CS 418: Interactive Computer Graphics

Texture Mapping

Eric Shaffer

Some slides adapted from Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

The Limits of Geometric Modeling

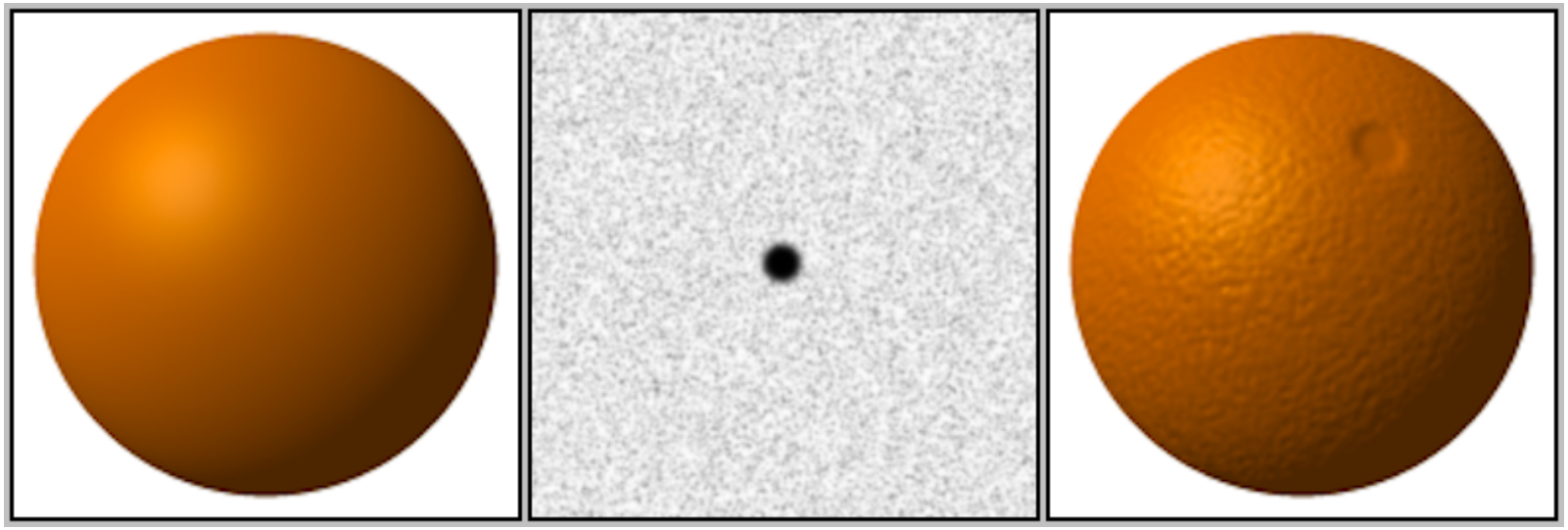
- Graphics cards can render over 10 million polygons per second
- That number is still insufficient for many visual phenomena
- Consider rendering a herd of bumpy-skinned dinosaurs

Or Consider Modeling an Orange

- Consider the problem of modeling an orange (the fruit)
- Start with an orange-colored sphere
  - Too simple
- Replace sphere with a more complex shape
  - Does not capture surface characteristics (small dimples)
  - Takes too many polygons to model all the dimples

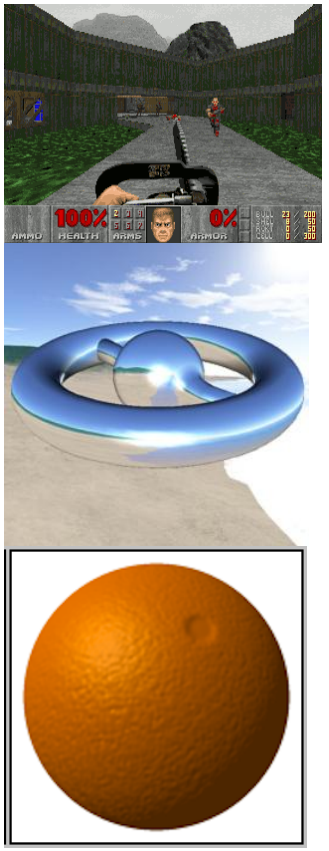
Modeling an Orange

- Take a picture of a real orange; "paste" onto simple geometric model
  - This process is known as texture mapping
- Still might be problematic…resulting surface will be smooth
  - Need to change local shading…use bump mapping

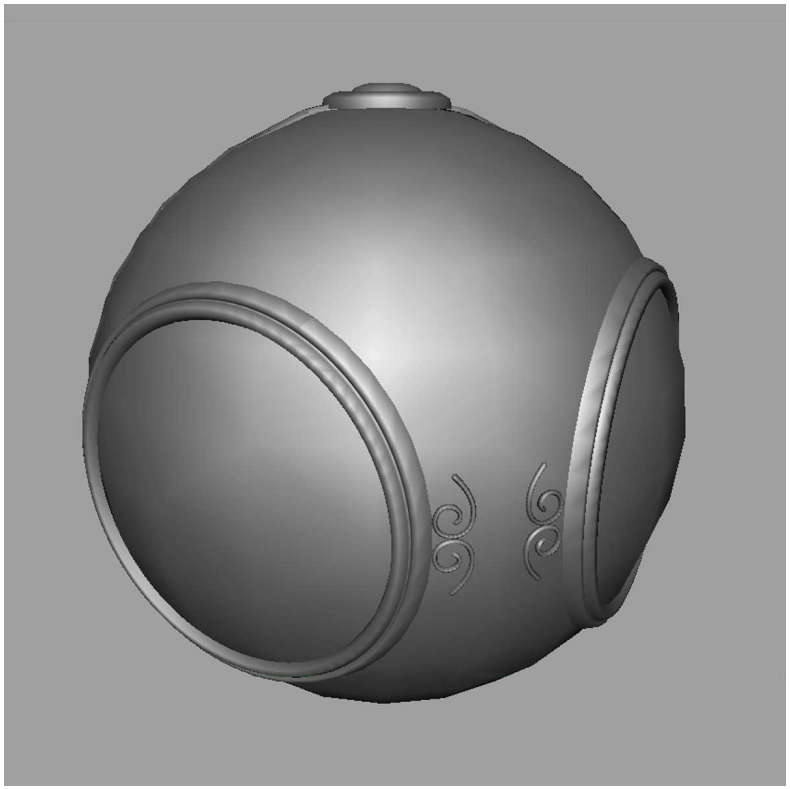


Three Types of Mapping

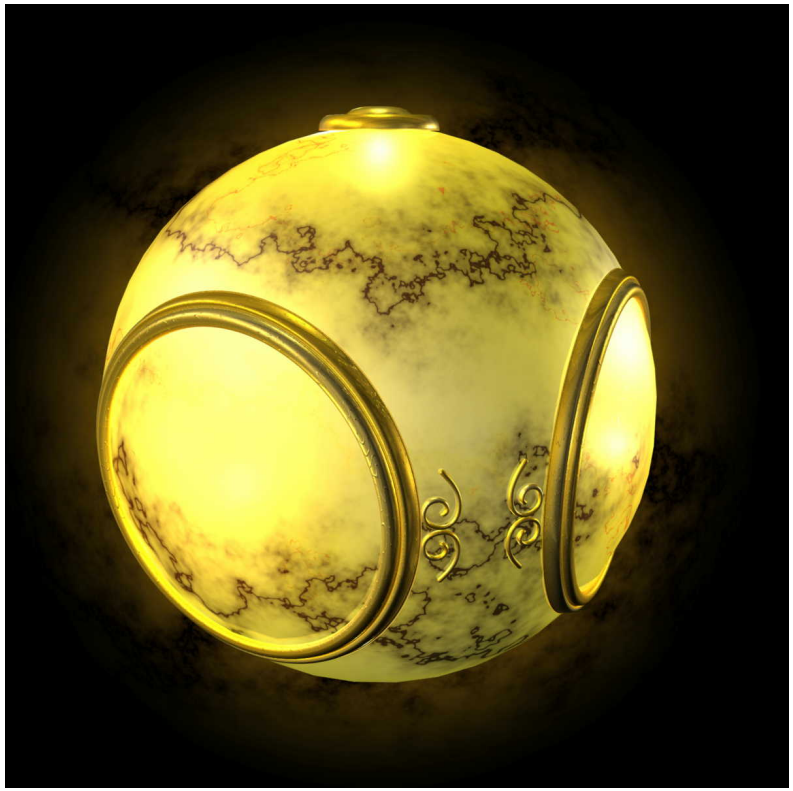
- Texture Mapping
  - Uses images to fill polygons
- Environment Mapping
  - Uses a picture of the environment for texture maps
  - Allows simulation of mirror-like surfaces
- Bump mapping
  - Alters normal vectors during the rendering process
  - Generates a bumpy looking surface



Texture Mapping

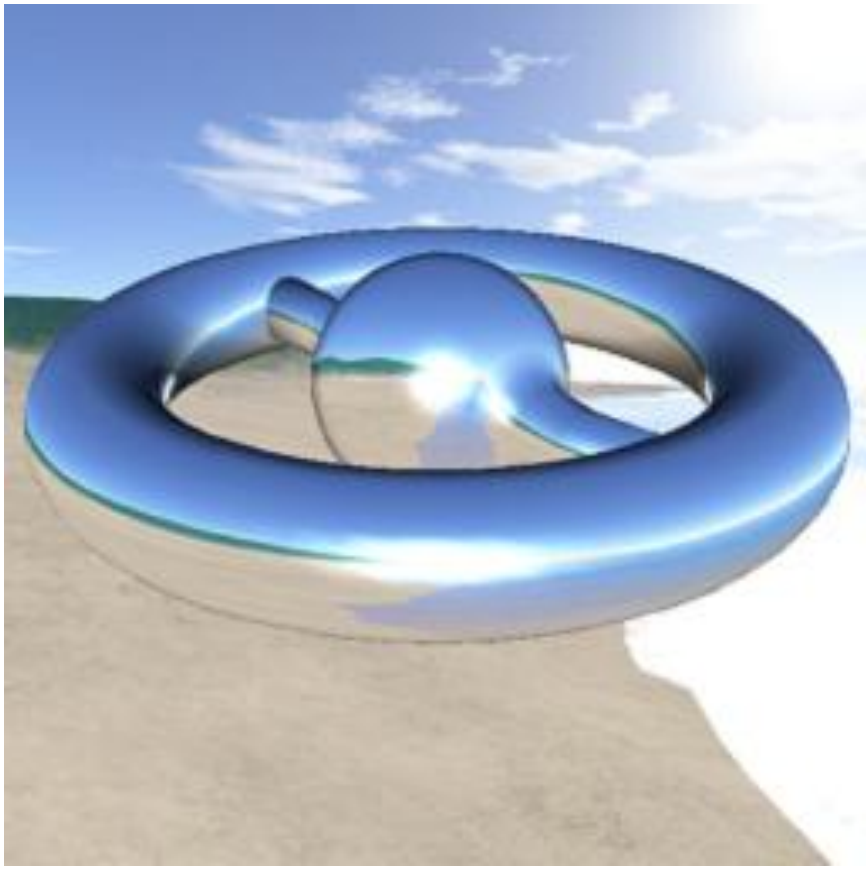


geometric model

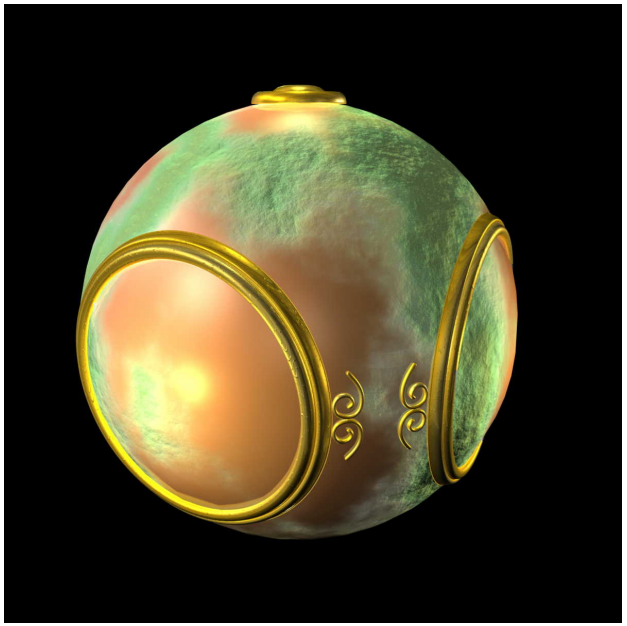


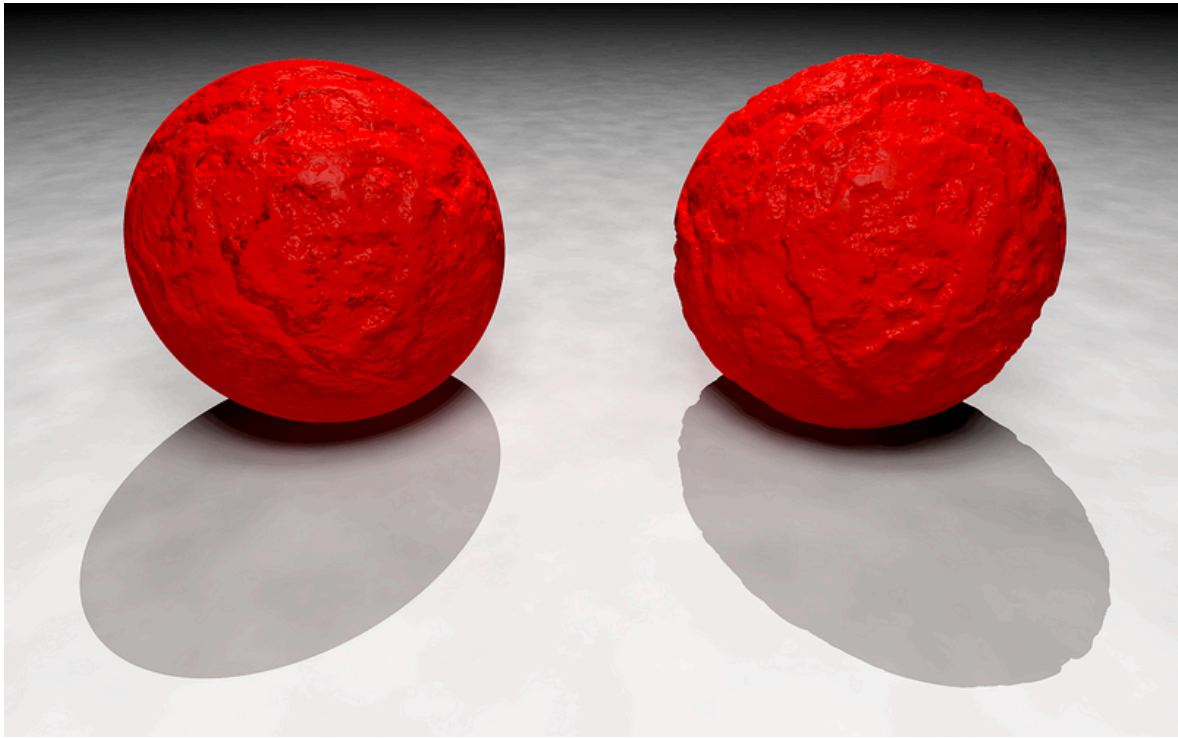
texture mapped

Environment Mapping



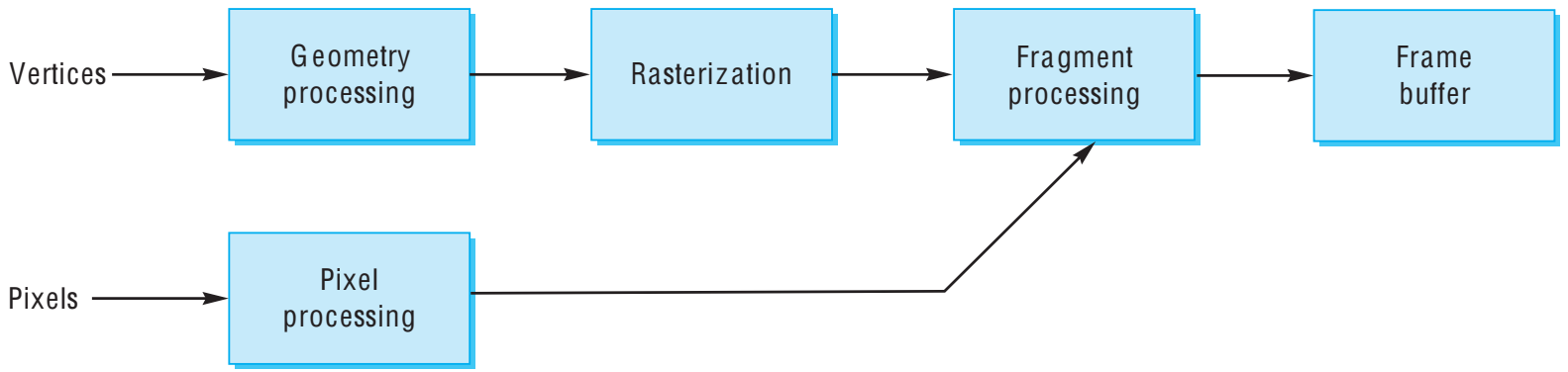
Bump Mapping





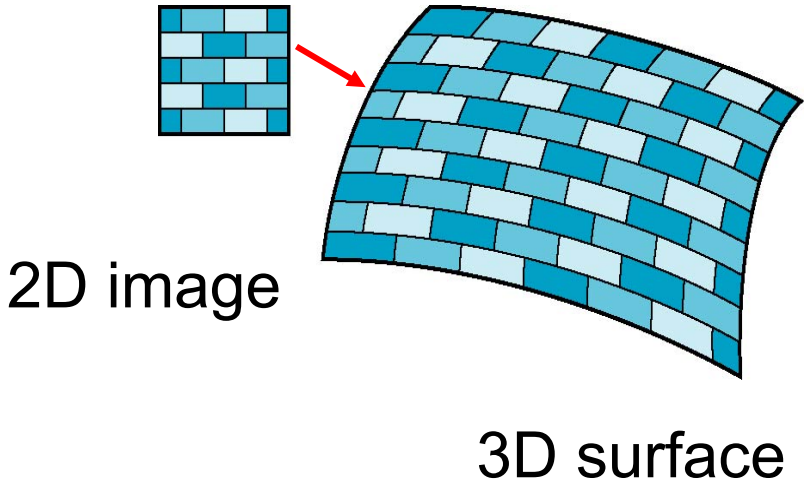
Texturing Mapping and the Pipeline

- In the simplest form *texture mapping* is a process in which
  - A point on a surface to be rendered
  - is mapped to a color in a texture (an image),
  - and then this color (called a texel) is used in shading
- Mapping is implemented in the fragment shader
  - Efficient because relatively few polygons make it past the clipper



The Idea is Simple

- Map an image to a surface
- Each fragment samples a color from the image



- Two things to understand:
  - The code to grab the color from the texture usually is within the library (i.e. you'll call a WebGL function to do it).
  - You then use the color in shading
    - You can modify it according to lighting condition

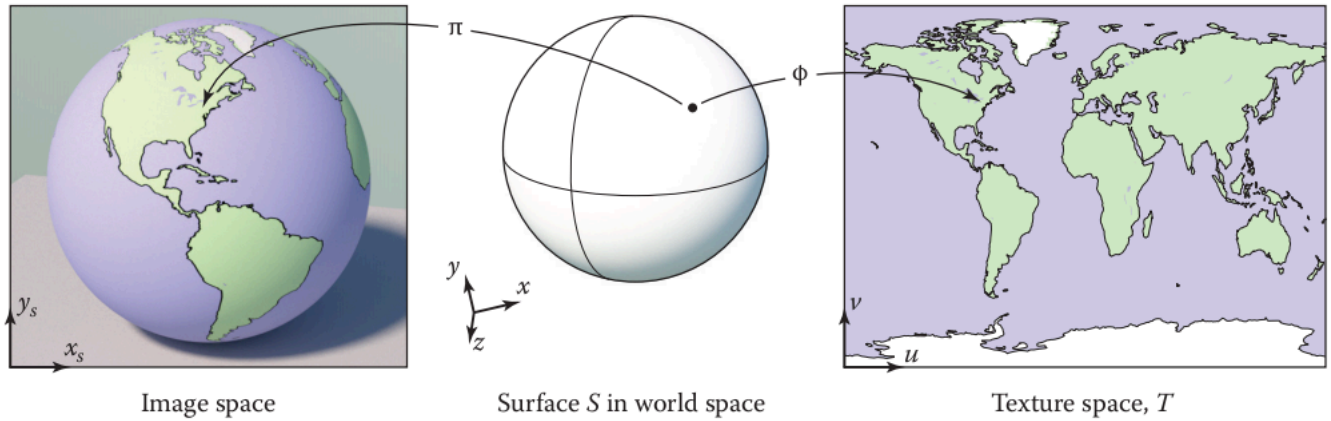
What would the (pseudo)code look like?

```
Color texture_lookup(Texture t, float u, float v)
{ int i = round(u * t.width() - 0.5)
  int j = round(v * t.height() - 0.5)
  return t.get_pixel(i,j)
}

Color shade_surface_point(Surface s,Point p,Texture t)
{
  Vector normal = s.get_normal(p)
  (u,v) =s.get_texcoord(p)
  Color diffuse_color = texture_lookup(u,v)
  // compute shading using diffuse_color and normal
  // return shading result
}
```

Marschner, Steve; Shirley, Peter (2015-11-18).
Fundamentals of Computer Graphics, Fourth Edition

What do you think u and v are?

Texture Coordinate Function



We need a function that maps any point on a surface to a texel

$$\phi: S \rightarrow T: \ (x, y, z) \rightarrow (u, v)$$

Typically $(u, v) \in [0,1]^2$

What is π?

Texture Coordinate Function

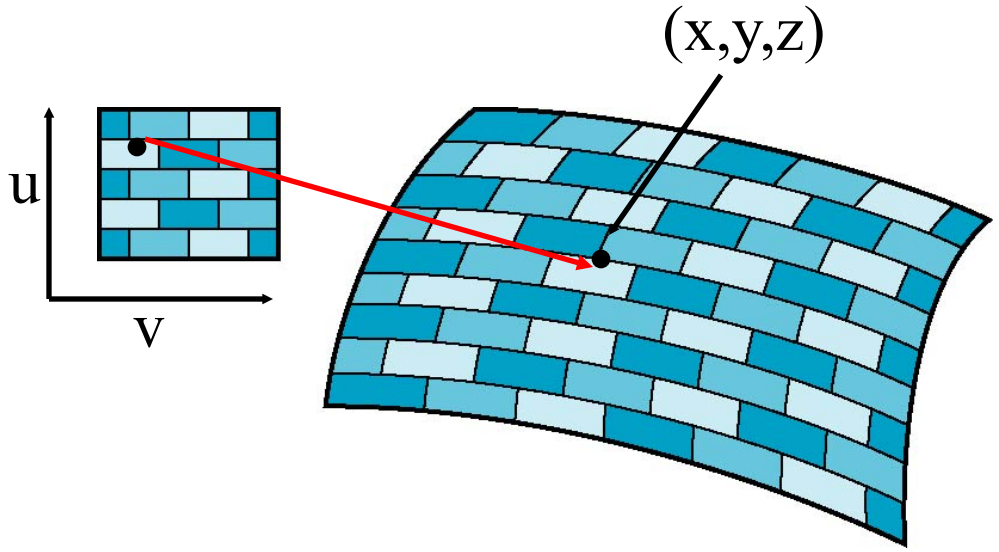
- Consider mapping from texture coordinates to point on surface
- Appear to need three functions

  x = x(u,v)

  y = y(u,v)

  z = z(u,v)

- But we really want to map from a surface point (a fragment) to a texel

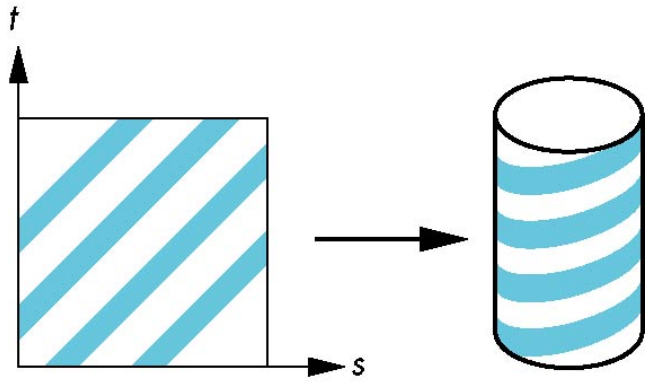


Example: Cylindrical Mapping

You will sometimes see people refer to s,t coordinates

In this class, we will refer to coordinates in the texture image as s,t coordinates

These are in the range $[0,width_{texture}]$ and $[0,height_{texture}]$



$x = r \cos 2\pi s$
$y = r \sin 2\pi s$
$z = t/h$

maps rectangle in s,t space to cylinder of radius r and height h in world coordinates

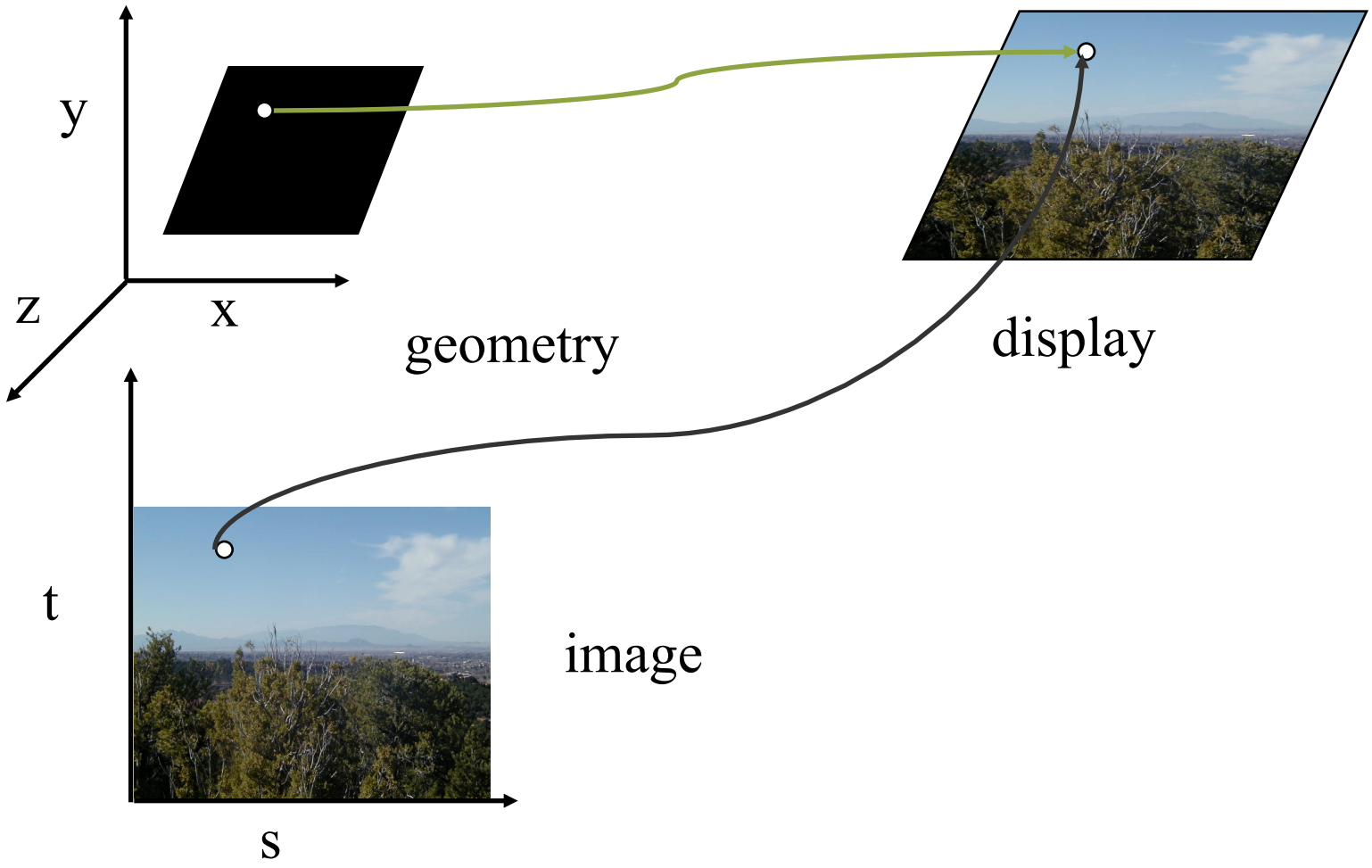
Backward Mapping

- We really want to go backwards
  - Given a fragment, we want to know to which point on an object it corresponds to
  - Given a point on an object, we want to know to which point in the texture (or texel) it corresponds
- Need a map of the form

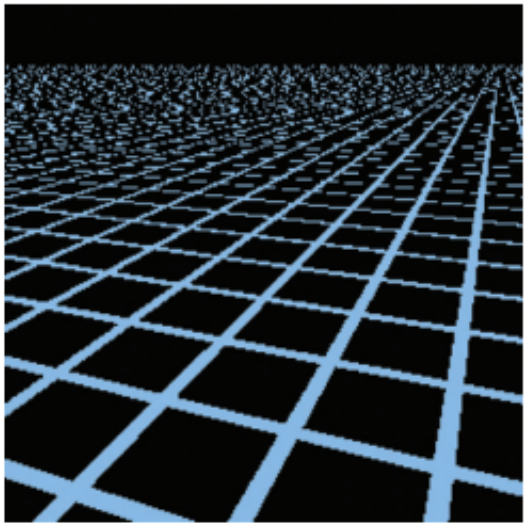
  $u = U(x,y,z)$

  $v = V(x,y,z)$
- To use texture mapping…you will define (u,v) coordinates at the vertices of your surface
  - WebGL will then map them to (s,t) coordinates and sample the texture
  - …and then return a color

Texture Mapping



Issues in Texture Mapping



- The wood floor looks good…but what about the grid texture?

Issues in Texture Mapping

- Two main issues in texture mapping:
  - How to generate texture coordinates
  - How to filter texture samples to remove aliasing artifacts
- We'll look at these in detail in future lectures
  - For, now let's look at the steps you take to use textures in WebGL

Overview: Basic Steps

Three steps to applying a texture

1. **specify the texture**
   - read or generate image
   - assign to texture
   - enable texturing
2. **assign texture coordinates to vertices**
   - proper mapping function is left to application
3. **specify texture parameters**
   - wrapping, filtering

Specifying a Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory
- Use an image in a standard format such as JPEG
  - Scanned image
  - Generate by application code
- WebGL supports only 2 dimensional texture maps
  - no need to enable as in desktop OpenGL
  - desktop OpenGL supports 1-4 dimensional texture maps

Define Image as a Texture

```
gl.texImage2D( target, level, components,
      w, h, border, format, type, texels );
```

`target:` type of texture, e.g. `GL_TEXTURE_2D`

`level:` used for mipmapping (discussed later)

`components:` elements per texel

`w, h:` width and height of `texels` in pixels

`border:` used for smoothing (discussed later)

`format and type:` describe texels

`texels:` pointer to texel array

Example

- Once the image data is loaded into an array
  - We'll discuss how to do that in a future lecture
- You then need to tell WebGL the image is a texture
- WebGL has a concept of the *current* texture
  - gl.bindTexture sets the current texture
  - this is the texture that gets operated on

```
function handleLoadedTexture(texture) {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                  gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Example

- When the image data is given to WebGL you tell it how to "unpack" it
- Here, we tell it to flip the image vertically
  - Most image formats increase coords going down vertical axis
  - WebGL expects the coordinates to increase going up

```
function handleLoadedTexture(texture) {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                  gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Example

- We can give WebGL hints about how to scale the image data
- Need to magnify if we have lots of fragments and few texels
- Need to minify if we have few fragments and lots of texels

```
function handleLoadedTexture(texture) {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                          gl.UNSIGNED_BYTE, texture.image);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
      gl.bindTexture(gl.TEXTURE_2D, null);
   }
```

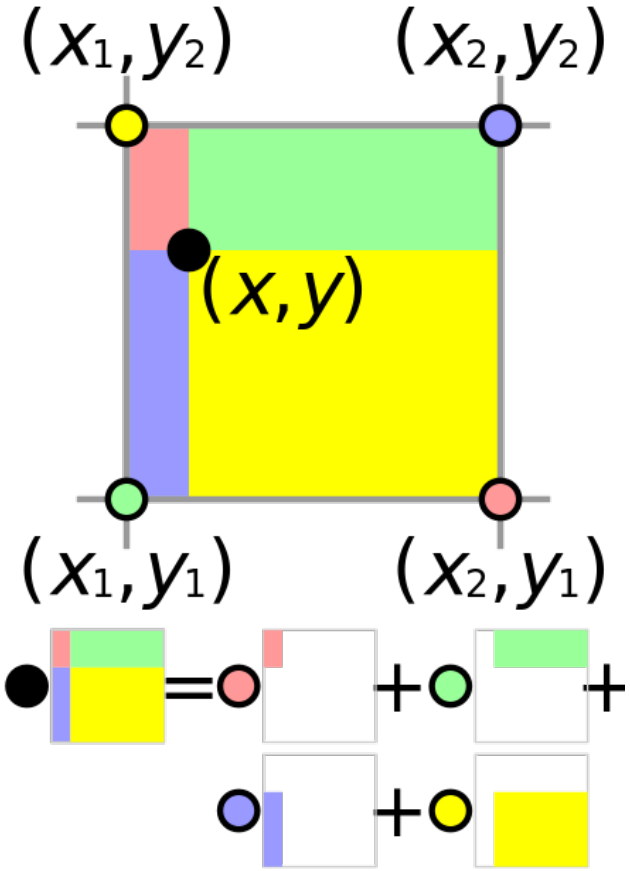
Filtering using Nearest Neighbor

- A (u,v) parametric coordinate mapped to texel coordinates
  - Imagine a 10 x 10 texture
  - e.g. (0.78, 0.22) → (0.78 x 10 - 0.5, 0.22 x 10 - 0.5) → (7.3, 1.7)
  - …but we have only integer texel coordinates
- Nearest Neighbor picks the closest texel in Manhattan Distance
  - e.g. (0.78, 0.22) → (7.3, 1.7) →(7,2)
- Why did we subtract 0.5?

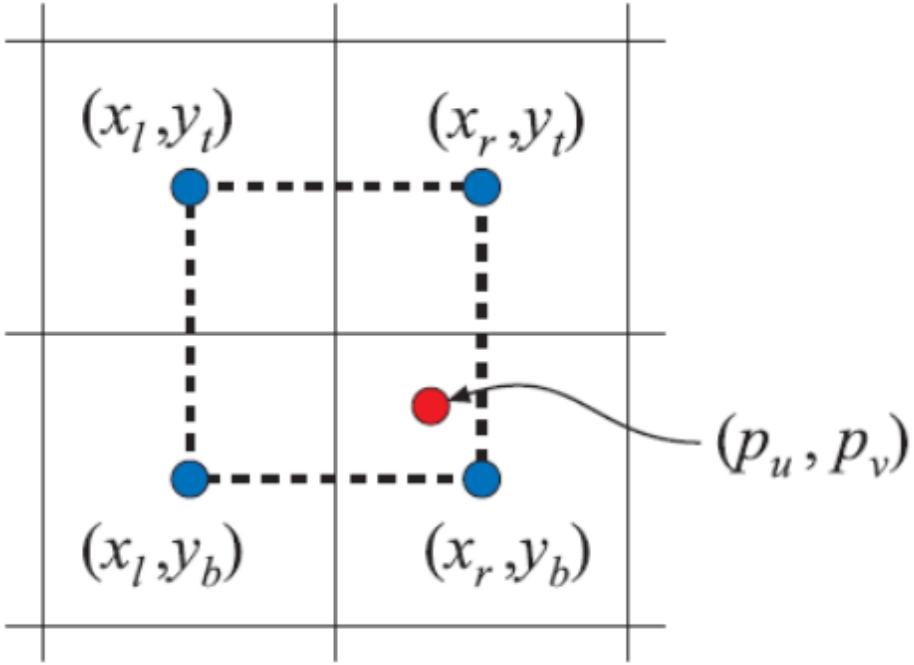
Filtering using Bilinear Interpolation

- Bilinear Filtering blends the 4 closest texels
  - weighted by distance
- Example…let T(s,t) be the color at texel s,t
- For (7.3, 1.7) we blend T(8,2) , T(8,1) , T(7,2) , T(7,1)

Bilinear Interpolation



Bilinear Interpolation



$$(u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor)$$

$$\mathrm{b}(p_u, p_v) = (1-u')(1-v')\,\mathrm{t}(x_l, y_b)$$
$$+u'(1-v')\,\mathrm{t}(x_r, y_b)$$
$$+(1-u')v'\,\mathrm{t}(x_l y_t)$$
$$+u'v'\,\mathrm{t}(x_r, y_t)$$

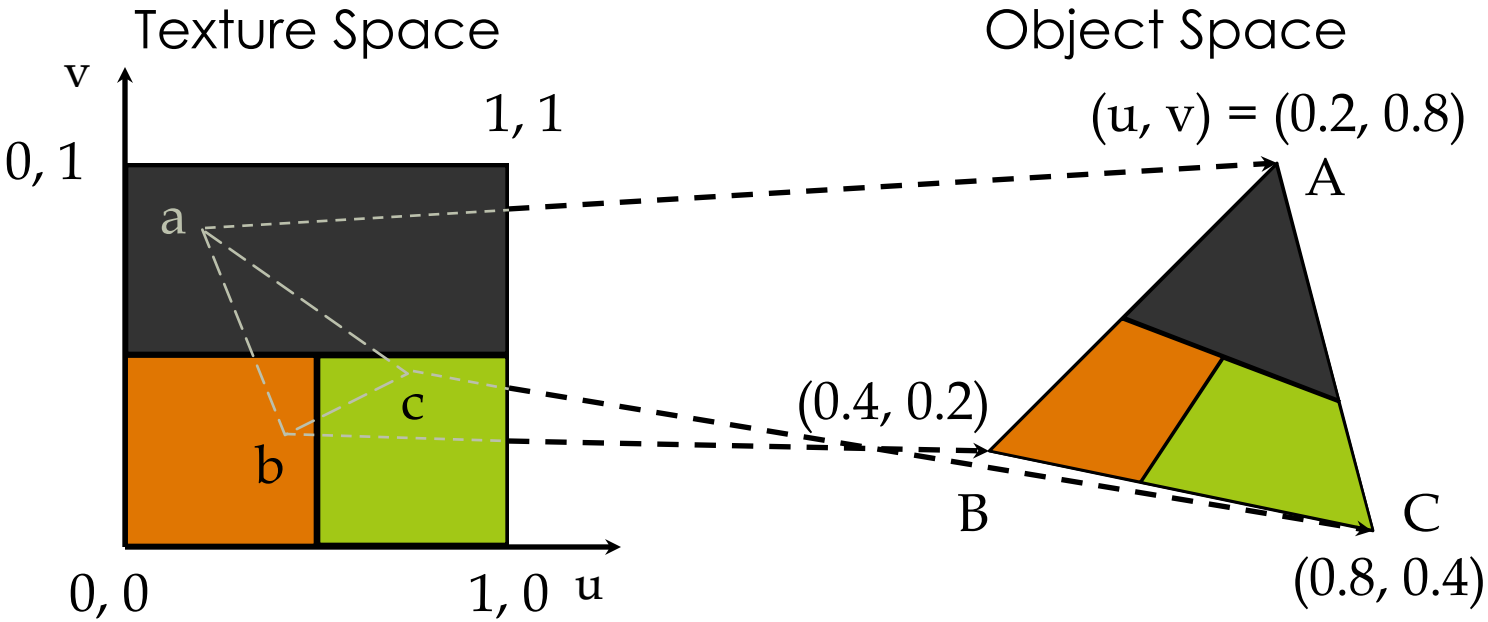
Examples

Nearest neighbor filtering

Bilinear Interpolation

Mapping a Texture

- Based on parametric texture coordinates
- Specify as a 2D vertex attribute



Example

You specify texture coordinates per-vertex…just like normals or colors

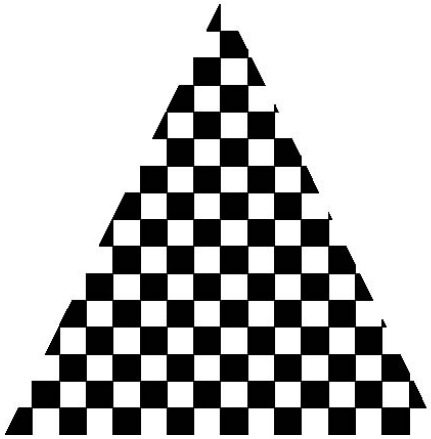
```
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
      colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);
// etc
```

Interpolation

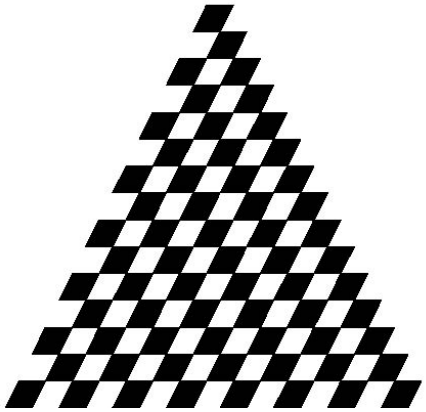
WebGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

good selection of tex coordinates



poor selection of tex coordinates



texture stretched over trapezoid showing effects of bilinear interpolation