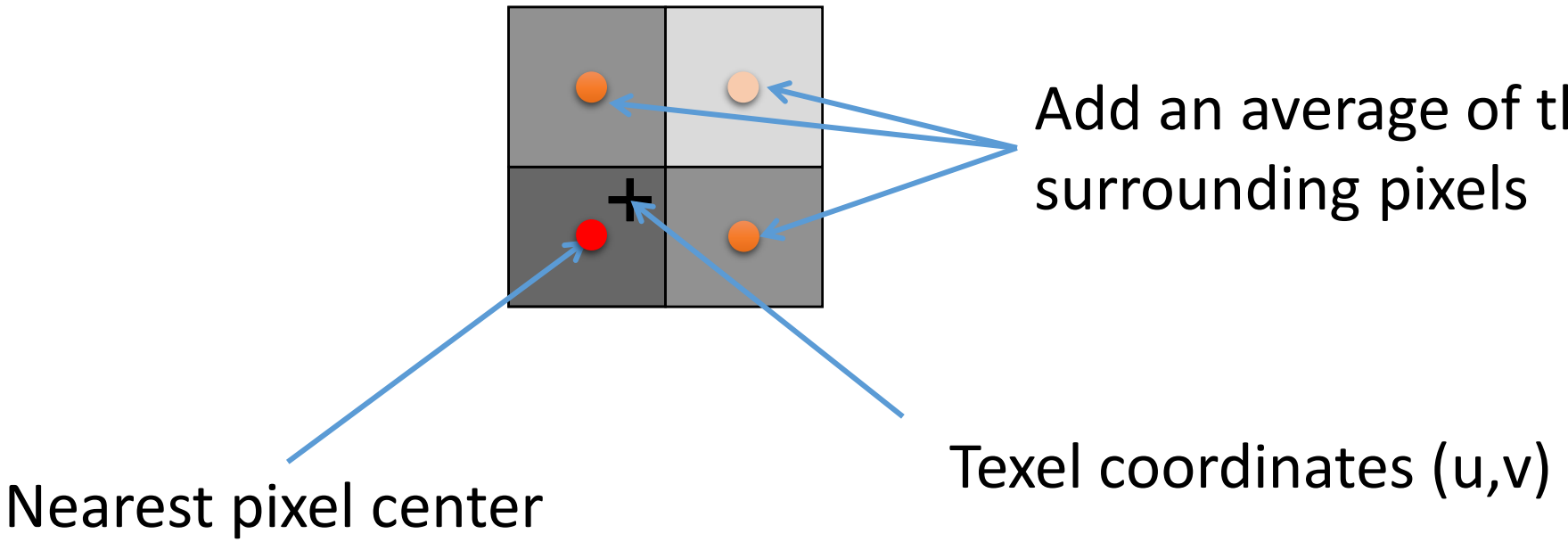
Nearest Neighbor

- Pick the pixel with the nearest center



Bilinear Interpolation

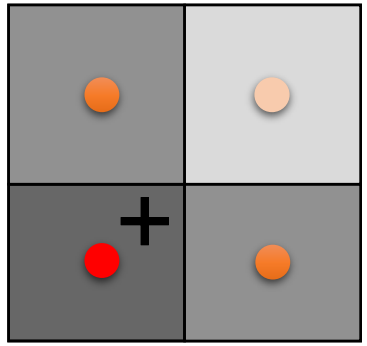
- Linearly interpolate across u and v

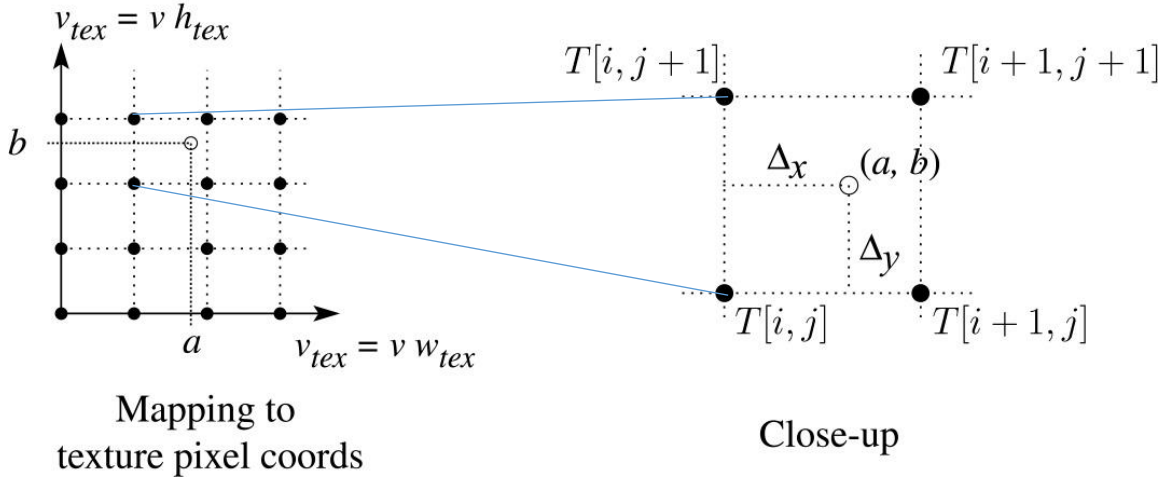


Bilinear Interpolation

- Linear interpolate in both directions

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





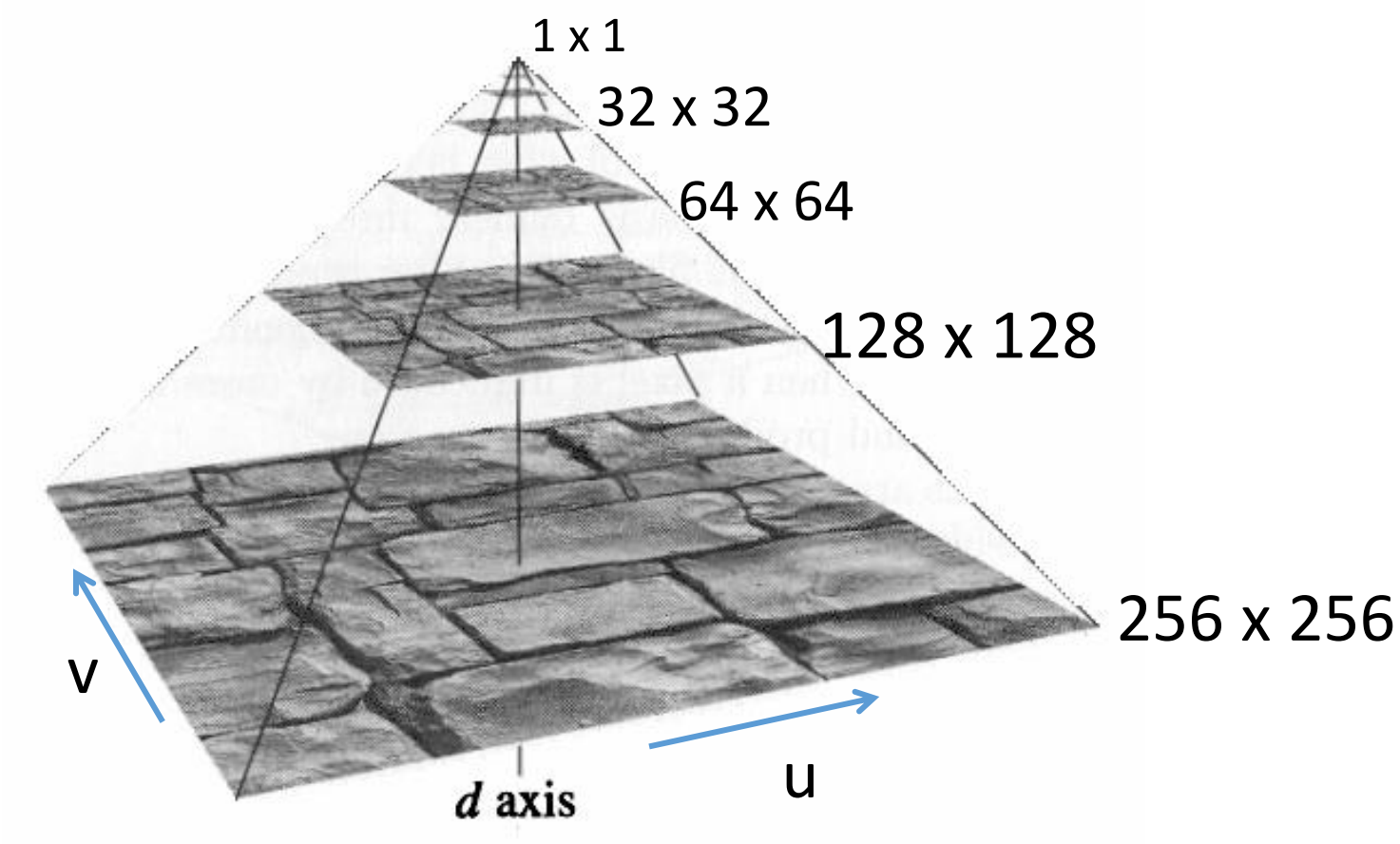
Mapping to texture pixel coords

Close-up

MipMaps

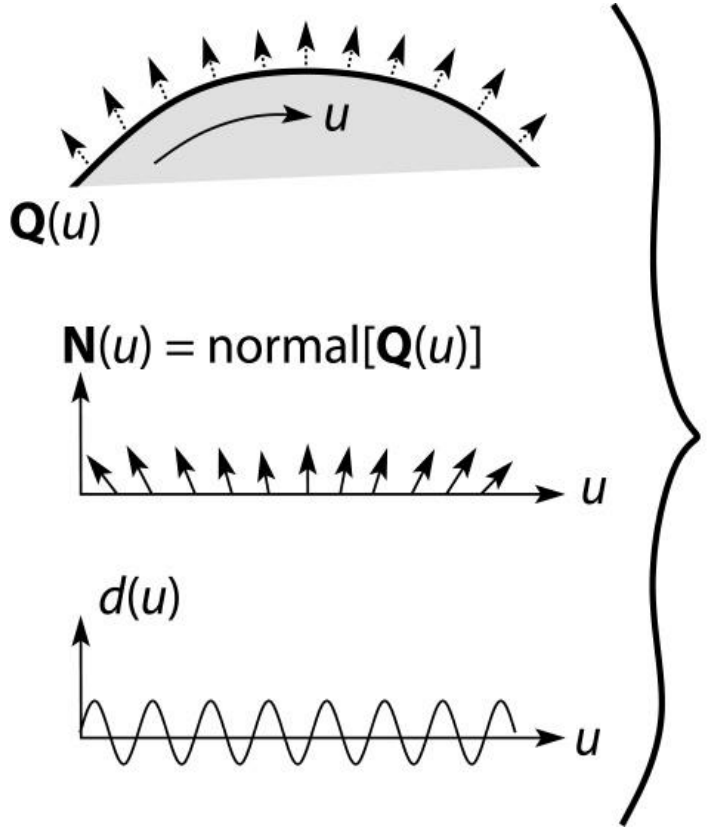
- Texture images must be a power of 2
  - 64, 128, 256, 512, 1024
- Resample the image at lower resolutions
- Create a pyramid of texture images
- Interpolate the texture between two adjacent layers

MipMaps



Bump Mapping

- Perturb the Normal Vector

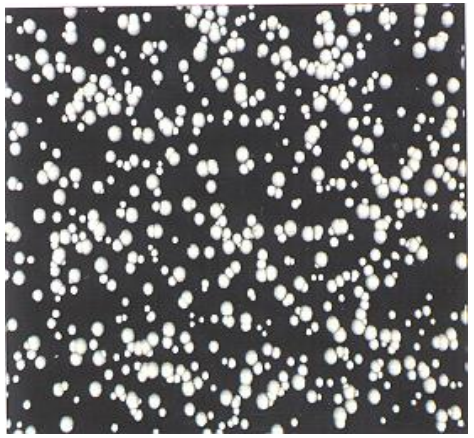


$$\tilde{\mathbf{Q}}(u) = \mathbf{Q}(u) + d(u)\mathbf{N}(u)$$

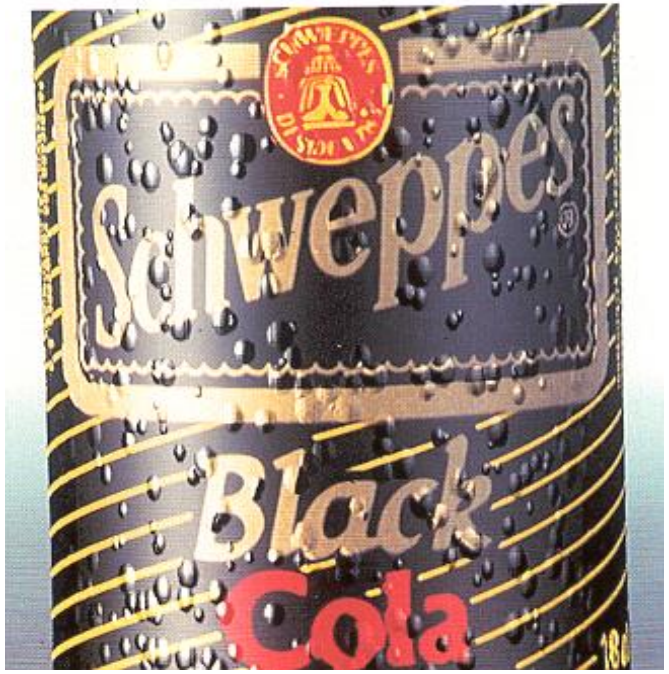
Bump Mapping

Texture 1

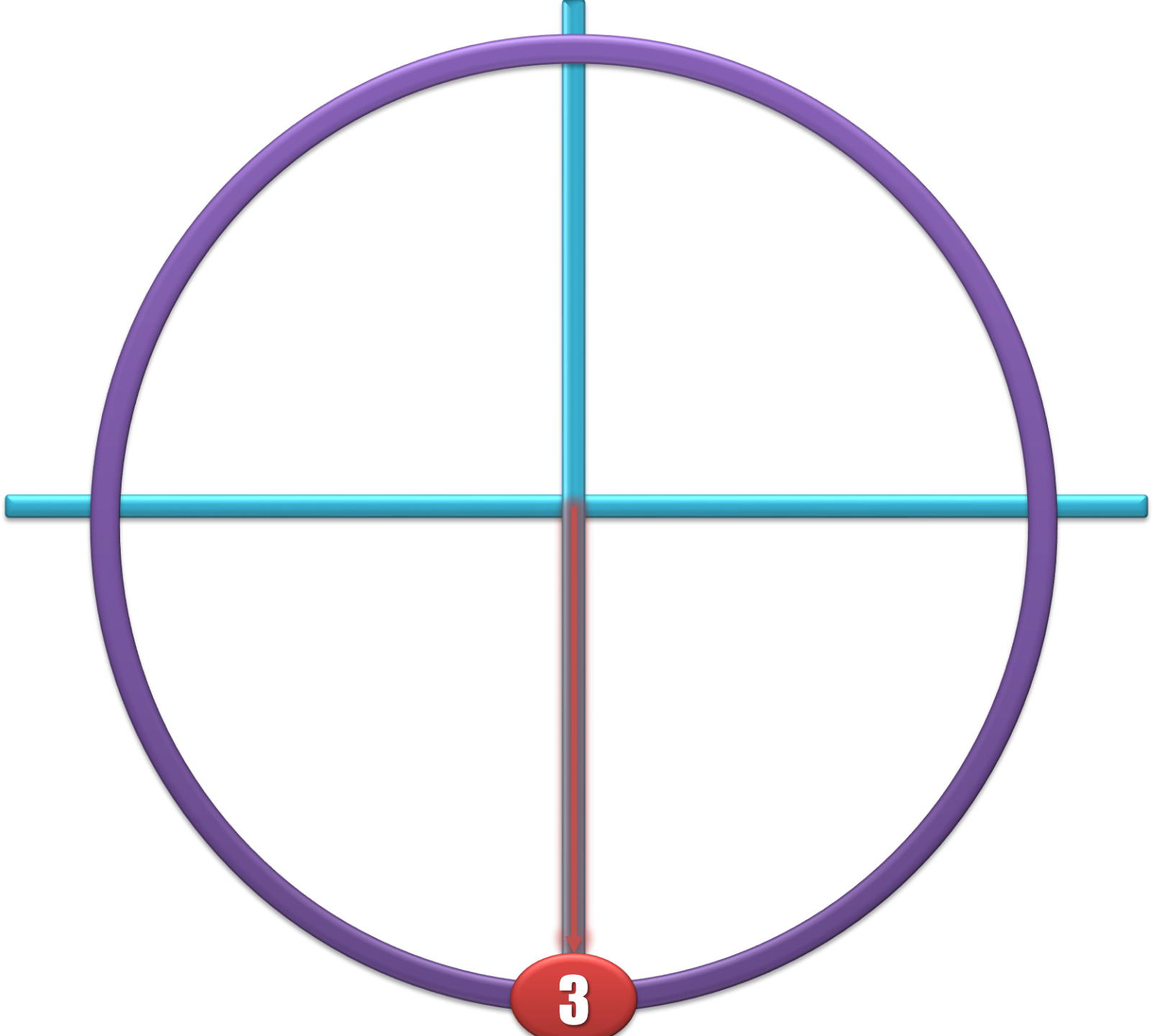
Texture 2
Bump Map



Rendered Image



Environment Mapping


3

WebGL Textures

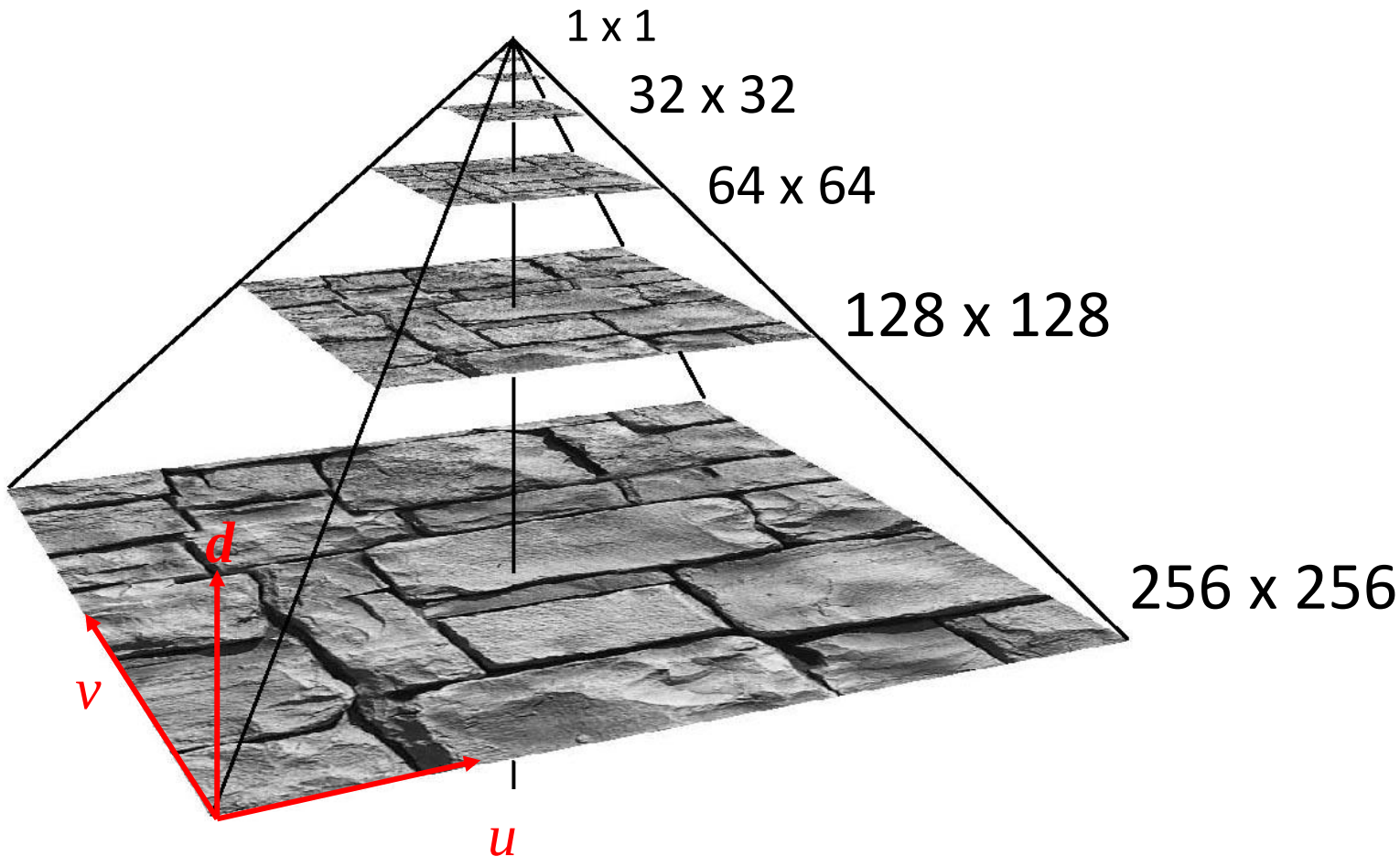
Read Texture Image

```
// 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



NPOT: Not Power-of-Two

- Texture images with sides 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 sides as a 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 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:

```
textureCoordAttribute =
gl.getAttribLocation(shaderProgram, "aTextureCoord");
gl.enableVertexAttribArray(textureCoordAttribute);
```

Update the VS

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

Update the FS

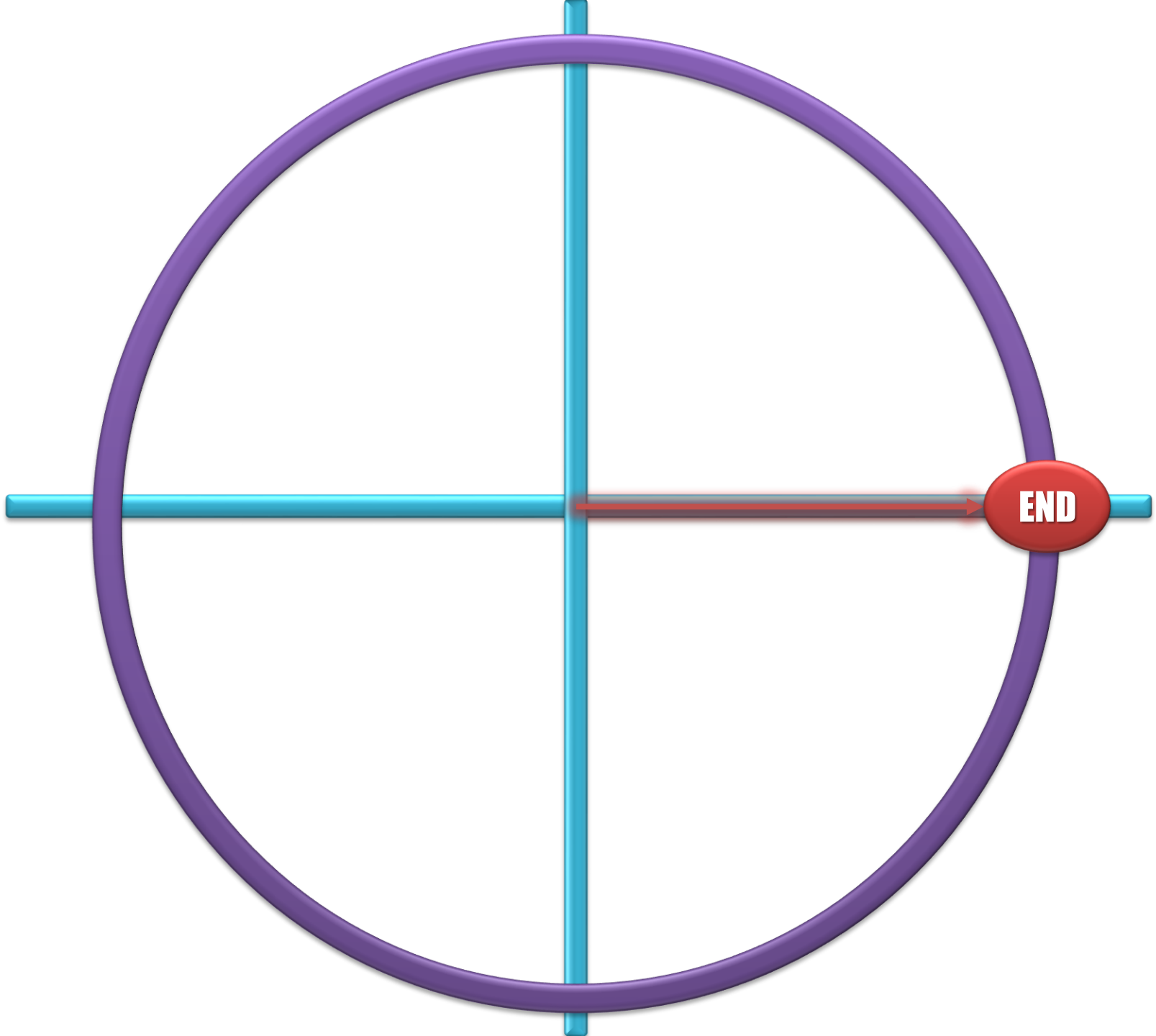
```
<script id="shader-fs" type="x-shader/x-fragment">
  varying highp vec2 vTextureCoord;  // passed from VS
  uniform sampler2D uSampler; // holds the texture image
  void main(void) {
        gl_FragColor = texture2D(uSampler,
                    vec2(vTextureCoord.s, vTextureCoord.t));
  }
</script>
```

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);
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.
```


END