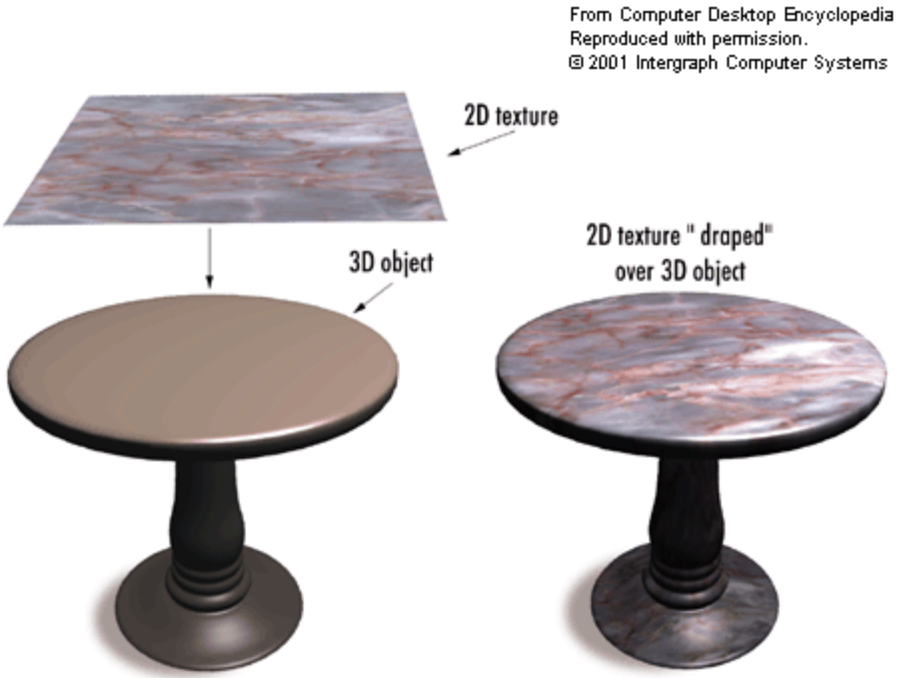
CENG 477
Introduction to Computer Graphics

Textures and Framebuffers

Texture Mapping

- **Goal:** Increase visual realism by using images to simulate reflectance characteristics of objects.
- A cheap and effective way to spatially vary surface reflectance.

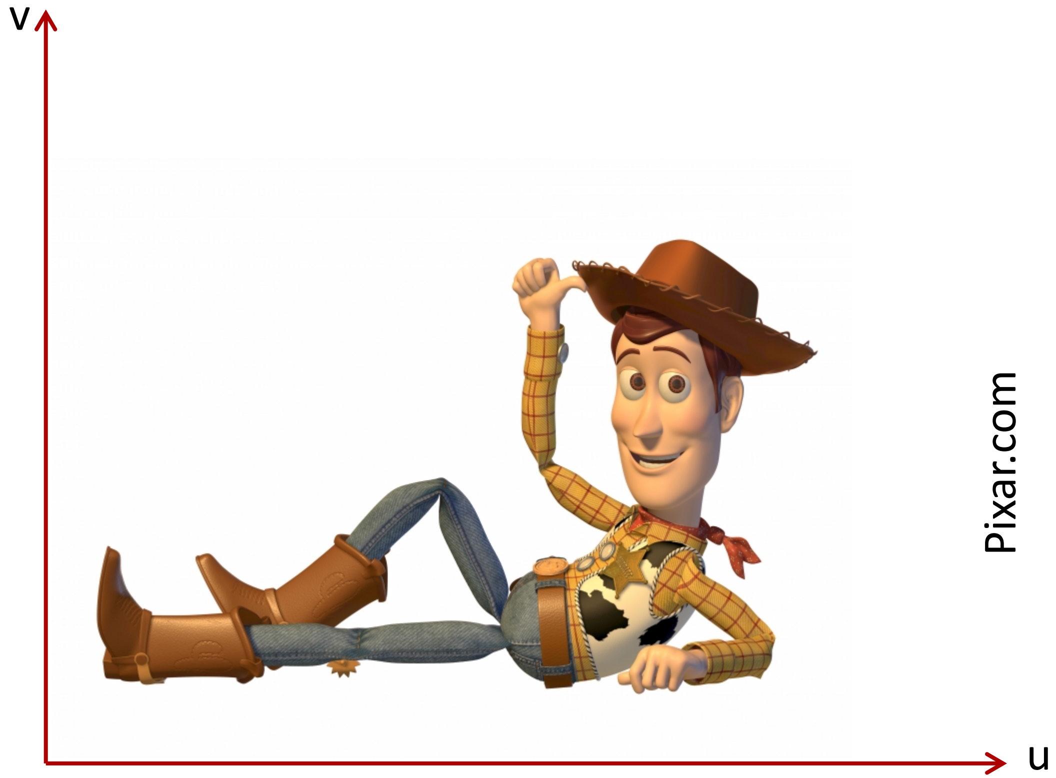


From http://img.tfd.com

The ideas we learned during ray tracing apply here as well!

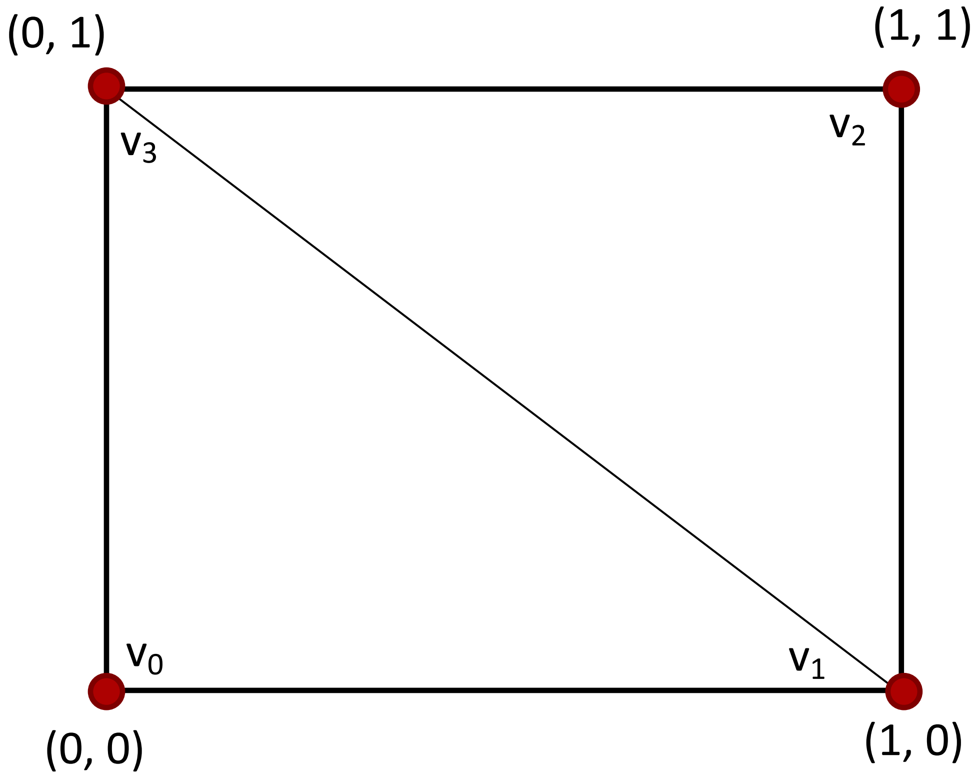
Texture Mapping

- **Step 1:** Associate an (u, v) coordinate system with the texture image where (u, v) ∈ [0,1]x[0,1]



Texture Mapping

- **Step 2:** Parameterize the surface to be texture mapped using two coordinates:



Texture Mapping

- **Step 3:** Compute a (u, v) value for every surface point For a triangle, this can be computed using barycentric interpolation (rasterizer does it for us):

$$u(\beta, \gamma) = u_a + \beta(u_b - u_a) + \gamma(u_c - u_a)$$
$$v(\beta, \gamma) = v_a + \beta(v_b - v_a) + \gamma(v_c - v_a)$$

- **Step 4:** Find the texture image coordinate (i, j) at the given (u, v) coordinate:

$$i = u.n_x$$
$$j = v.n_y$$

Note that i, j can be fractional!

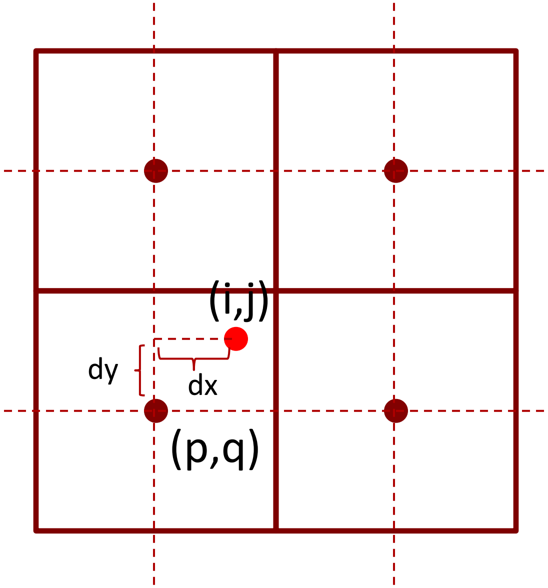
n_x = texture image width
n_y = texture image height

Texture Mapping

- **Step 5:** Choose the texel color using a suitable interpolation strategy
  - **Nearest Neighbor:** fetch texel at the nearest coordinate

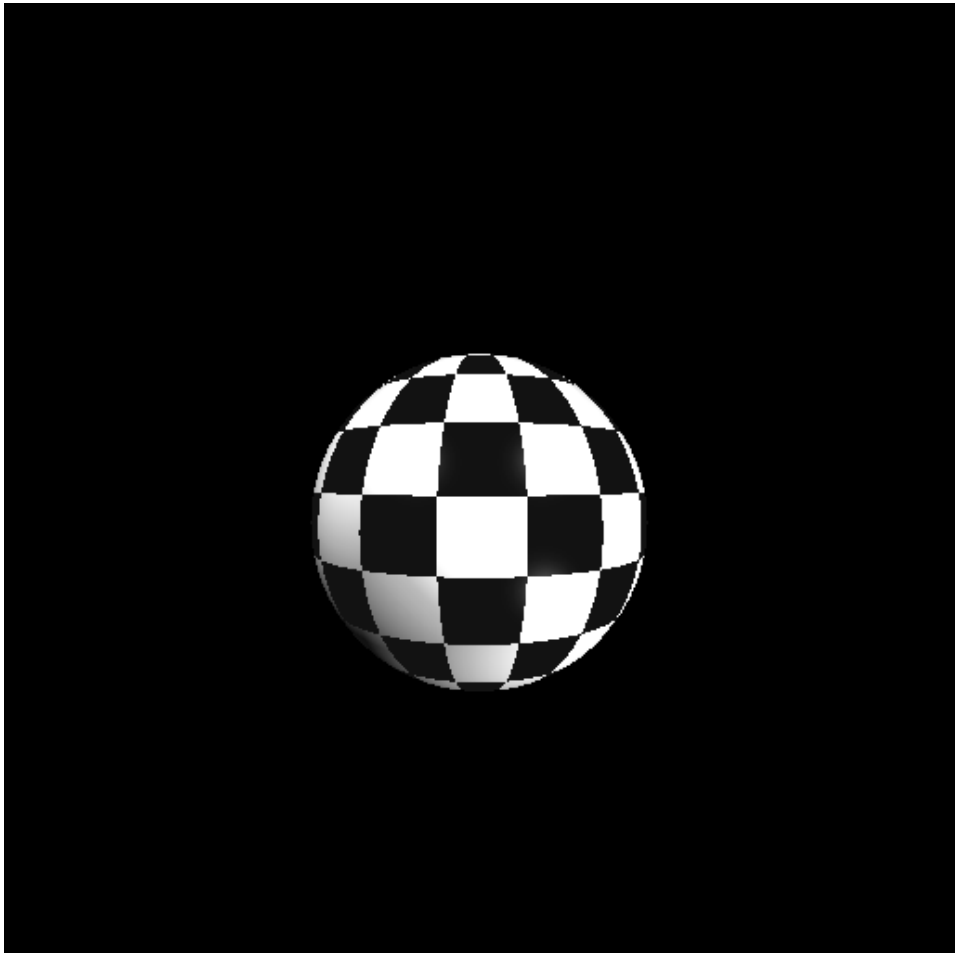
    Color(x, y, z) = fetch(**round(i, j))**

  - **Bilinear Interpolation:** Average four closest neighbors:



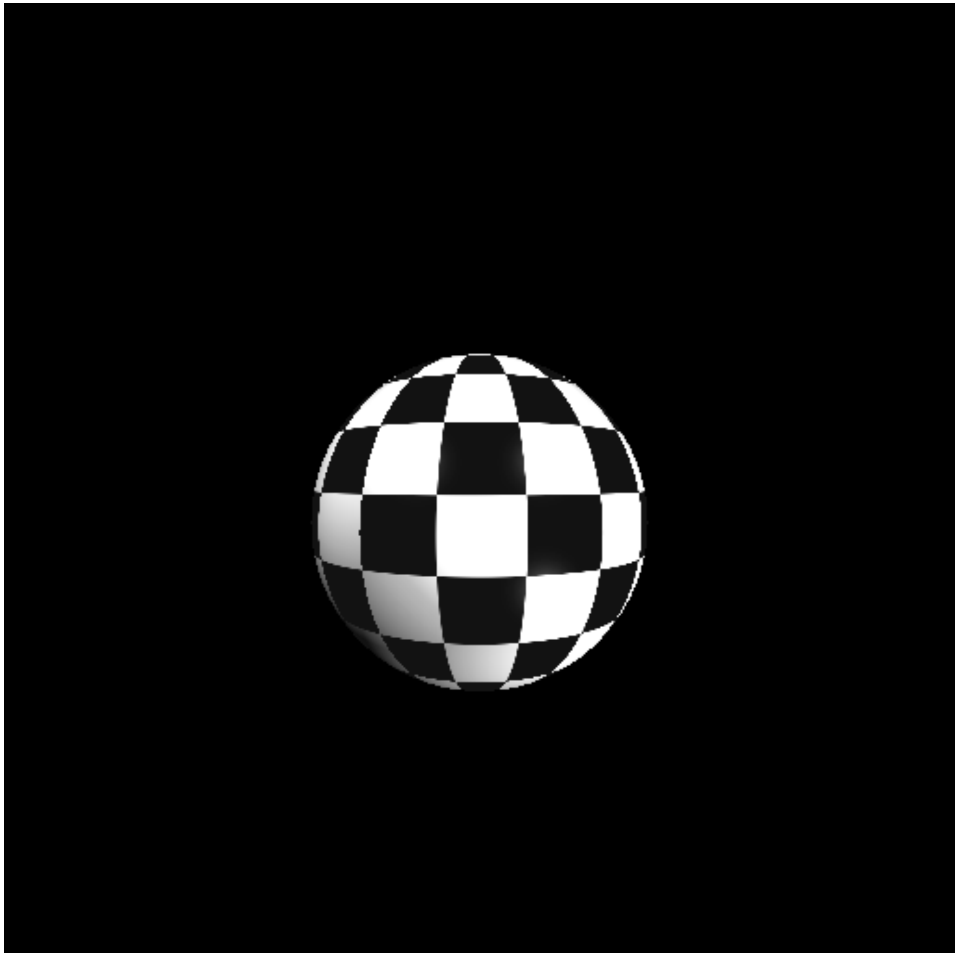
p = floor(i)
q = floor(j)
dx = i – p
dy = j - q
Color(x, y, z) = fetch(p, q).(1 – dx).(1 – dy) +
fetch(p+1, q).(dx).(1 – dy) +
fetch(p, q+1).(1 – dx).(dy) +
fetch(p+1, q+1).(dx).(dy)

NN vs Bilinear Interpolation



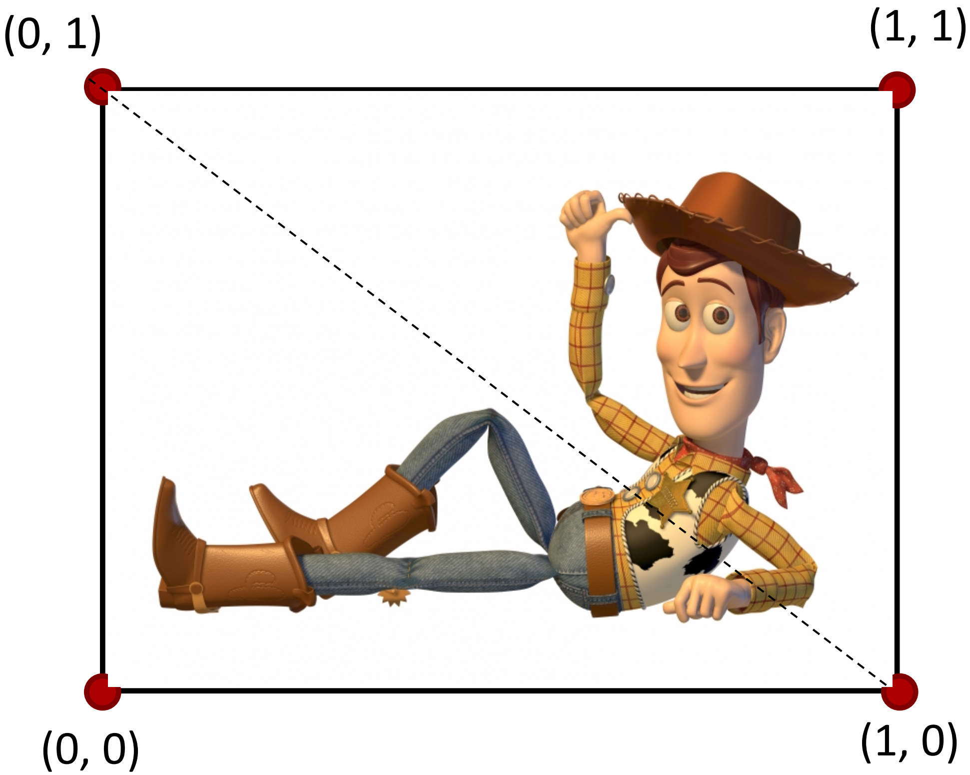
Nearest-neighbor

NN vs Bilinear Interpolation



Bilinear

Result



Texture Mapping using OpenGL

- **Step 1:** Generate a name for your texture and sampler:

```
GLuint mySampler, myTexture;

glGenSamplers(1, &mySampler);
glGenTextures(1, &myTexture);
```

these are just handles to refer to your texture and sampler

Texture Mapping using OpenGL

- **Step 2:** Set your sampling parameters:

```
glSamplerParameteri(mySampler, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glSamplerParameteri(mySampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

What to do if your texture needs to be minified

What to do if your texture needs to be magnified

What to do if you make out-of-bounds access

Texture Mapping using OpenGL

- **Step 3:** Bind your sampler to the desired texture unit:

```
// Bind mySampler to unit 0 so that texture fetches from unit 0
// will be done according to the above sampling properties

glBindSampler(0, mySampler);
```

- **Step 4:** Activate the desired unit and bind your texture to the proper target of that unit as well

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, myTexture);
```

Texture Mapping using OpenGL

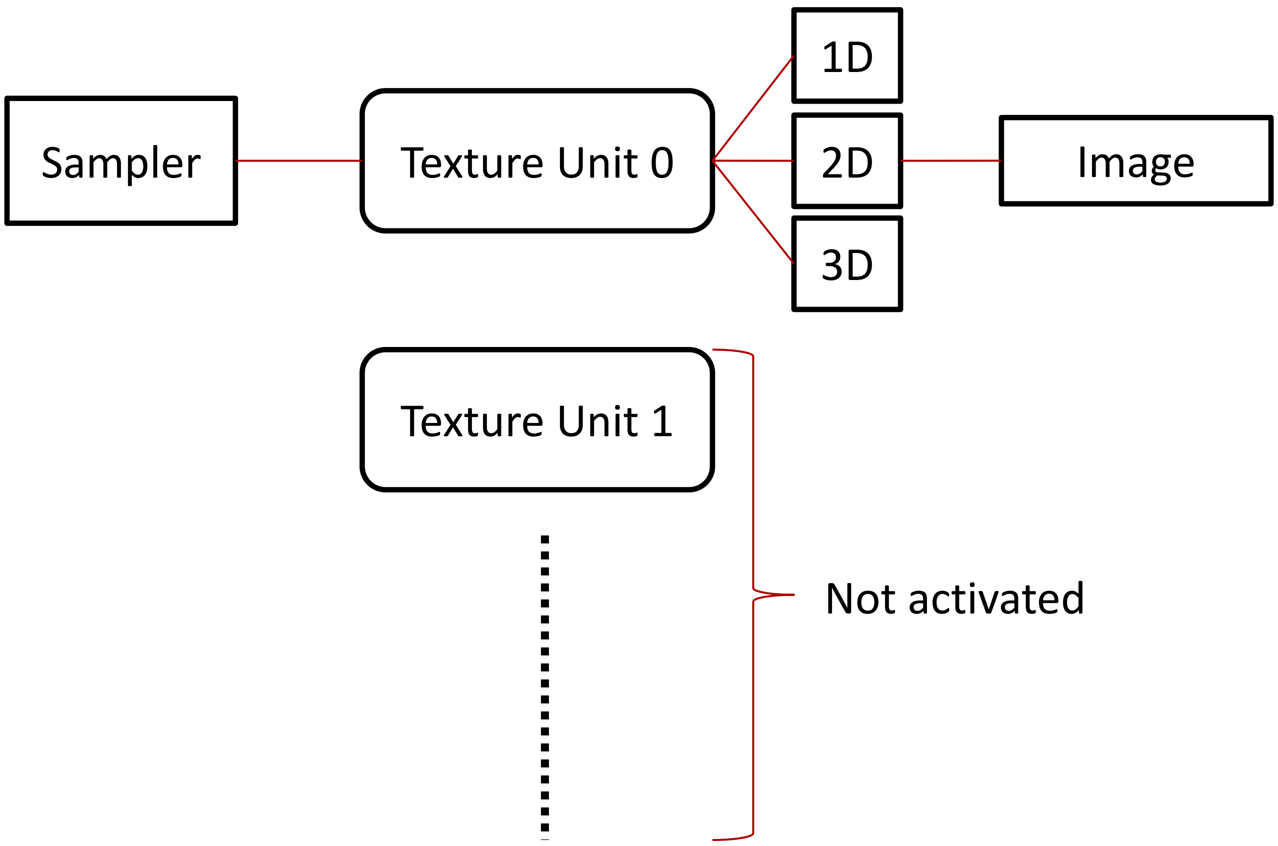
- **Step 5:** Read the texture image from an image file (.jpg, .png, etc.) into a one dimensional array and tell OpenGL about the address of this array:

```
// When reading a texture image, do not assume that it is aligned
// to any boundary larger than a single byte

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);


// Upload the image to the texture

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE, image1D);
```

Pointer to the first byte of your image

Texture Mapping using OpenGL

- At this point we have the following picture:



Texture Mapping using OpenGL

- **Step 6:** Provide *uv* coordinates for each vertex
  - In immediate mode you can use: glTexCoord2f
  - If using vertex arrays, we must provide the texture coordinates in an array (as we did for vertex positions, colors, etc.)
  - As before, this array can be on the system memory or uploaded to GPU memory (remember VBOs)

glTexCoordPointer(size, type, stride, pointer)

Number of coordinates per texture vertex

Type of each coordinate

Byte offset between consecutive texture vertices

Pointer to texture vertex coordinate data

Texture Mapping using OpenGL

- **Step 7:** In the vertex shader, pass along these texture coordinates to the rasterizer:

This is a built-in varying variable
which will be automatically interpolated

```
void main(void)
{
    gl_FrontColor = gl_Color; // vertex color defined by the programmer
    gl_TexCoord[0] = gl_MultiTexCoord0; // pass along to the rasterizer
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

This value comes from the
vertex array whose data
is provided by *glTexCoordPointer*

This value comes from the
vertex array whose data
is provided by *glVertexPointer*

Texture Mapping using OpenGL

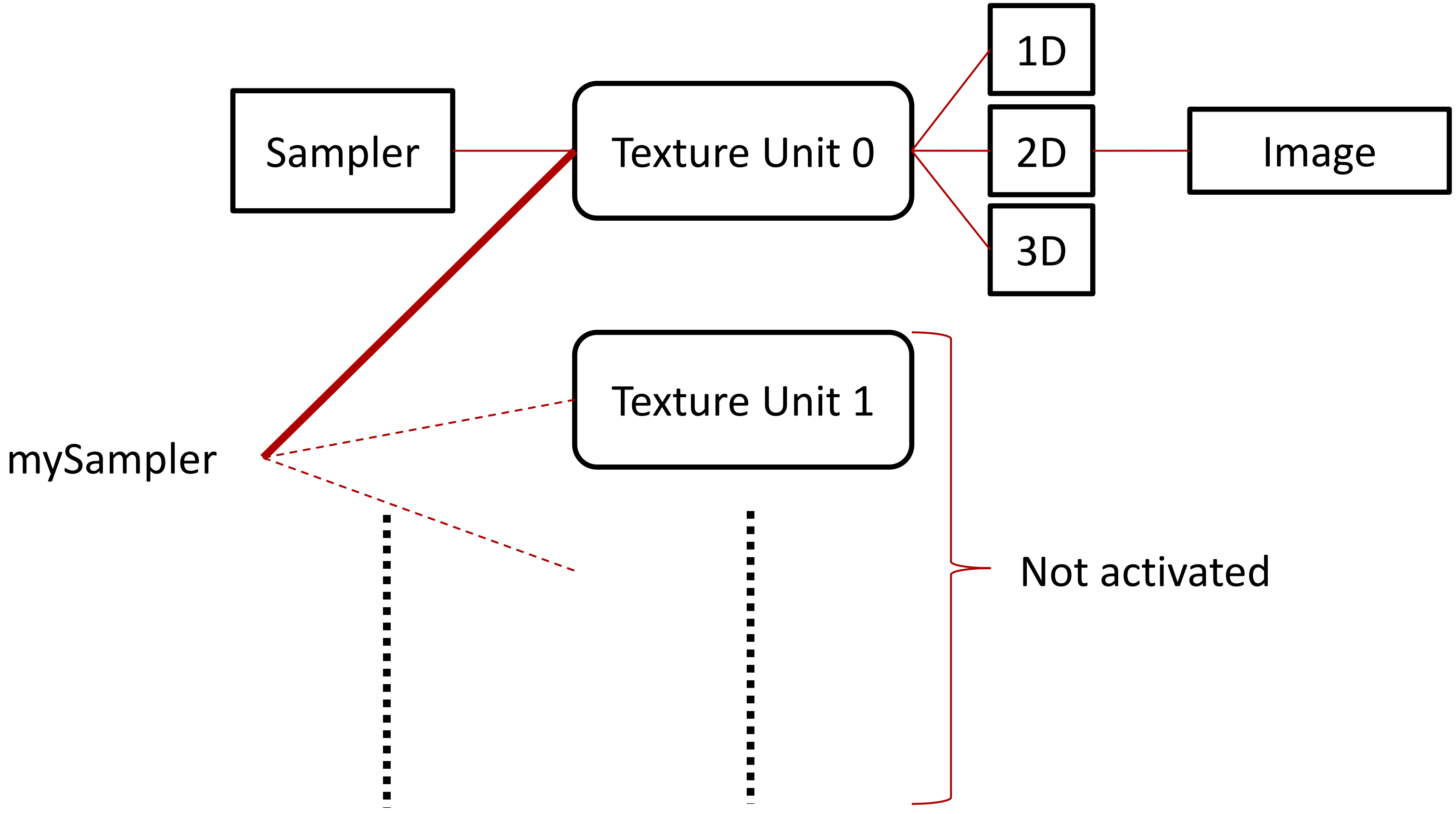
- **Step 8:** In the fragment shader, fetch from the texture image using a suitable sampling method:

```
uniform sampler2D mySampler;

void main(void)
{
    // get the color from the texture
    gl_FragColor = texture2D(mySampler, gl_TexCoord[0].st);
}
```

This variable represents the texture unit index. If its value is zero it will fetch from texture unit 0. Its value is given such as *glUniform1i(mySamplerLoc, 0)*

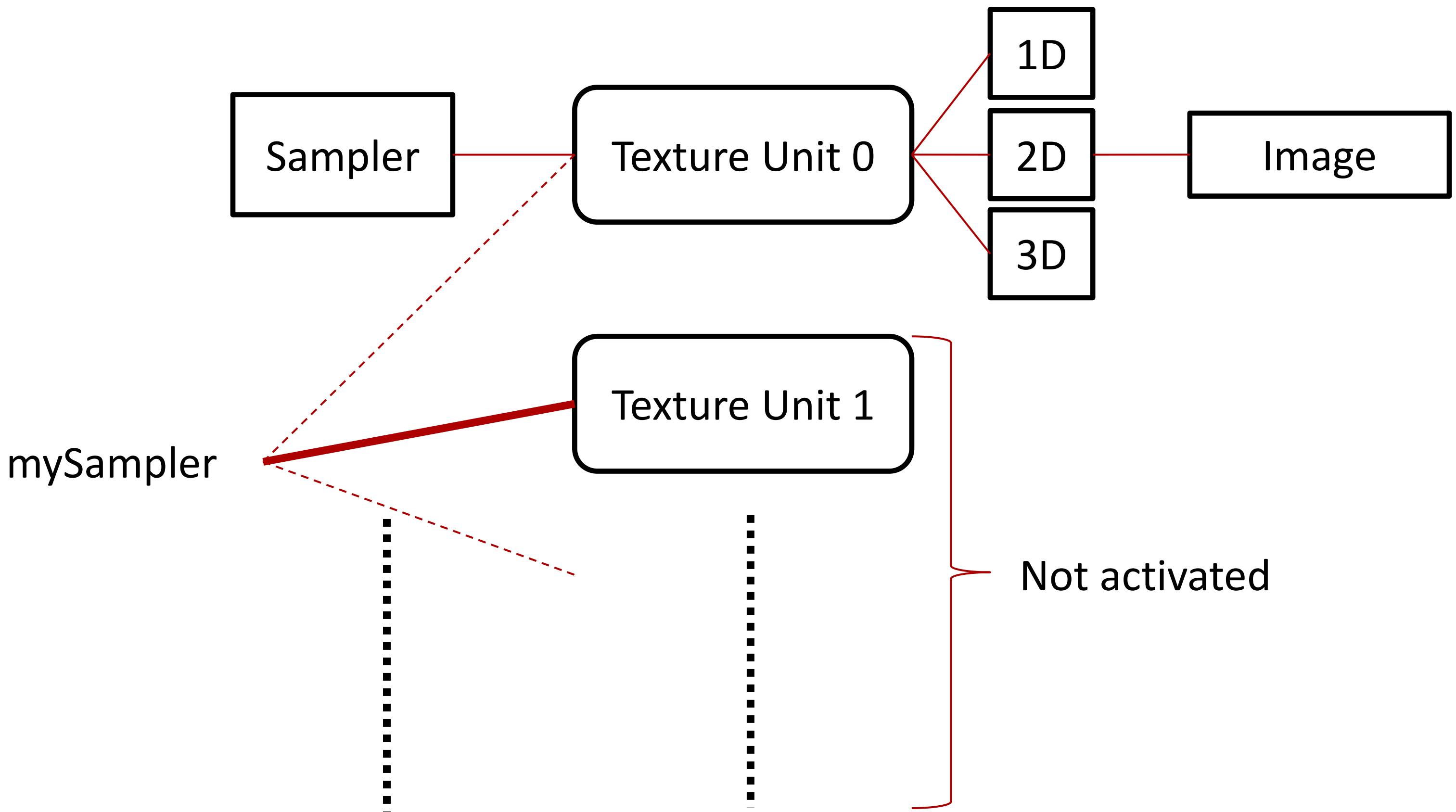
Texture Mapping using OpenGL

- At this point we have the following picture:



Texture Mapping using OpenGL

- If we call *glUniform1f(mySamplerLoc, 1)*:



Texture Mapping using OpenGL

- What to do once we have the texture color? We have several options
- For instance to blend the texture color with the color of the fragment:

```
void main(void)
{
    // get the color from the texture
    gl_FragColor = alpha * gl_Color +
        (1 - alpha) * texture2D(mySampler, gl_TexCoord[0].st);
}
```

Interpolated color value

User-defined interpolation parameter. Can be a *uniform*.

Sampling

- Sampling is the process of fetching the value from a *texture image* given its *texture coordinate*
- Nearest-neighbor and bilinear interpolation are two examples
- Need to tell OpenGL about the type of sampling we want
- Previously we set sampling parameters using:

```
glSamplerParameteri(mySampler, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glSamplerParameteri(mySampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

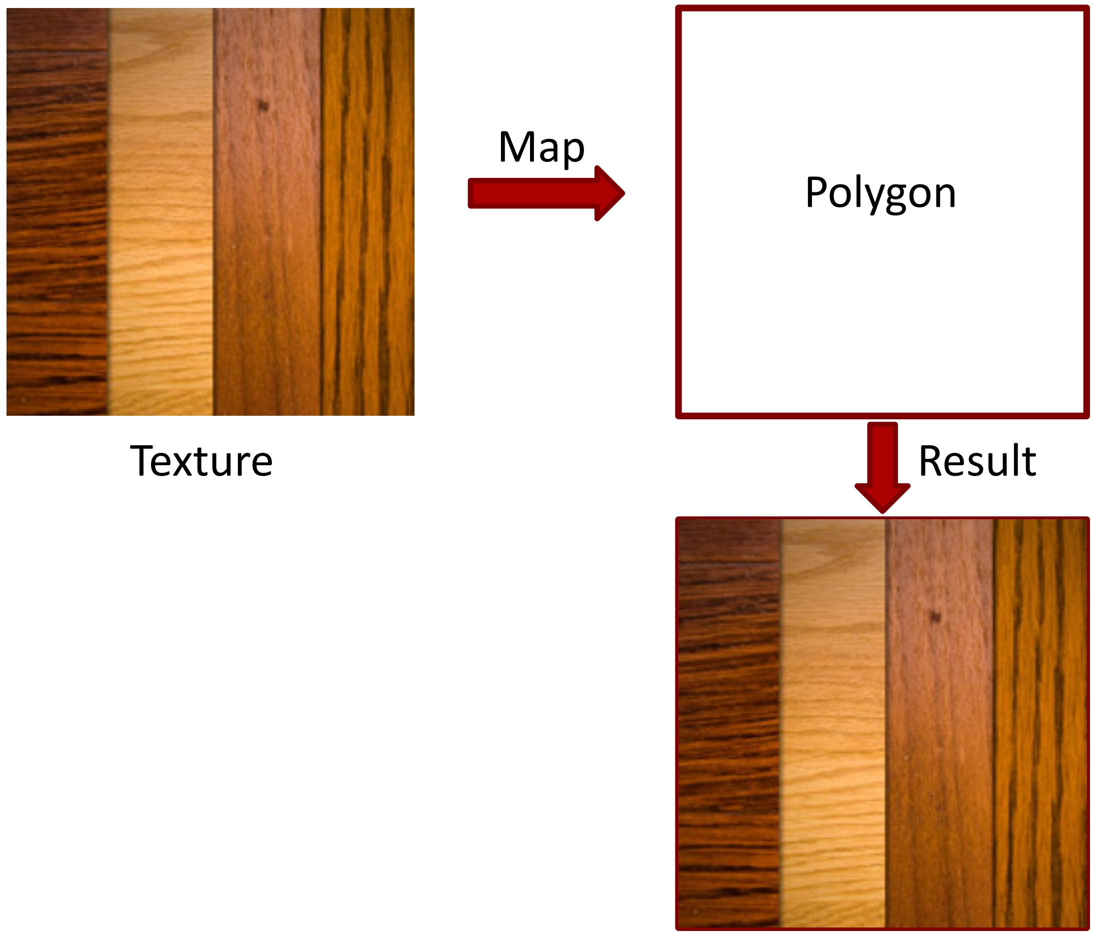
- There is another important concept called mipmapping

Mipmapping

- Mipmapping deals with cases when the resolution of the primitive is different from the resolution of the texture (which often is the case)
- Consider three cases where
  - The polygon that is texture mapped is the same size (in screen space) as the texture image
  - The polygon that is texture mapped is larger than the texture image
  - The polygon that is texture mapped is smaller than the texture image

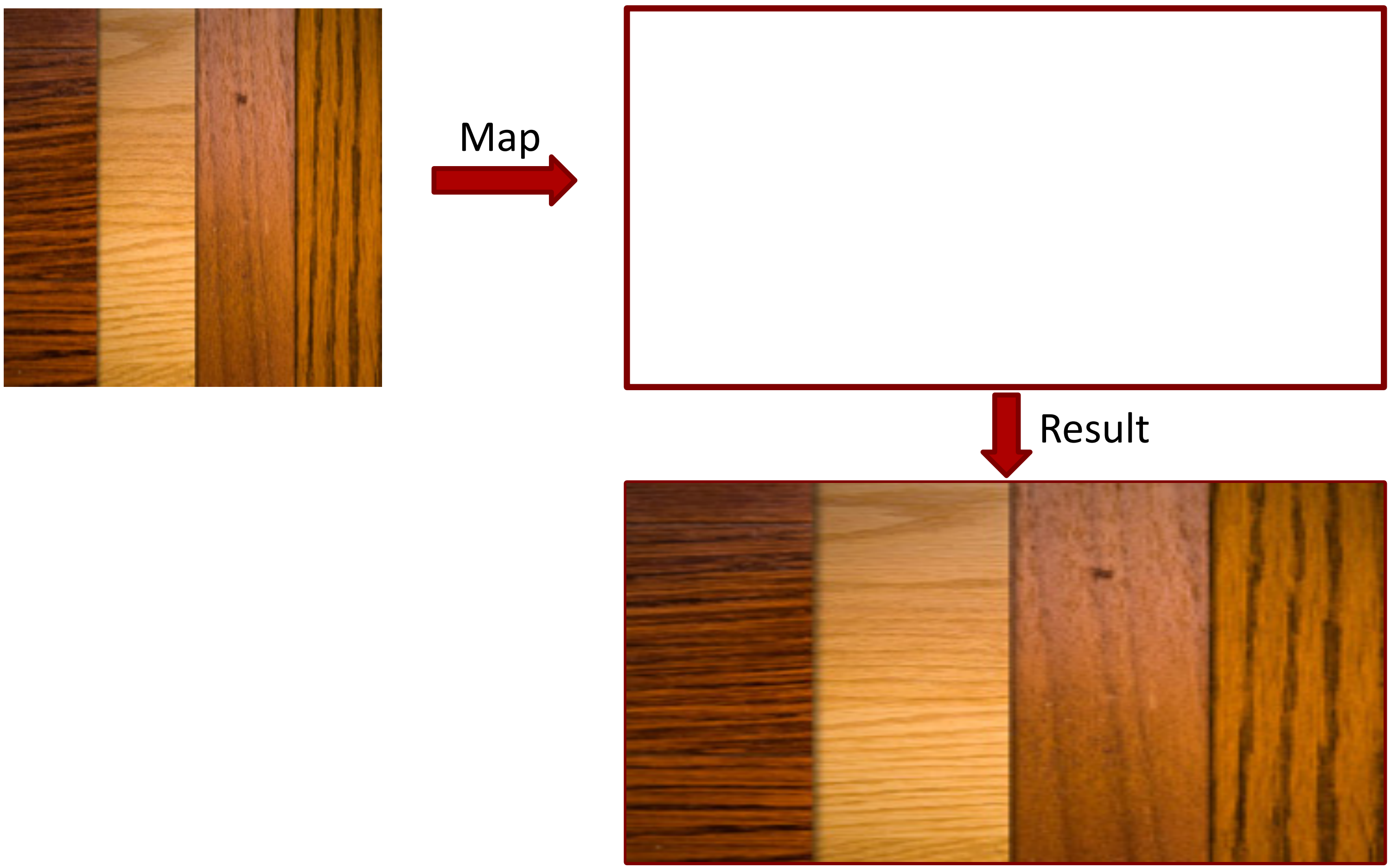
Mipmapping

- Polygon same size as texture (map as usual):



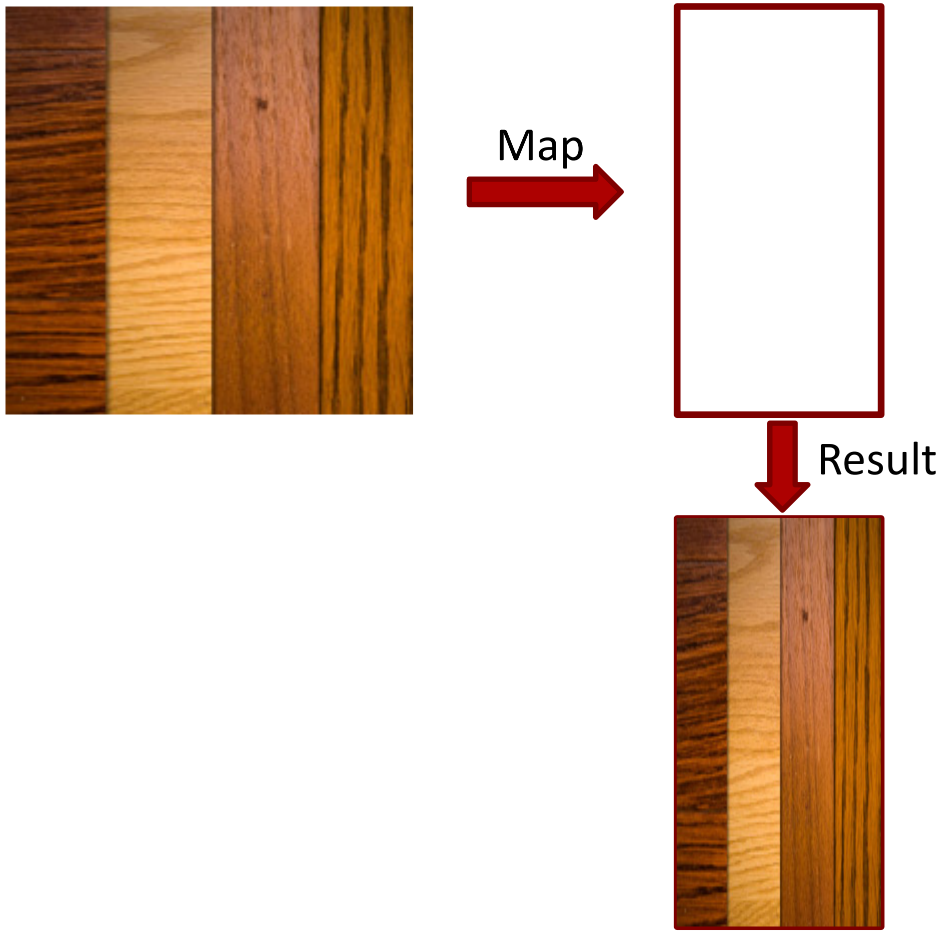
Mipmapping

- Polygon larger (texture needs to be magnified):



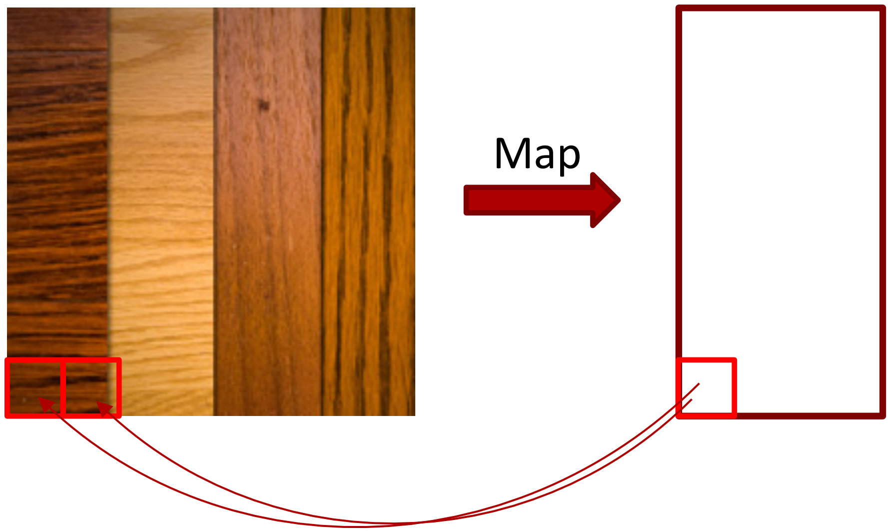
Mipmapping

- Polygon smaller (texture needs to be minified):



Mipmapping

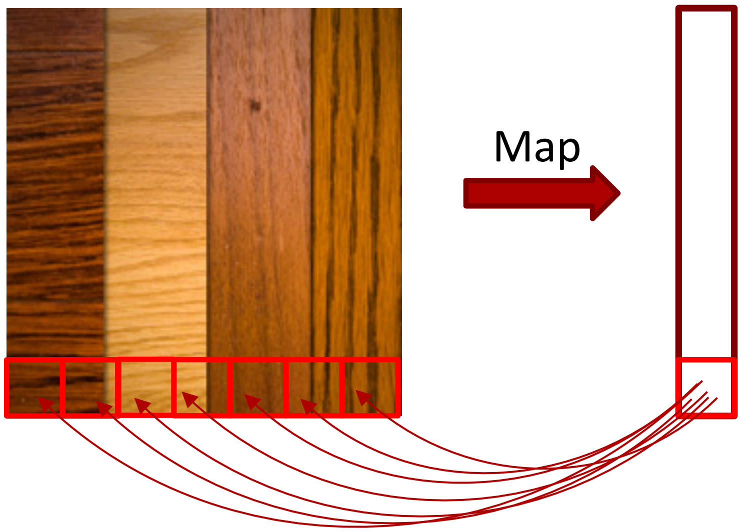
- **Minification:** A change of 1 pixel in image space causes a change of >1pixel in texture space



- To avoid artifacts, one should use the average of all texels that should fall on the same image pixel

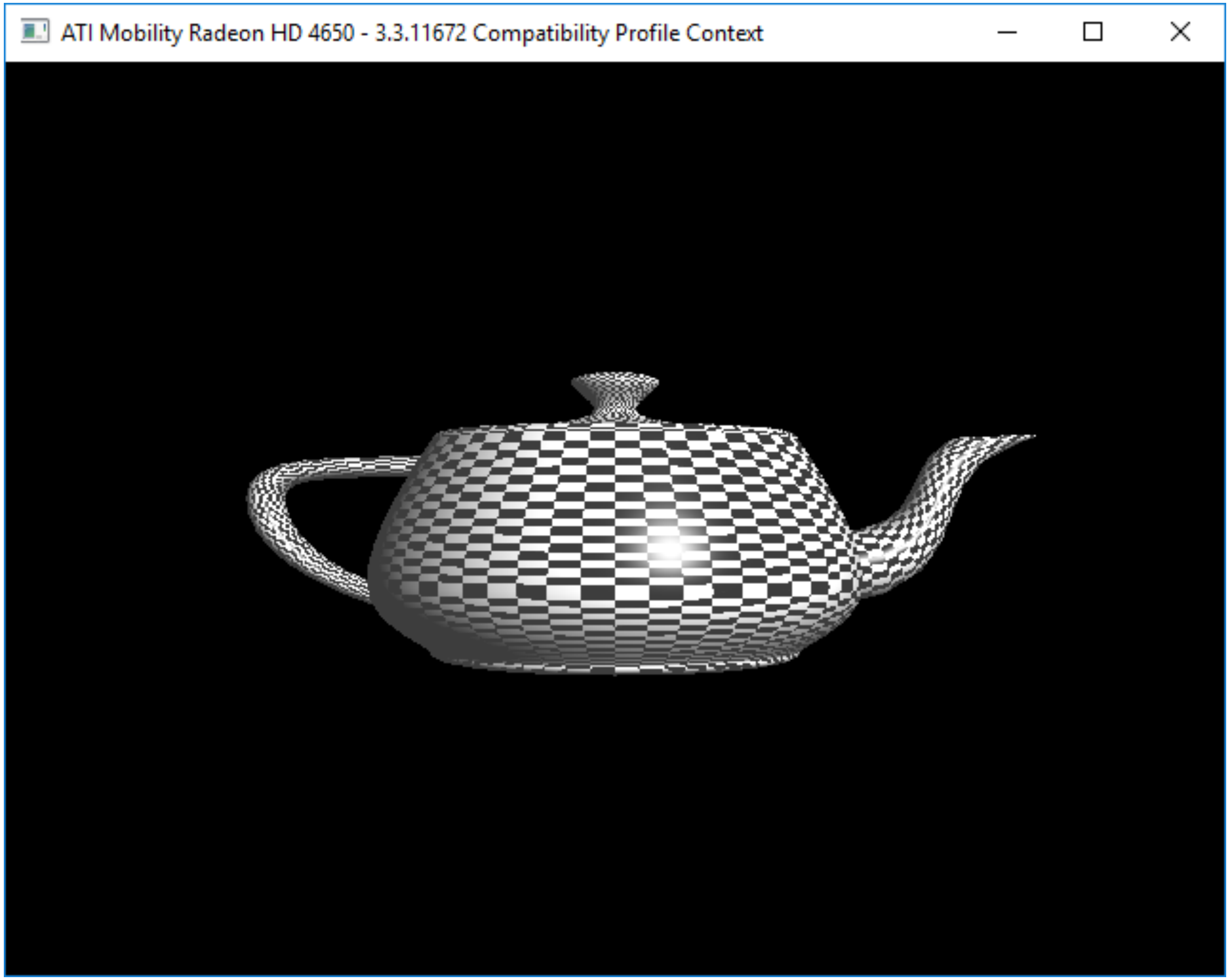
Mipmapping

- **Take the extreme case:** 1 pixel change in image space corresponds to as many pixels as the width of the texture in texture space:

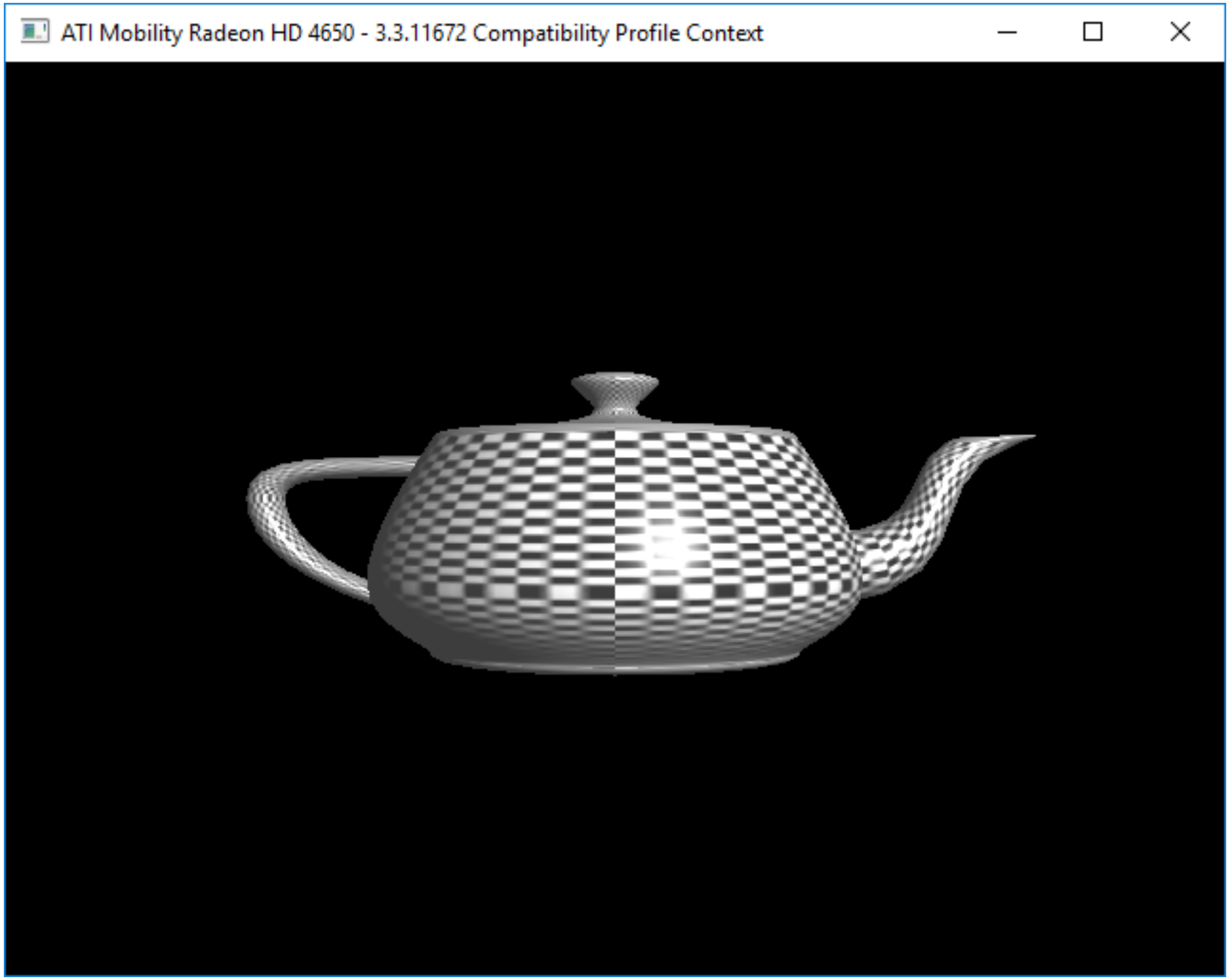


- For accurate mapping, this requires computing the average value of the entire row – otherwise aliasing artifacts will occur

Aliasing


ATI Mobility Radeon HD 4650 - 3.3.11672 Compatibility Profile Context

Anti-Aliased Result

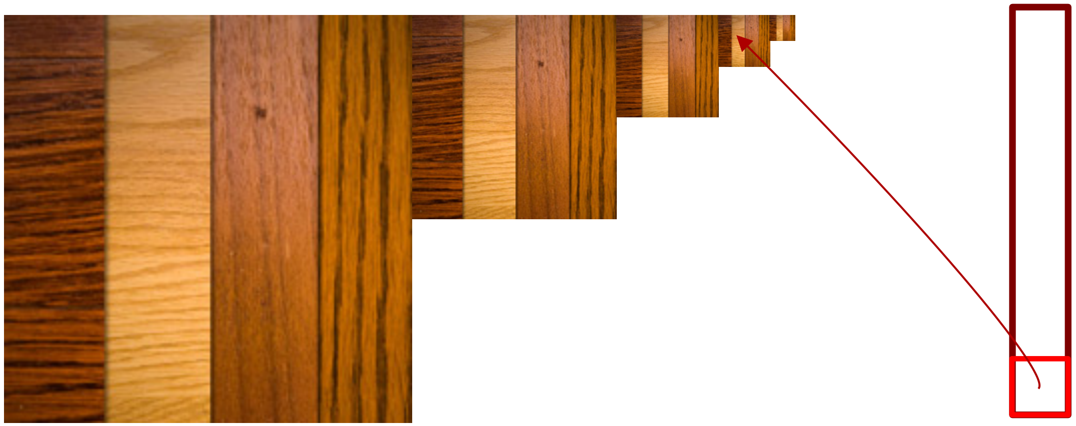


Fixing Aliasing

- Aliasing artifacts are even more disturbing if animation is present in the scene
- Aliasing artifacts occur as we are sampling a high frequency texture at very low frequencies
- Our sample does not faithfully represent the real signal
  - It adopts a different persona – thus called aliasing
- Sampling at a higher rate is not an option as samples are determined by our fragments
- **Solution:** Reduce the frequency of the original signal by low-pass filtering (blurring)
- **Problem:** Expensive to continuously filter in runtime

Mipmapping

- **Solution:** Pre-filter images to create smaller resolution versions during initialization (or offline):



- Then sample from the appropriate resolution in runtime
- Memory requirement – how much memory does a mipmap chain require?

$$A + A/4 + A/16 + A/64 + \ldots = 4A/3$$

OpenGL Support

- Mipmap levels can be created offline and then given to OpenGL. This allows custom filtering for each level:

```
for (int level = 0; level < numLevels; ++level)
{
    glTexImage2D(GL_TEXTURE_2D, level, GL_RGB,
                 width, height, 0,
                 GL_RGB, GL_UNSIGNED_BYTE, image[level]);
}
```

OpenGL Support

- Alternatively, we can ask OpenGL to automatically generate mipmap levels for us:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             width, height, 0,
             GL_RGB, GL_UNSIGNED_BYTE, image1D);

glGenerateMipmap(GL_TEXTURE_2D);
```

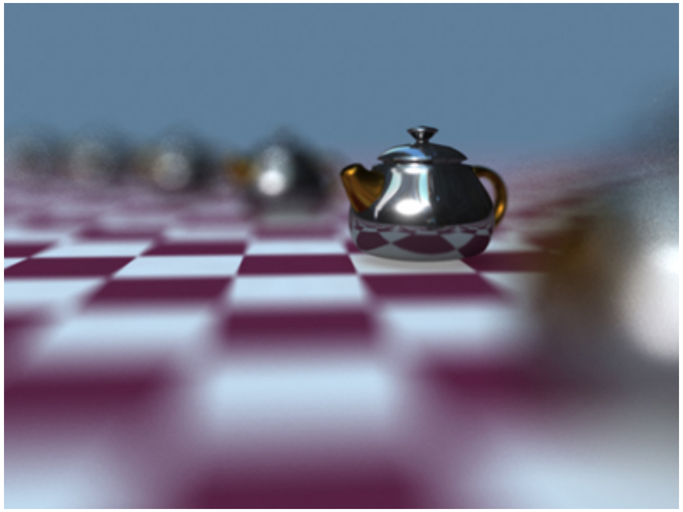
- To use mipmapping, we must set the sampler parameters correctly:

```
//glSamplerParameteri(mySampler, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glSamplerParameteri(mySampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glSamplerParameteri(mySampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri(mySampler, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Framebuffer Objects

- Until now, we always rendered to the screen
- But many visual effects require rendering an image to an off-screen buffer and processing it before displaying it

Motion Blur



Depth of Field

Framebuffer Objects

- Framebuffer objects are designed to allow such effects
- **Step 1:** To use an FBO you must first generate a name for it and bind it as the current framebuffer

```
GLuint gFBOId;
glGenFramebuffers(1, &gFBOId);
glBindFramebuffer(GL_FRAMEBUFFER, gFBOId);
```

Framebuffer Objects

- **Step 2:** Next we must allocate memory for its color and (optionally) depth buffers. These memories are allocated as textures
- For color buffer:

```
glGenTextures(1, &gColorTextureId);
glBindTexture(GL_TEXTURE_2D, gColorTextureId);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, gFBOWidth, gFBOHeight, 0,
             GL_RGB, GL_UNSIGNED_BYTE, 0);
```

Framebuffer Objects

- **Step 2:** Next we must allocate memory for its color and (optionally) depth buffers. These memories are allocated as textures
- For depth buffer:

```
glGenTextures(1, &gDepthTextureId);
glBindTexture(GL_TEXTURE_2D, gDepthTextureId);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, gFBOWidth, gFBOHeight, 0,
             GL_DEPTH_COMPONENT, GL_FLOAT, 0);
```

Framebuffer Objects

- **Step 3:** We must attach these textures to the FBO:

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
                       GL_TEXTURE_2D, gColorTextureId, 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                       GL_TEXTURE_2D, gDepthTextureId, 0);
```

- **Step 4:** Make sure that FBO is complete:

```
GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
assert(status == GL_FRAMEBUFFER_COMPLETE);
```

Framebuffer Objects

- When we render while this FBO is bound, the attached textures’ contents will be updated
- **Important:** before rendering make sure that you set your viewport to match the resolution of this framebuffer using *glViewport(0, 0, gFBOWidth, gFBOHeight)*
- This is needed as the size of the window (for which the viewport was originally set) can be different from the size of our FBO

Framebuffer Objects

- Once you make the FBO rendering pass, you can *detach* your textures and switch back to the default framebuffer:

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
                       GL_TEXTURE_2D, 0, 0);

glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                       GL_TEXTURE_2D, 0, 0);

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

- Now you can use these textures as source textures for various special effects
- One such usage is for generating shadows as we will learn next week