Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

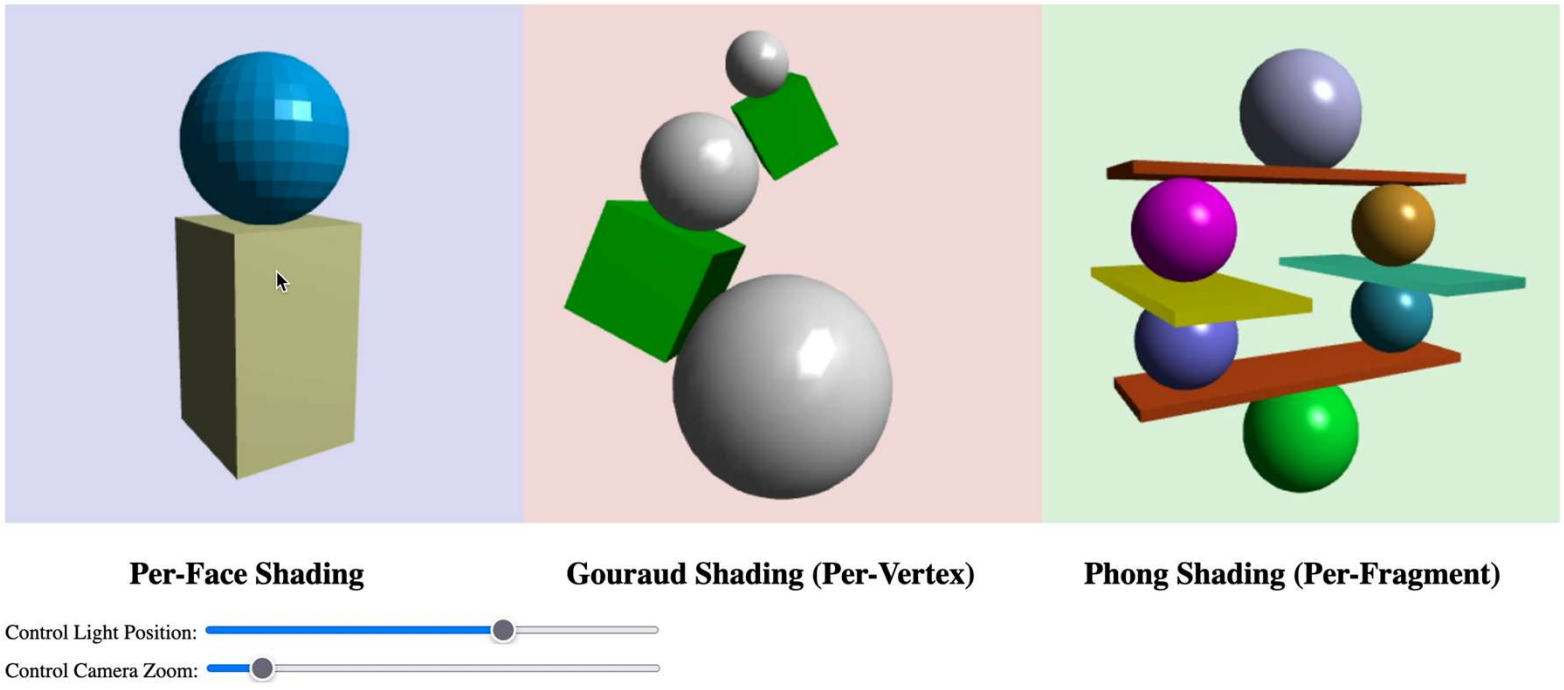
email: soumyad@cse.iitk.ac.in

Acknowledgements

- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis, and Prof. Cem Yuksel

Assignment 2: Due: Sept 10th 11:59pm

- Simple 3D Object rendering with 3 different shading models
  - Flat Shading, Gouraud Shading, Phong Shading
- Handling 3 viewports and allowing exclusive interactions on them
- Using sliders allow light movement and camera zooming



Switching Between Shaders

- function webGLStart() {

…..

```
flatShaderProgram = initShaders(flatVertexShaderCode,
                                flatFragShaderCode);
perVertshaderProgram = initShaders(perVertVertexShaderCode,
                                   perVertFragShaderCode);
perFragshaderProgram = initShaders(perFragVertexShaderCode,
                                   perFragFragShaderCode);
```

……

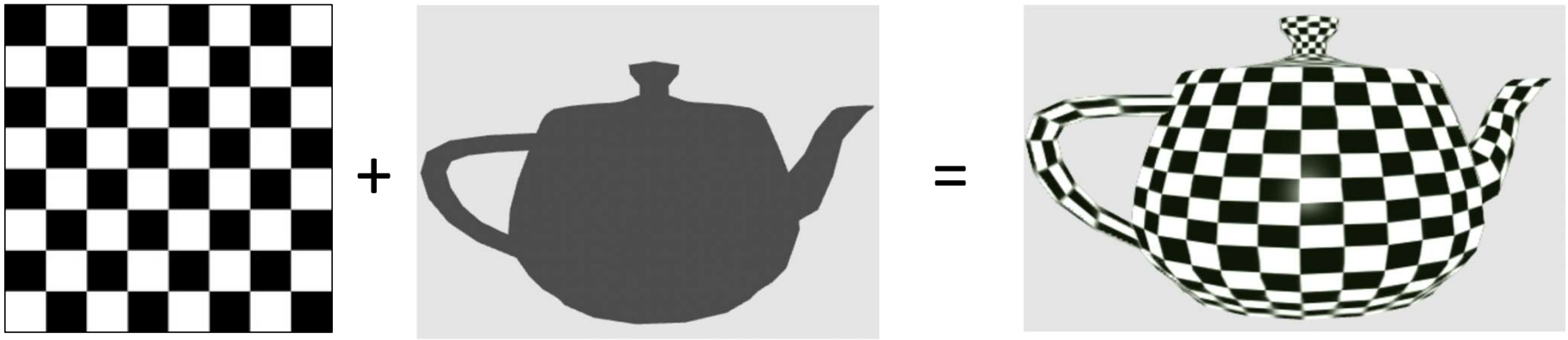
}

Structure of Your drawScene() Function

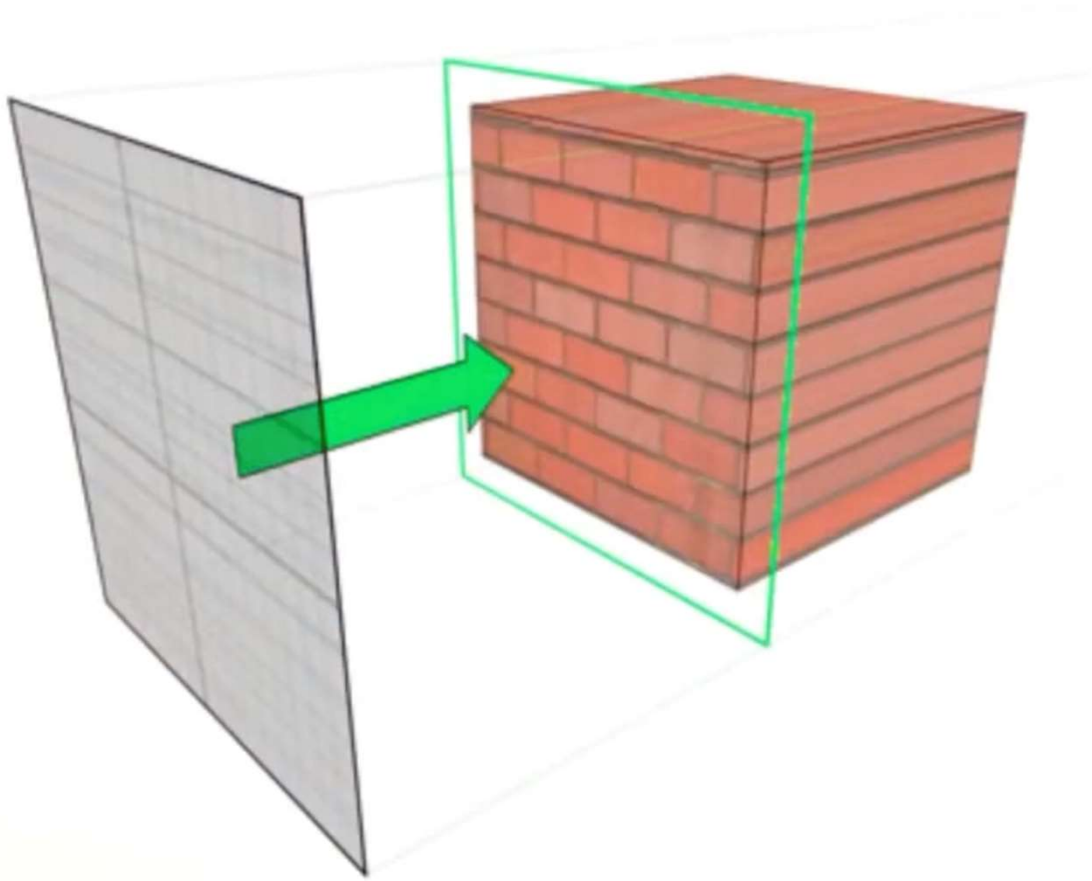
- **Setup viewport 1**
- shaderProgram = flatShaderProgram;
- gl.useProgram(shaderProgram);
- Now setup all shader variables, attributes, enable attributes, and setup uniforms, set up matrices and then draw scene
- **Setup viewport 2**
- shaderProgram = perVertshaderProgram;
- gl.useProgram(shaderProgram);
- Now setup all shader variables, attributes, enable attributes, and setup uniforms, set up matrices and then draw scene
- **Setup viewport 3**
- shaderProgram = perFragshaderProgram;
- gl.useProgram(shaderProgram);
- Now setup all shader variables, attributes, enable attributes, and setup uniforms, set up matrices and then draw scene

Textures

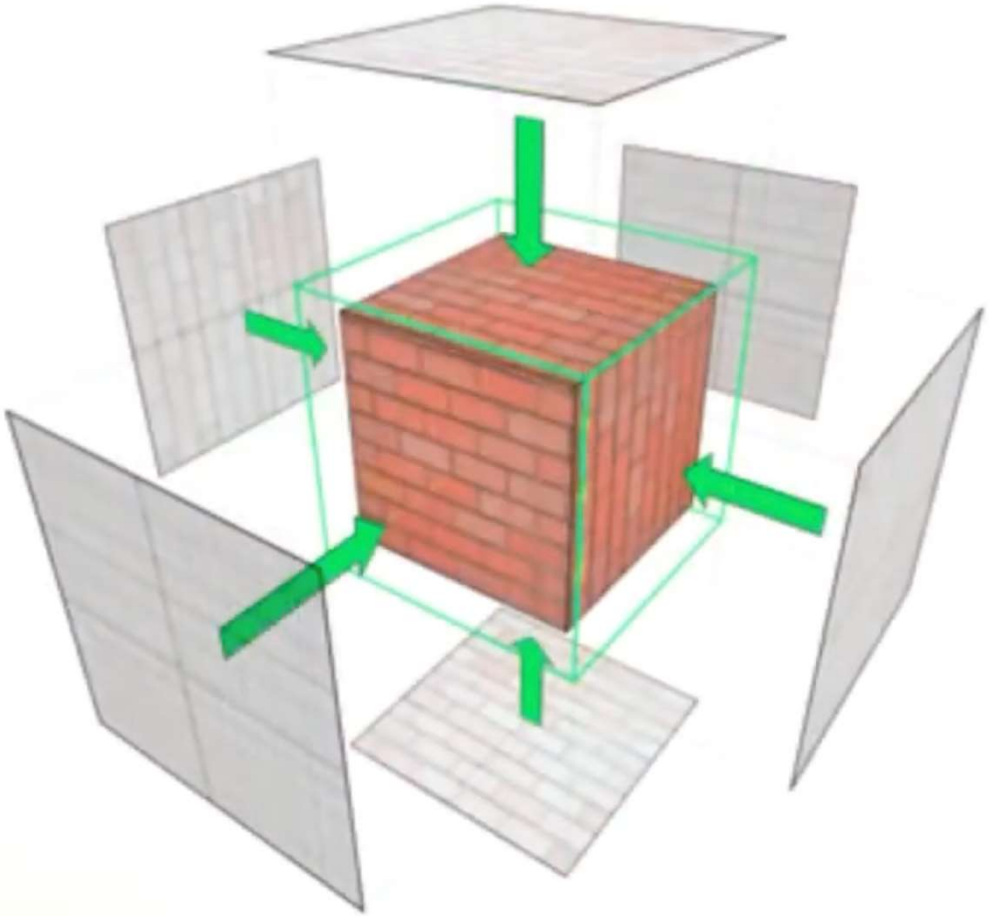
- In a broad level, textures are a means by which we can define information on the surface of an object
- In computer graphics, we typically indicate a texture as an image
- Textures can be generated by mathematical functions as well
  - Procedural texture



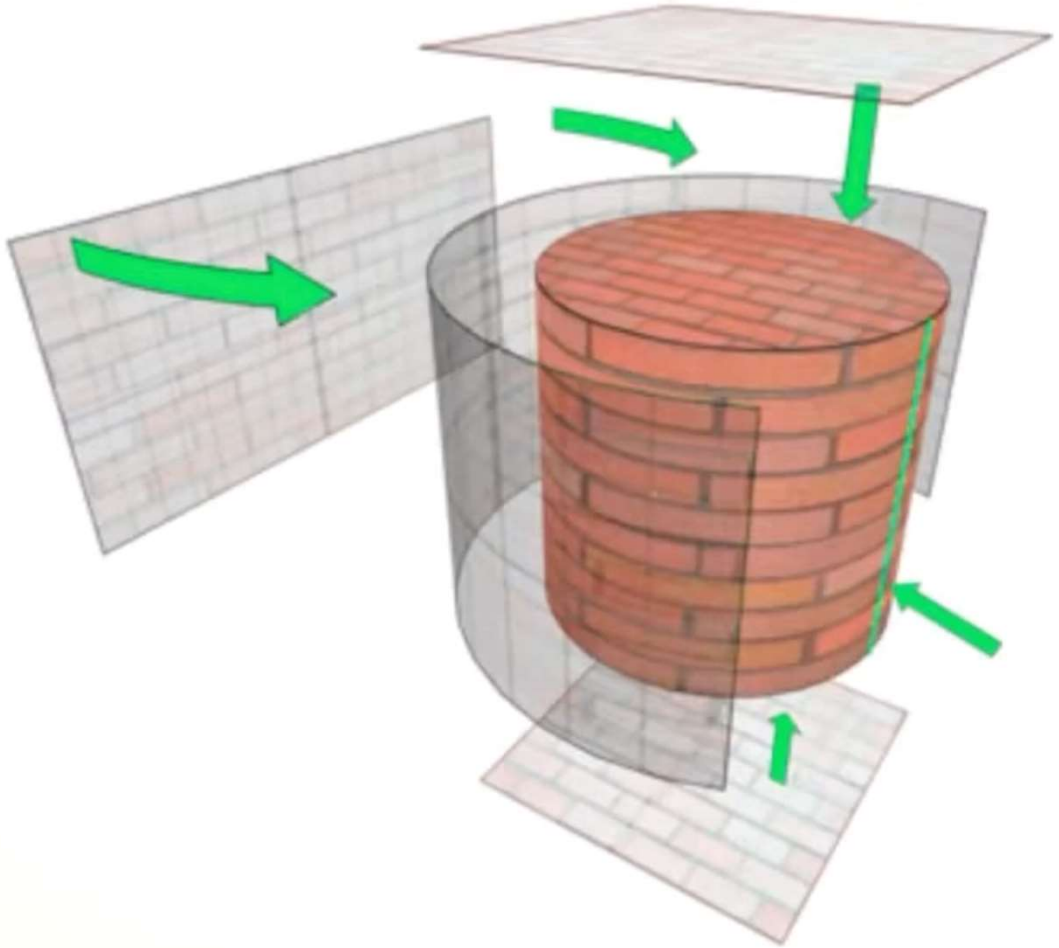
Texture Mapping



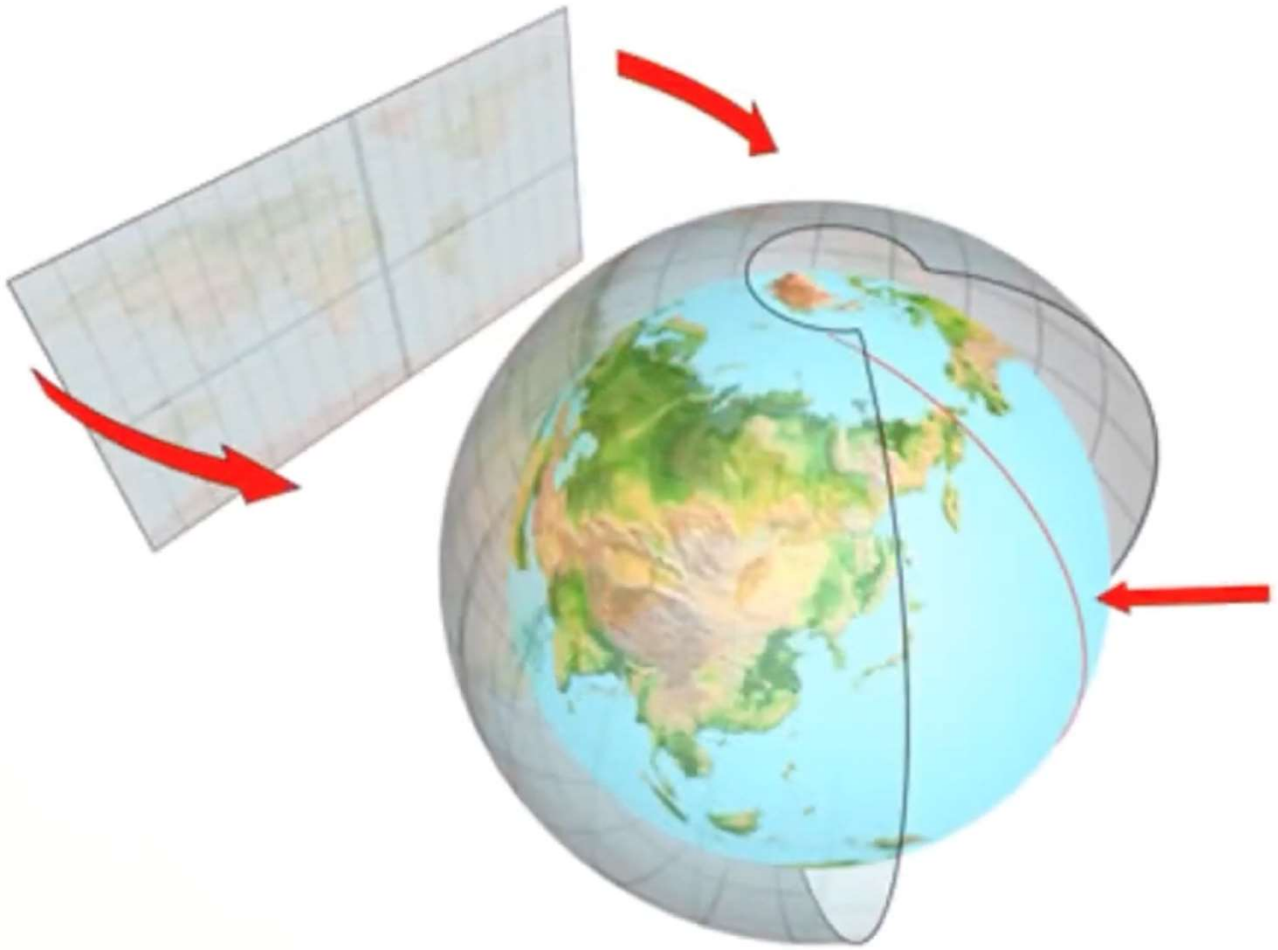
Texture Mapping



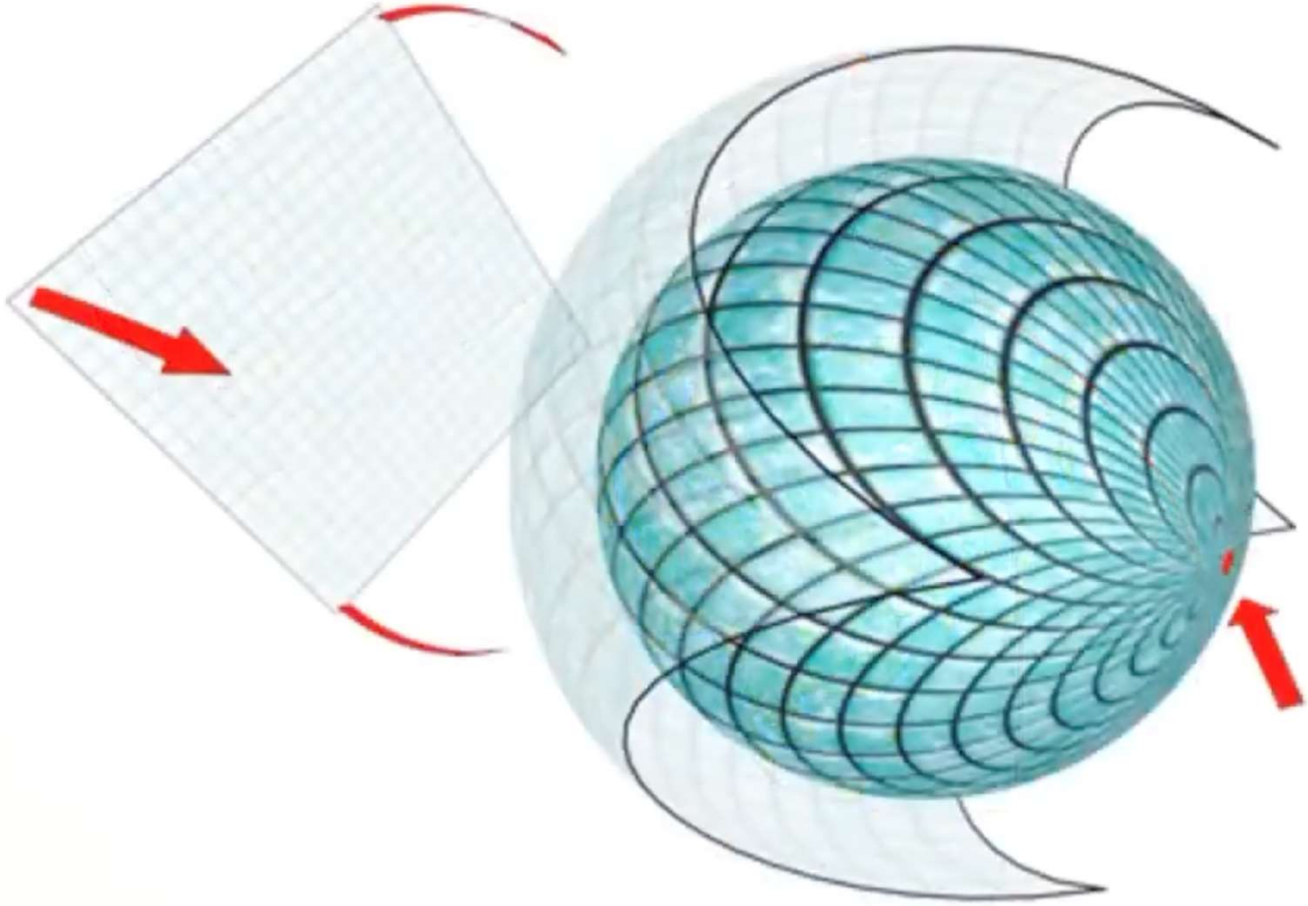
Texture Mapping



Texture Mapping

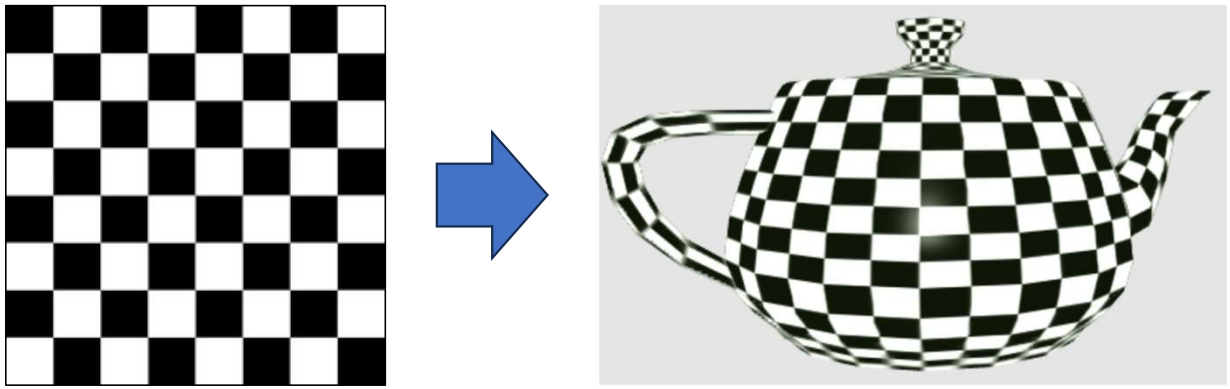


Texture Mapping



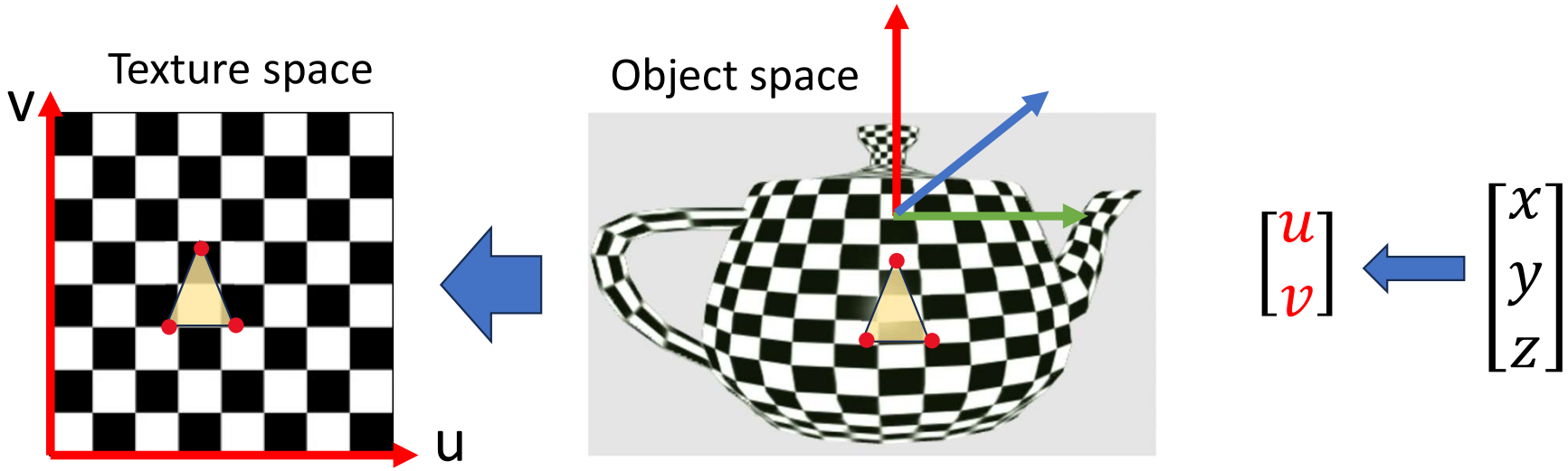
Texture Mapping

- A mapping from **Texture space** to **Object space**
- Texture mapping is sometimes hard
- There are many variants to do this correctly
- Even then, texture mapping on complex shaped objects is not straightforward



Texture Mapping

- A mapping from **Texture space** to **Object space**
- Texture mapping is sometimes hard
- There are many variants to do this correctly
- Even then, texture mapping on complex shaped objects is not straightforward

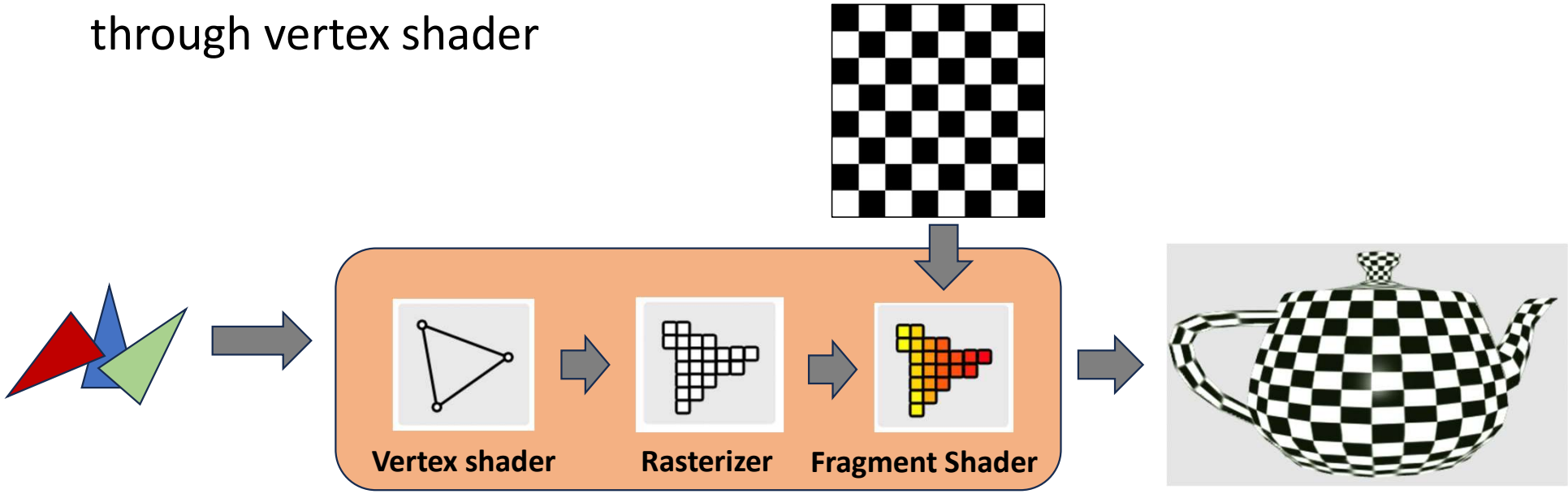


Texture Mapping

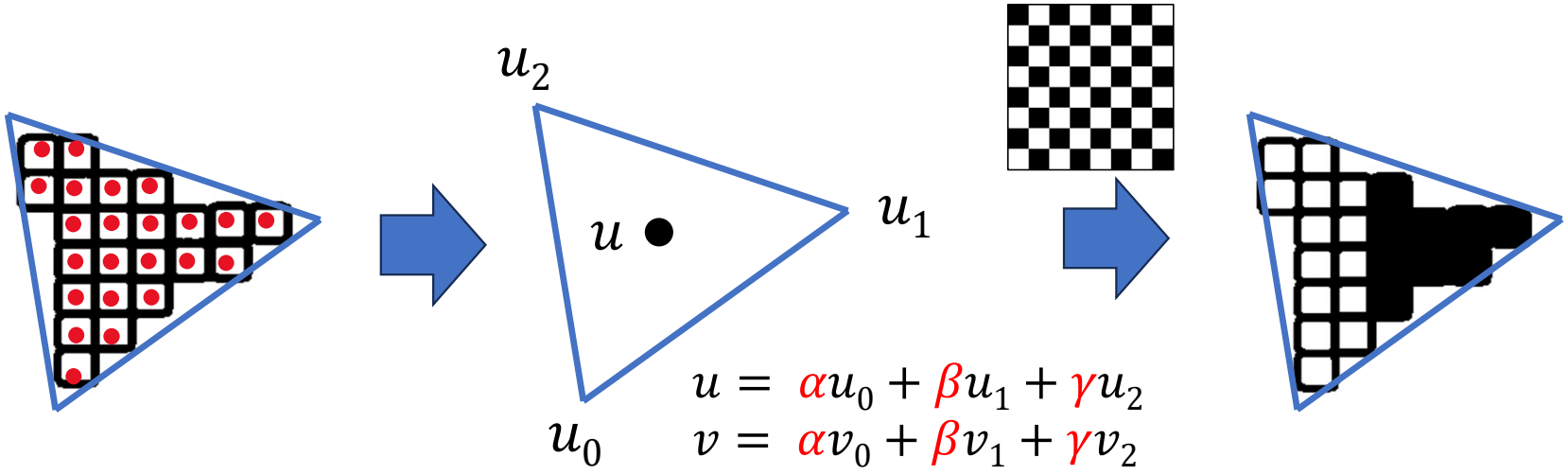
- Great when this mapping is given to us
- We can just do texture color look up from the texture image for each triangle and generate the texture mapped object
- How do we find such mappings?
  - Some automatic/semi-automatic techniques exist
  - Sometime challenging when the object is of complex shape

Texture Mapping: GPU Pipeline

- Texture mapping happens in fragment shader
- Pass vertex texture coordinates through vertex shader



Texture Mapping: GPU Pipeline

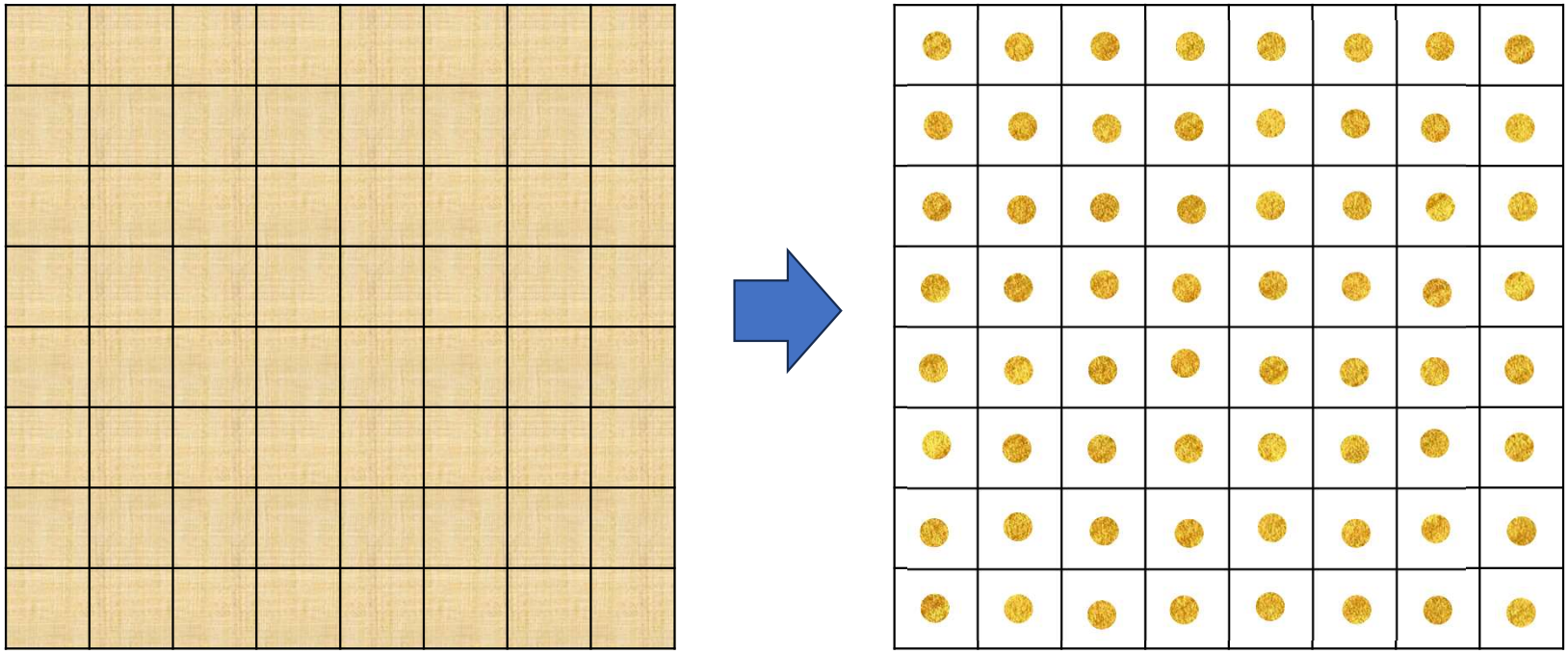


- Use Barycentric coordinates to compute texture coordinates at each fragment inside each triangle
  - This will be done by hardware for us
- Then we look up the color using the (u, v) coordinate from the texture and shade the fragment

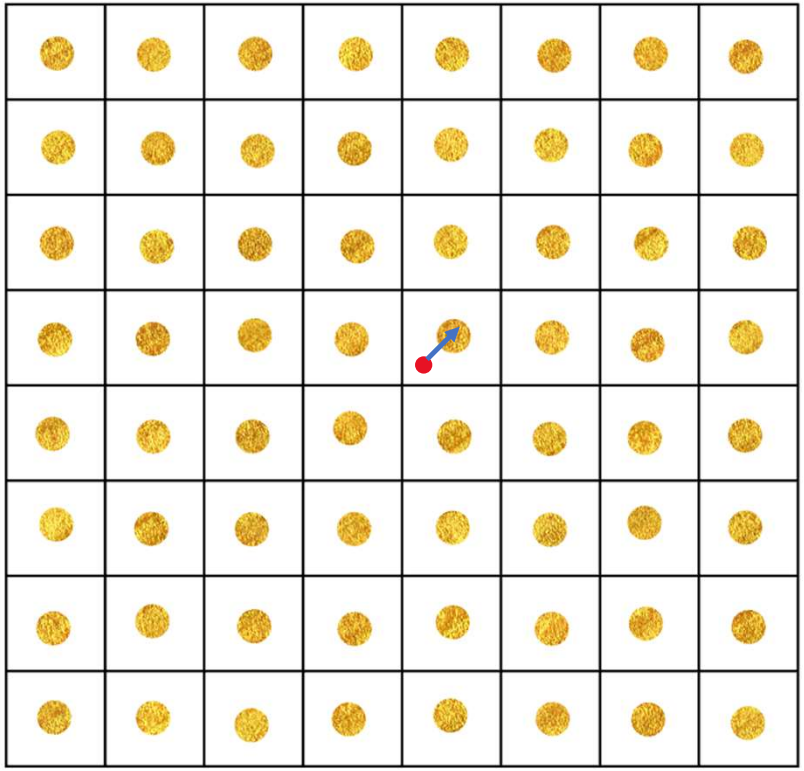
Texture Sampling: Pixel (Texel) Idea

IITK CS360A

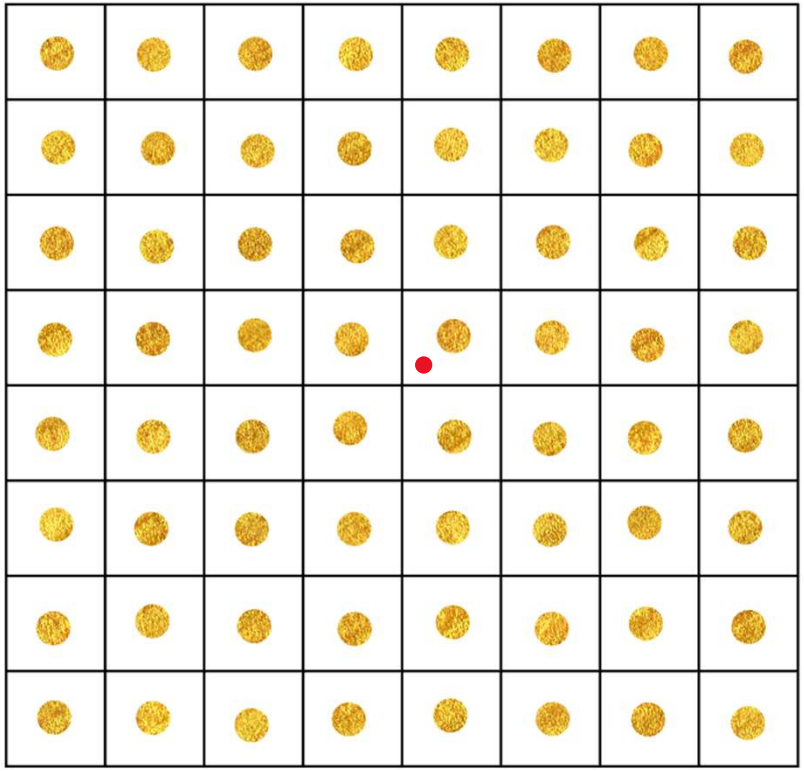
- How do we do this texture color lookup?



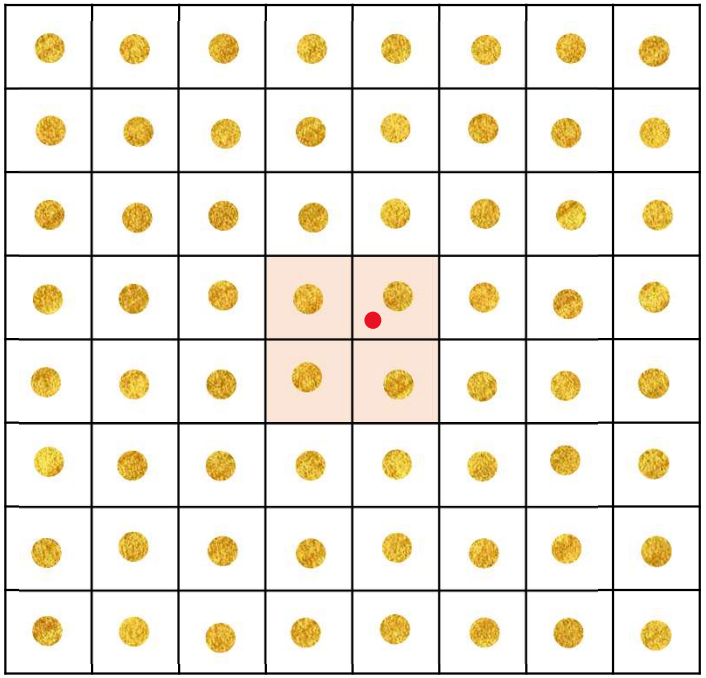
Texture Sampling: Nearest Neighbor



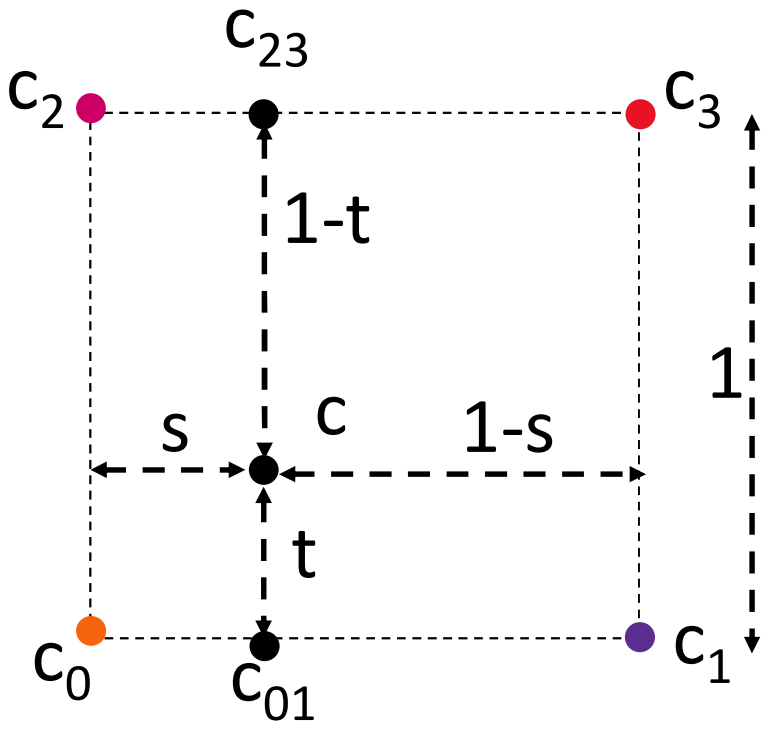
Texture Sampling: Bi-Linear Filtering



Texture Sampling: Bi-Linear Filtering



Bi-Linear Filtering (Interpolation)



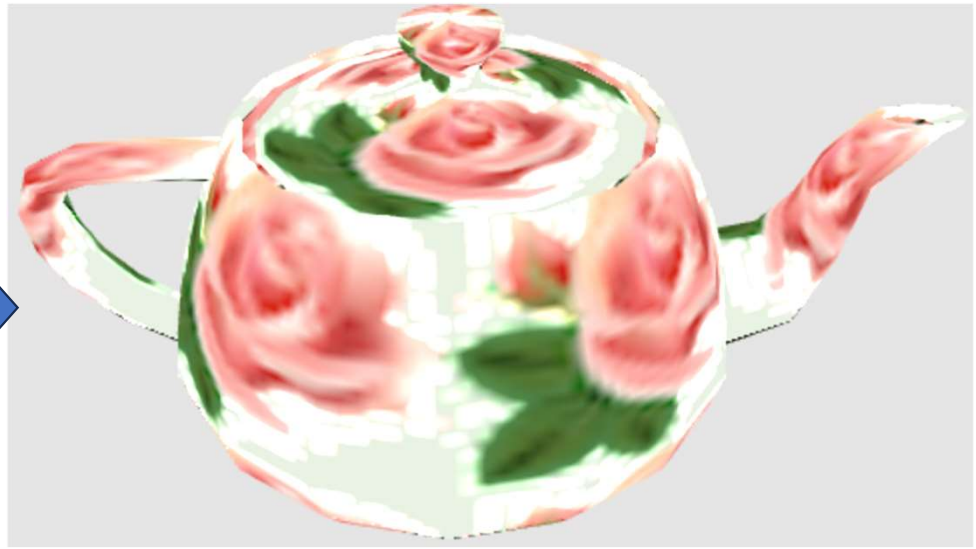
- We first find $\mathbf{c_{01}}$ and $\mathbf{c_{23}}$ using linear interpolation
- Then using another linear interpolation step, we find the value at C

$$C = (1-t)(1-s)c_0 + (1-t)sc_1 + t(1-s)c_2 + tsc_3$$

Nearest vs Bi-Linear Filtering

Nearest vs Bi-Linear Filtering

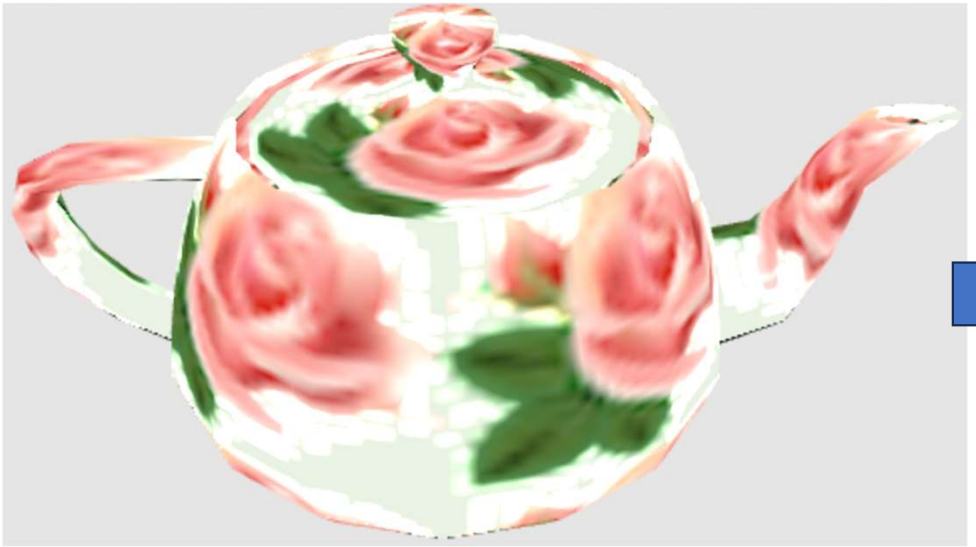
Nearest



Bi-linear

How Can We Improve Quality Further?

- How do we get the following quality?

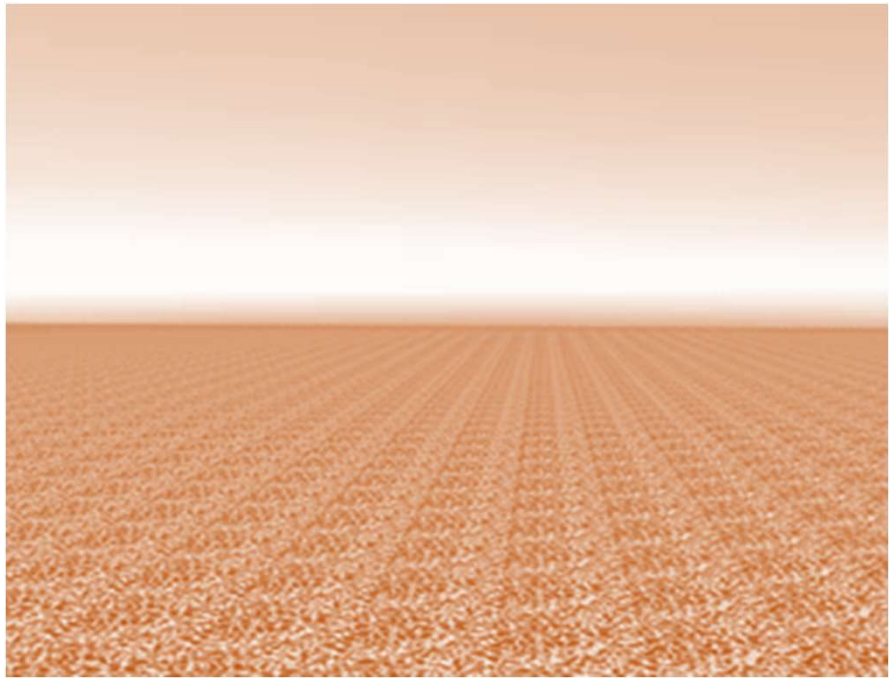


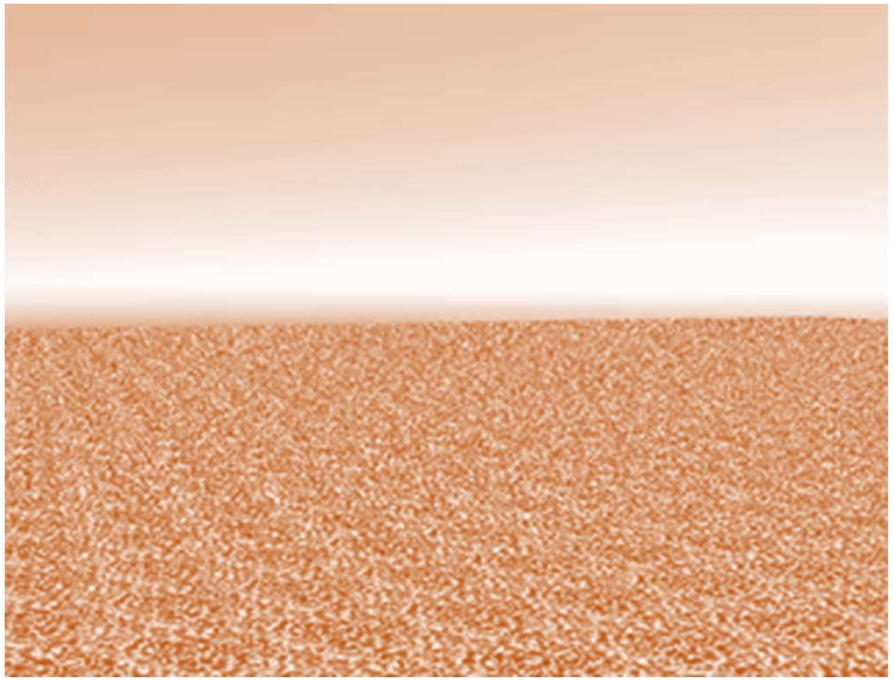
Bi-linear

Use a high–resolution texture map where the ‘texels’ are very small

Texturing Problem Solved?

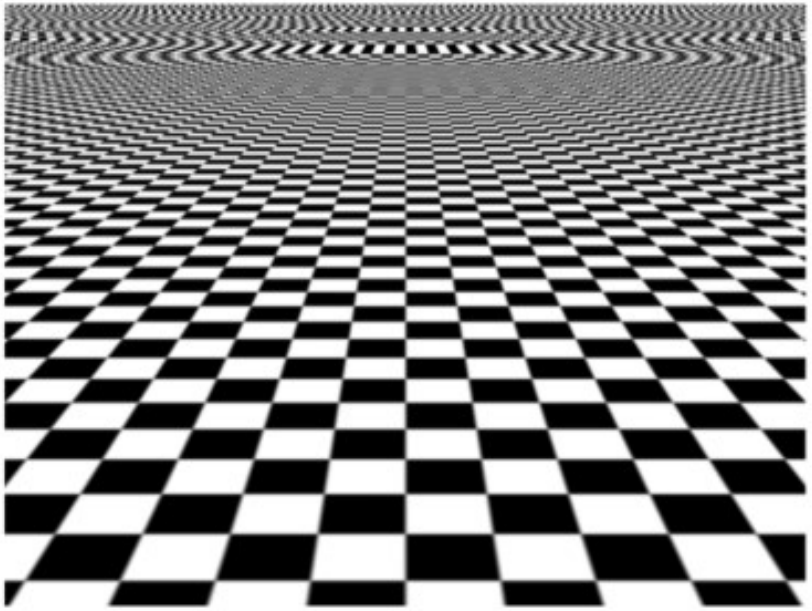
Not yet!





Aliasing!

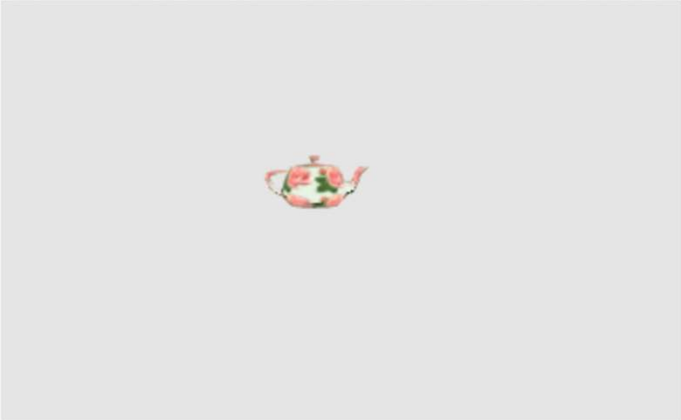
Aliasing



Bi-linear texture filtered image

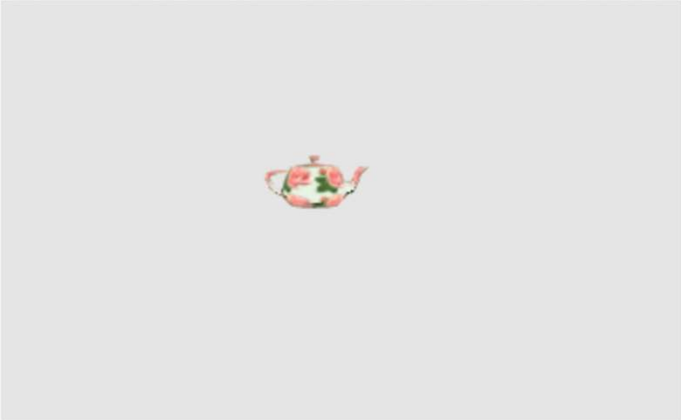
- When a high frequency texture is being used to render something far away from camera, some visible artifacts appear, called moiré patterns
- This problem is also known as ‘Aliasing’
- At a conceptual level, this is a signal processing problem

What is the Problem?



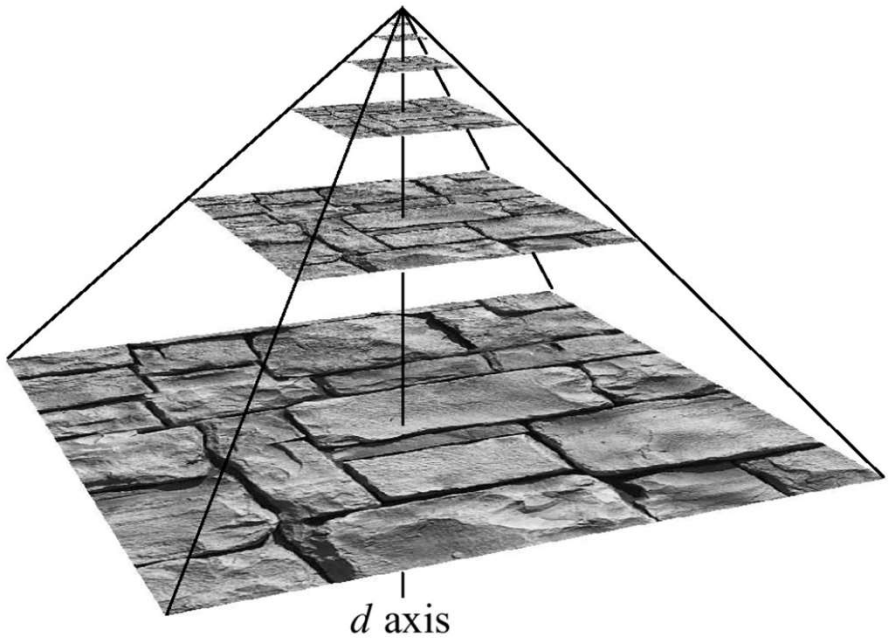
- When the object is near to the camera high-resolution texture makes sense
- When the object is far, we have information overload
- Too many texels gets concentrated in fewer pixels
  - Creates issues with bi-linear texture filtering

How Do We Solve the Problem?



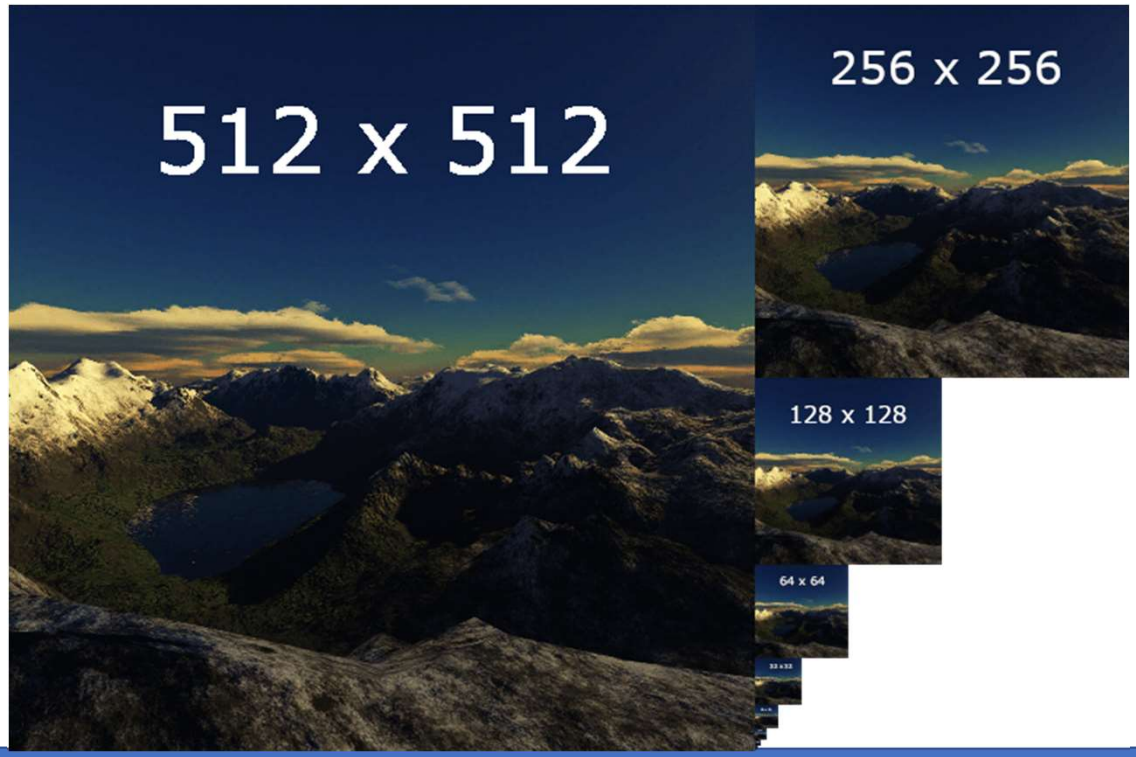
- Perhaps we can store images at multiple resolutions, from largest to smallest and based on how far the objects are from the camera, we will pick a resolution of the image appropriately so that the size of pixels and size of texels are more or less similar!
- This is called '**Mipmap**' technique

Mipmaps

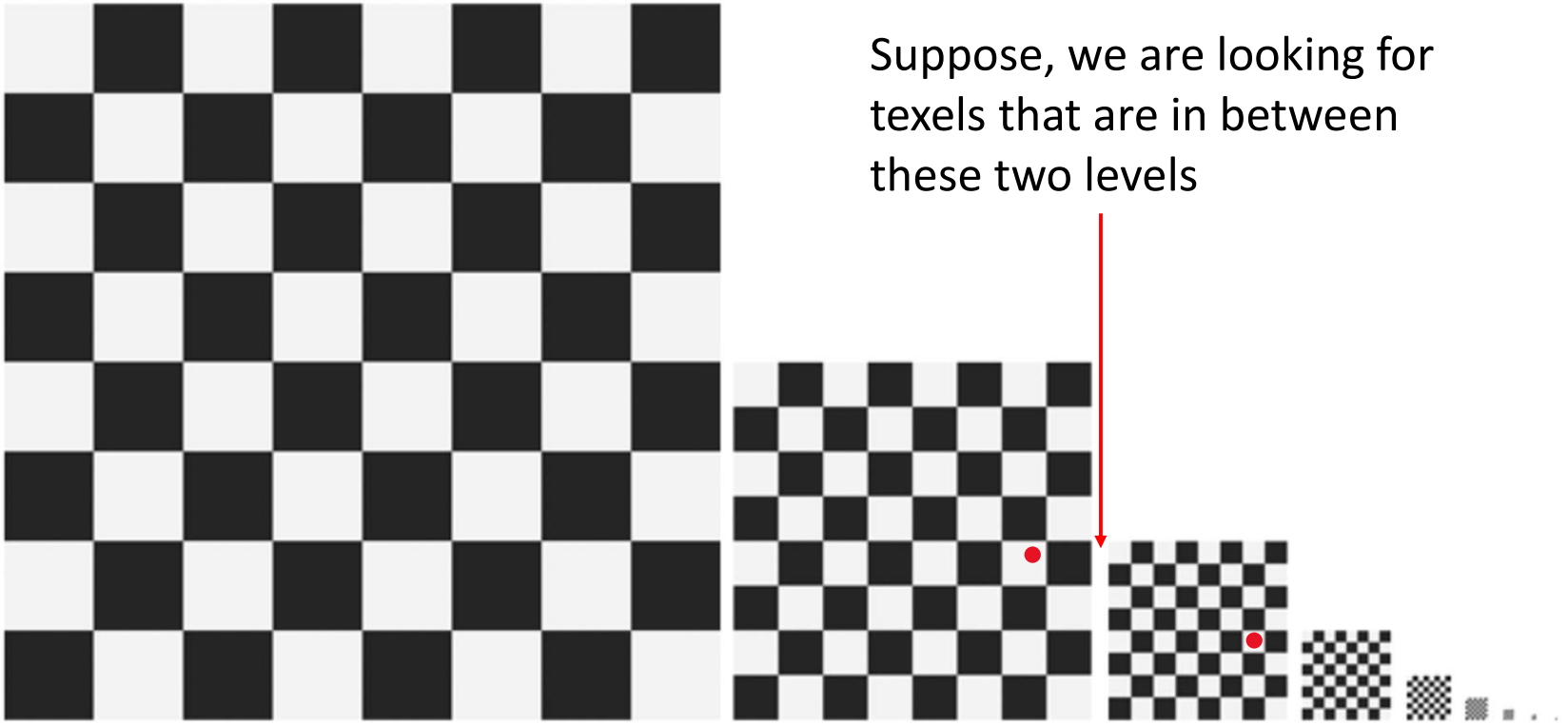


Mipmap levels

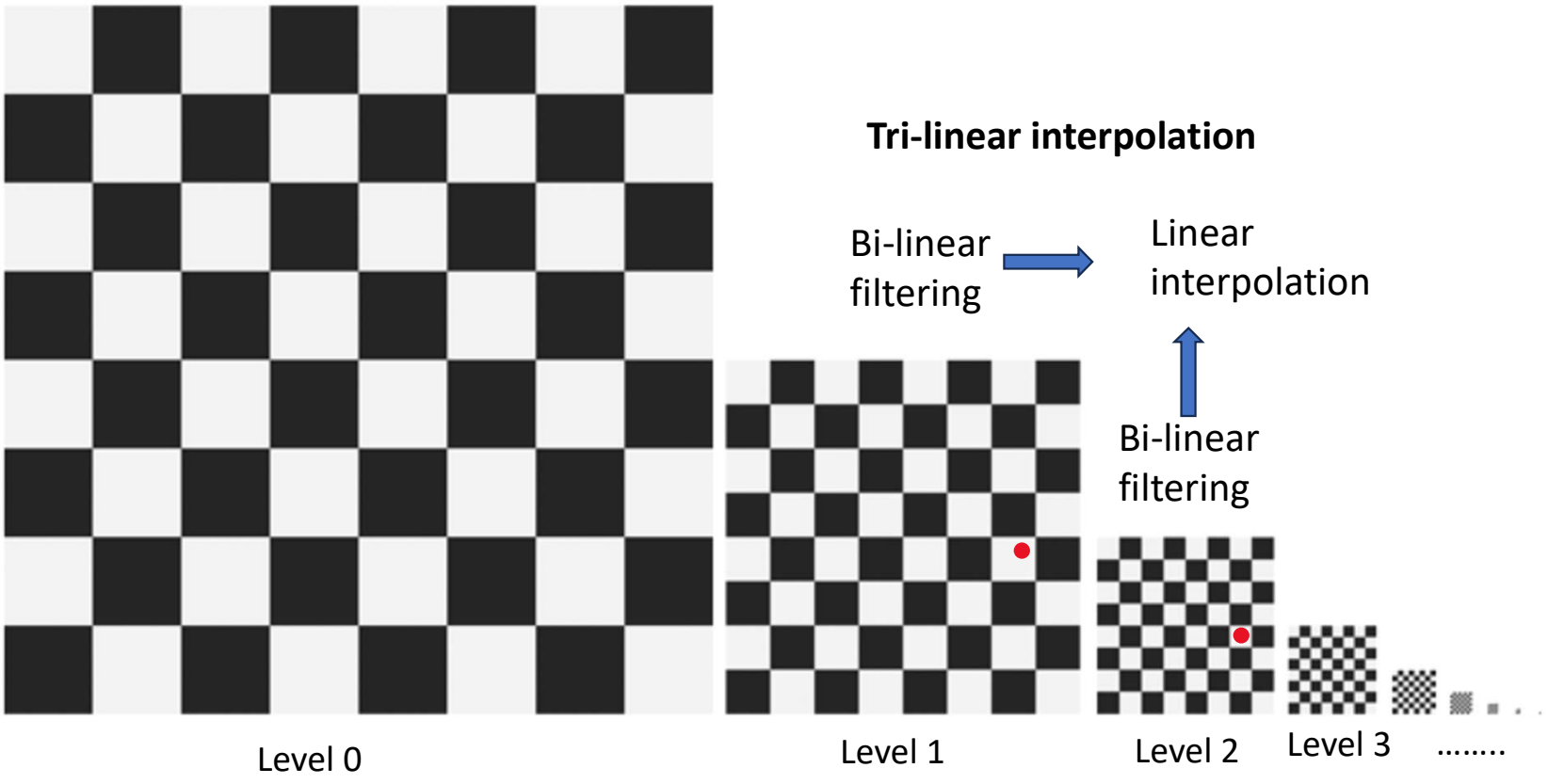
- Pre-generate several texture images at different level of detail



Mipmaps: How Do We Look Up?



Mipmaps: How Do We Look Up?

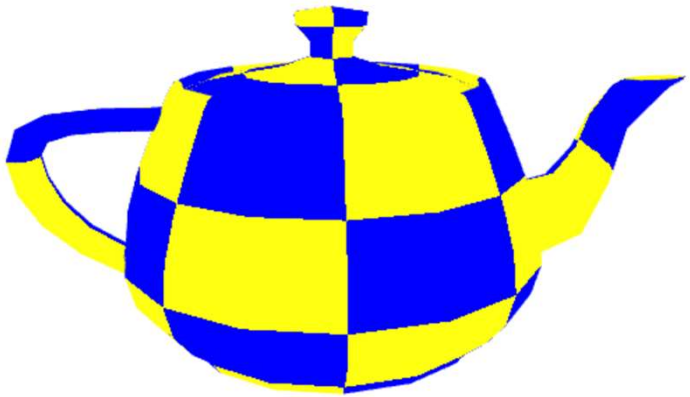


Textures on GPUs

Textures

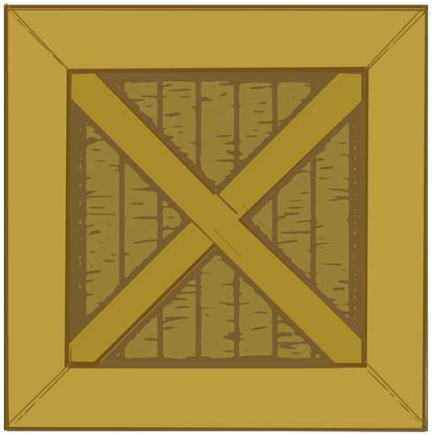
- Textures are generally assumed to be images
- Procedural textures
  - Textures generated using mathematical functions

```
vec3 checkerBoard(vec2 u){
  vec2 f = u - floor(u);
  return (
    (f.x < 0.5) ^^ (f.y < 0.5)) ?
    vec3(1,1,0) :
    vec3(0,0,1);
}
```



Textures in GPU

1D Texture

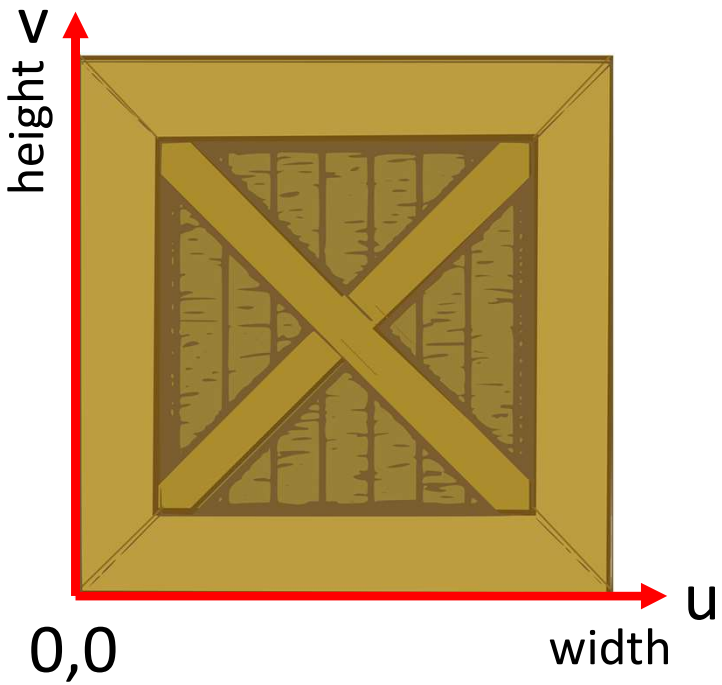


2D Texture

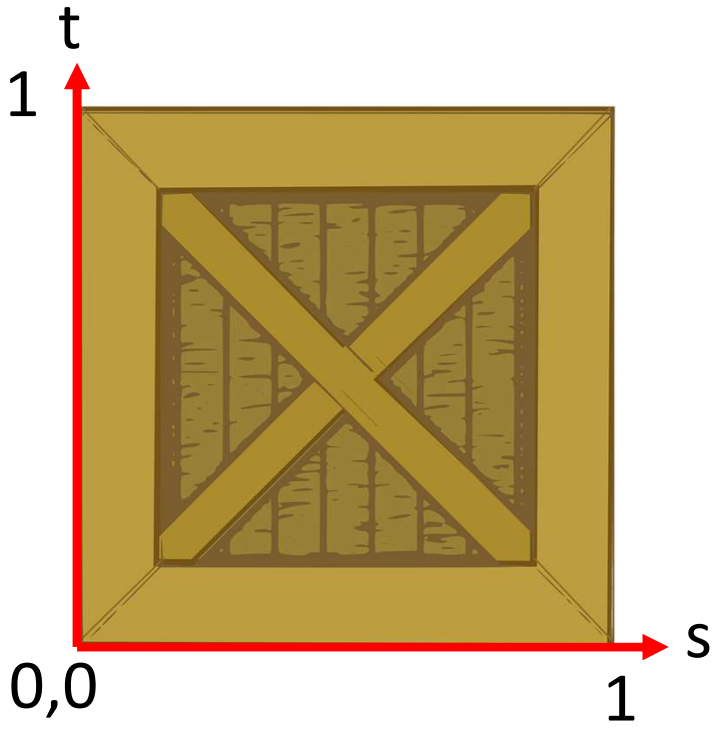
3D Texture

We will primarily use 2D textures

Texture Coordinates



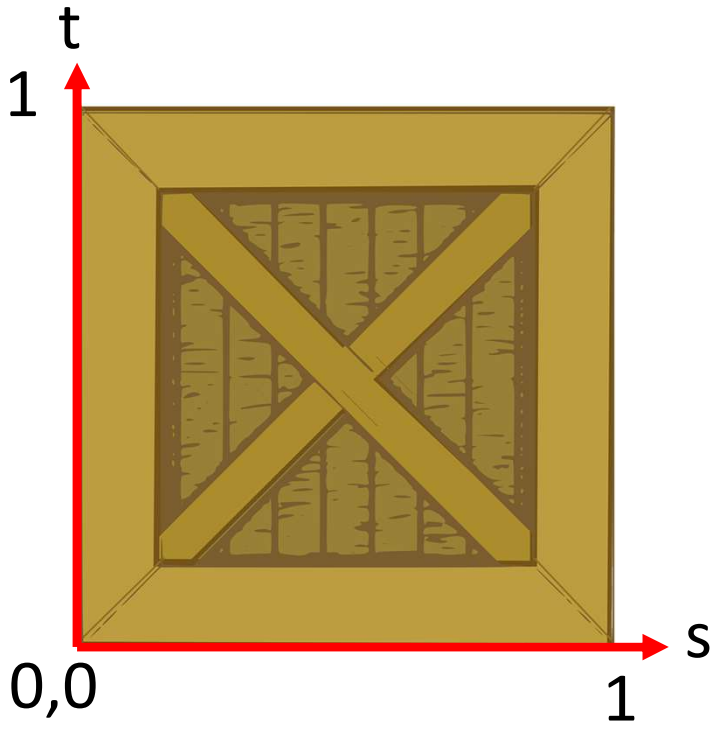
u,v coordinates
(resolution)



s,t coordinates
(normalized)

Texture Coordinates

- We are going to use s,t coordinates
- Normalized, resolution independent coordinates
- In general, we say u,v coordinates for textures but we actually mean s,t coordinates



s,t coordinates
(normalized)

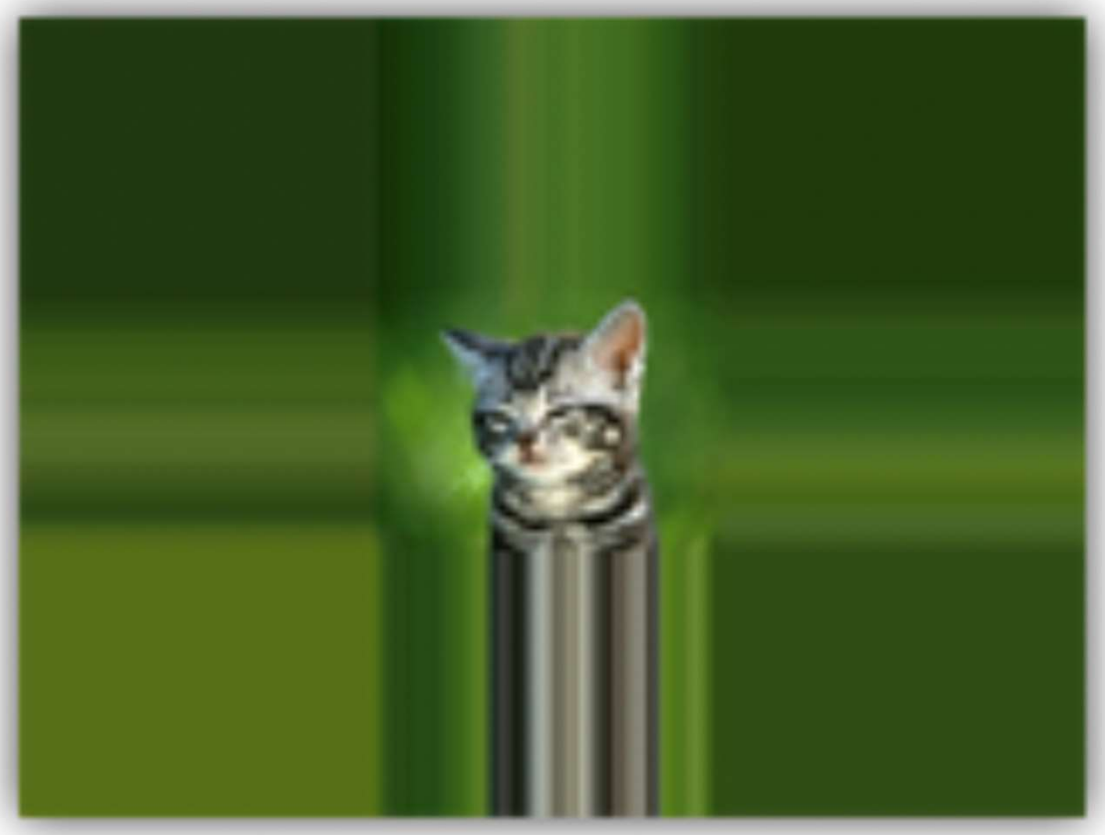
Texture Tiling

Texture Tiling

How do we handle the queries that are outside the range of this texture?

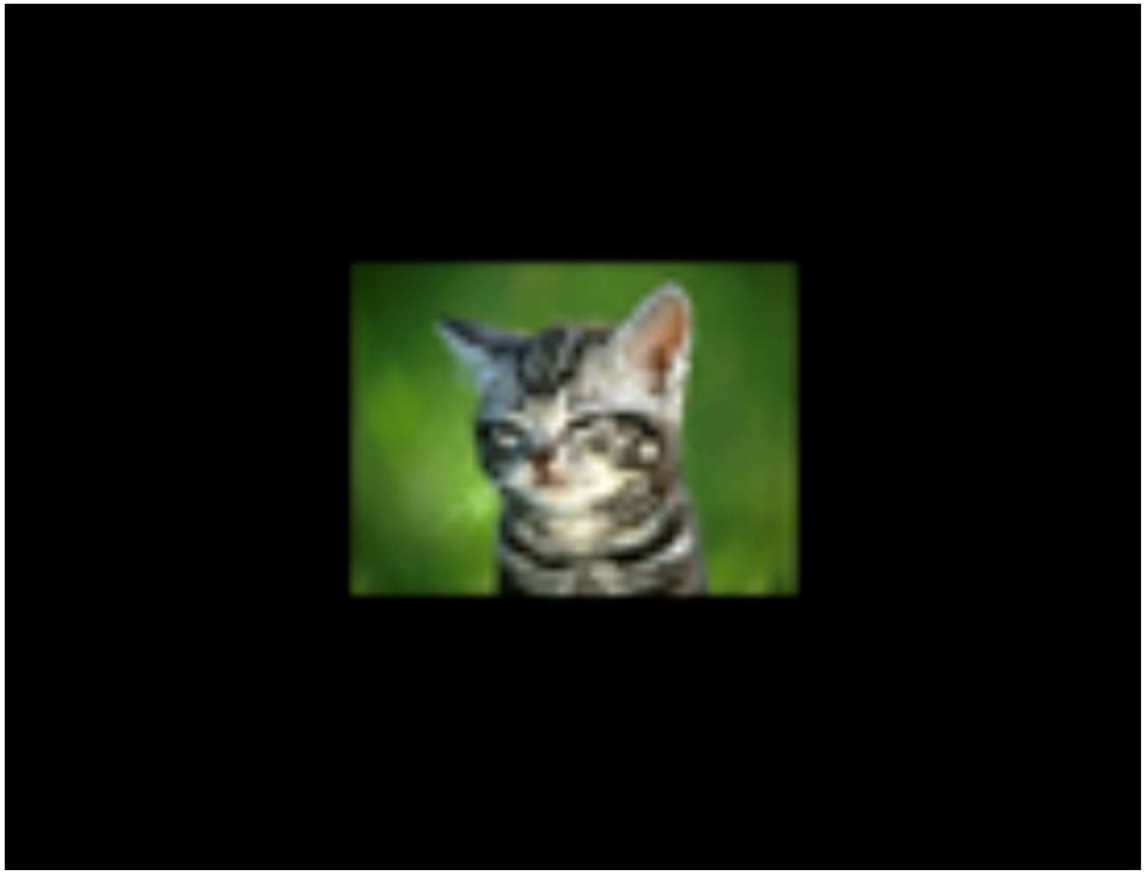
Texture Tiling

CLAMP_TO_EDGE



Texture Tiling

CLAMP_TO_BORDER

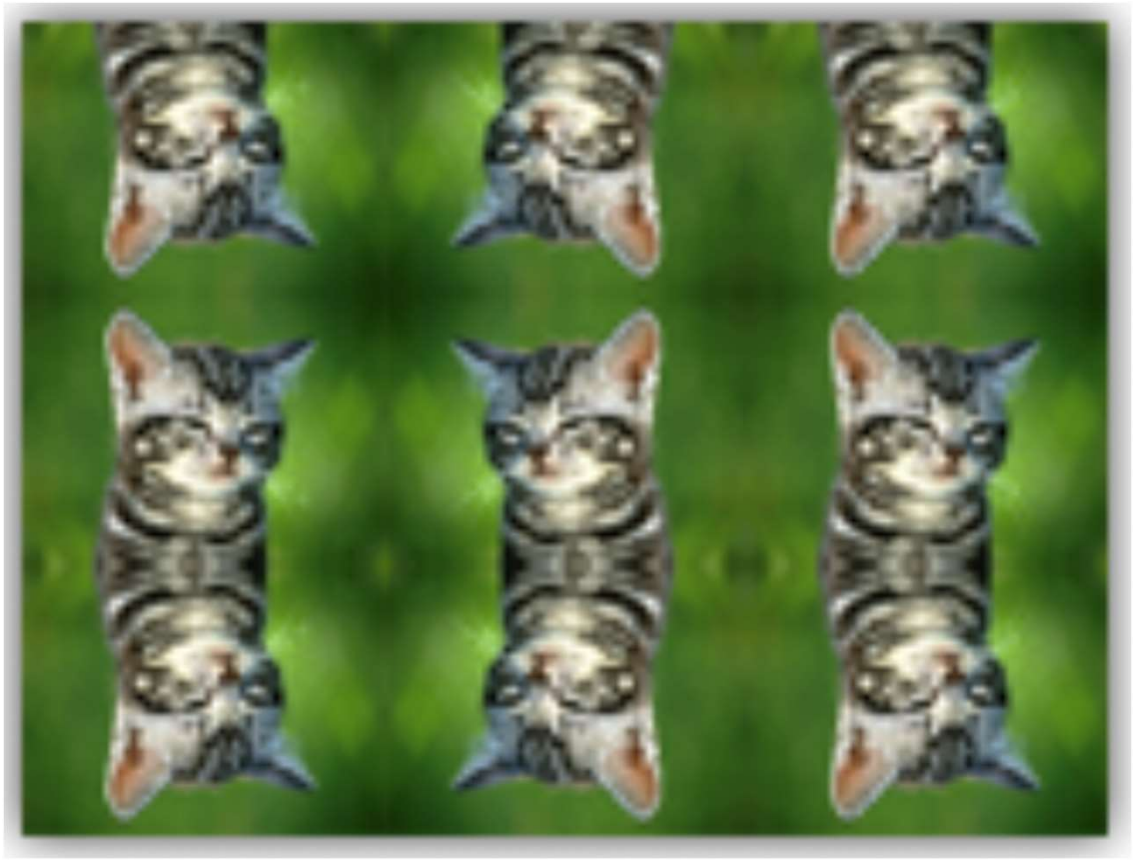


Texture Tiling

REPEAT

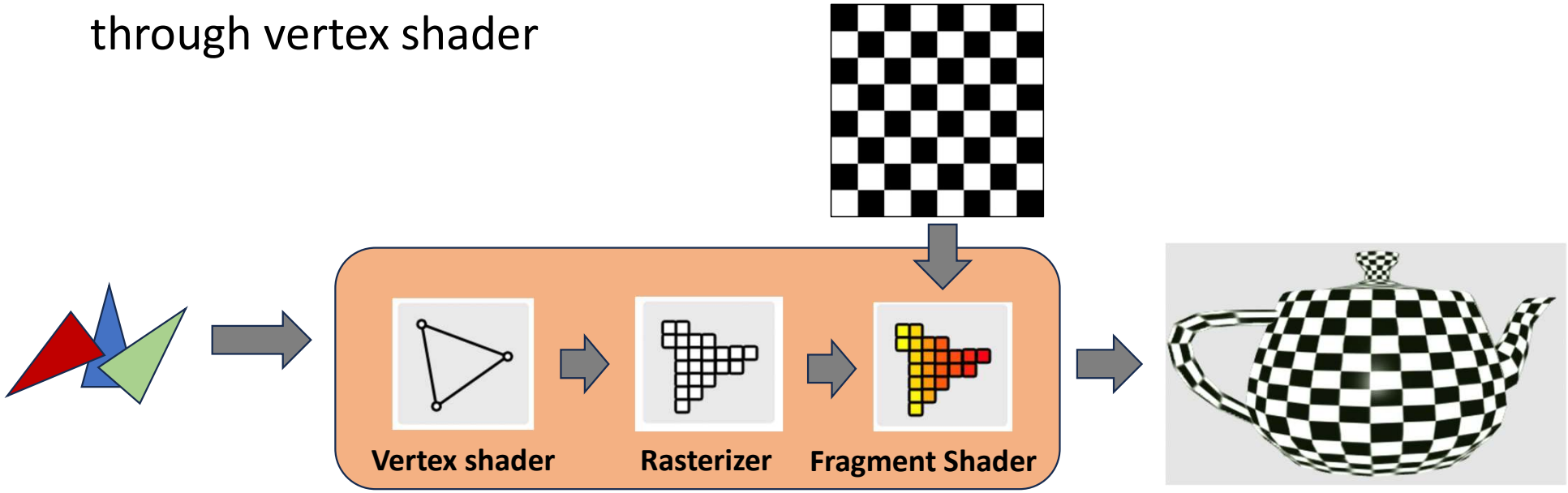
Texture Tiling

MIRRORED_REPEAT

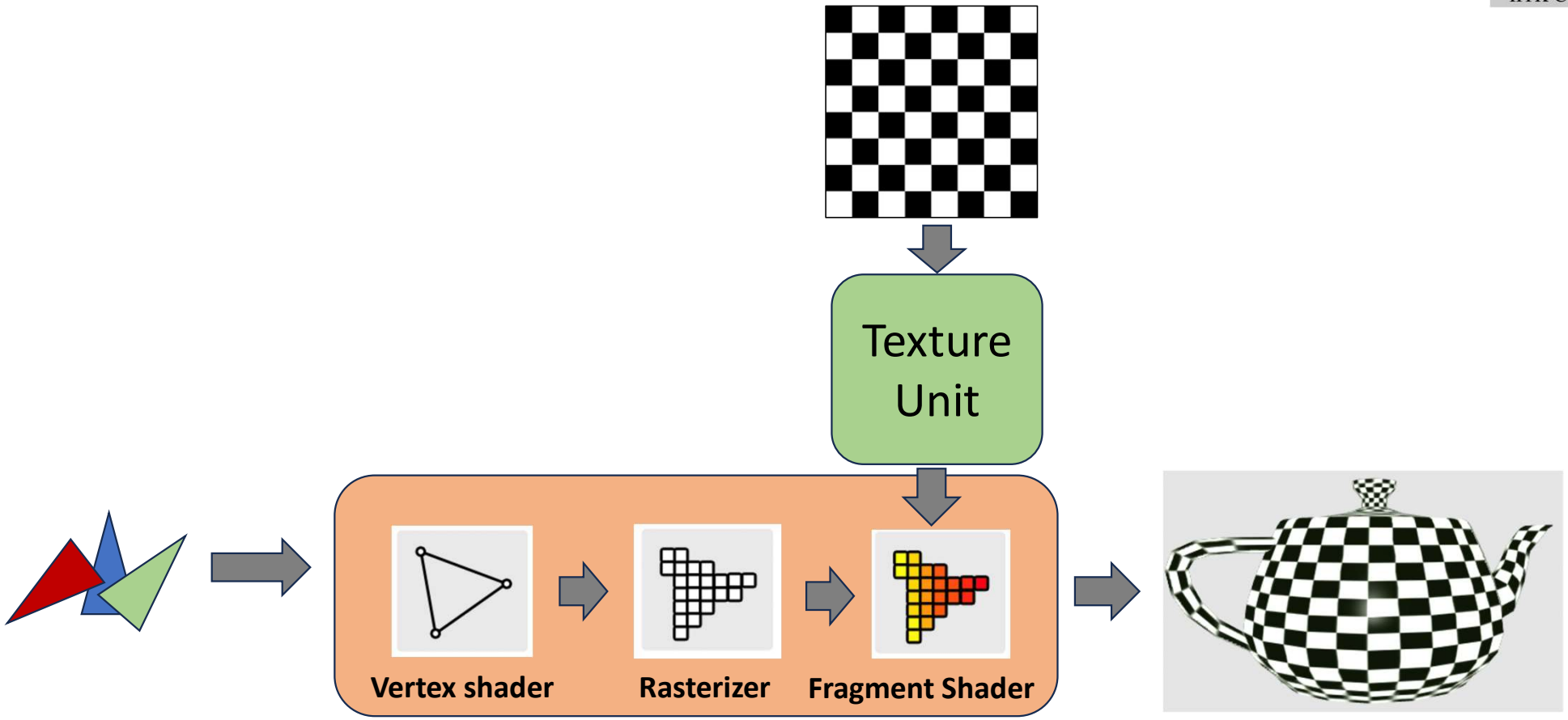


Revisit Texture Mapping: GPU Pipeline

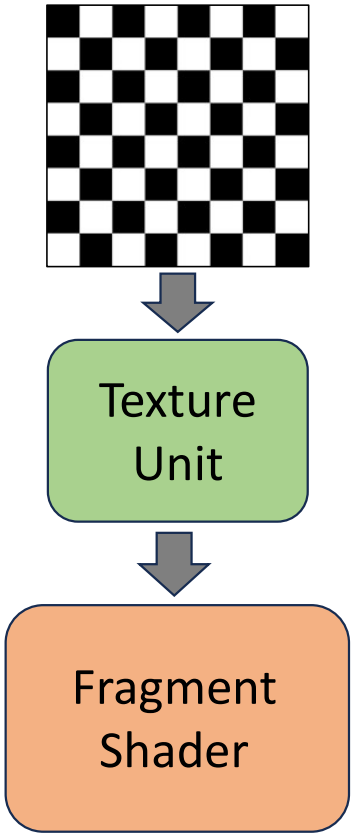
- Texture mapping happens in fragment shader
- Pass vertex texture coordinates through vertex shader



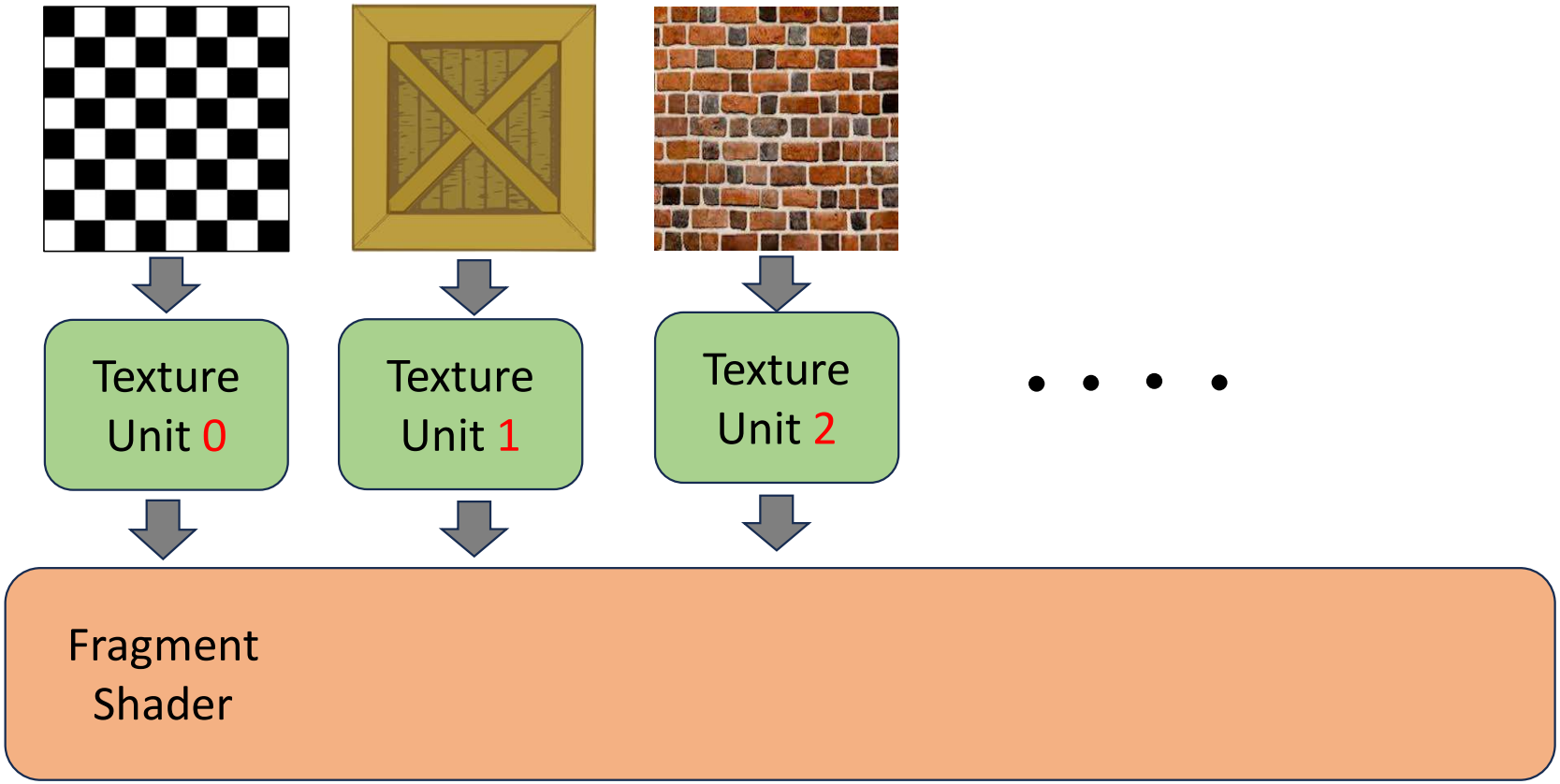
Texture Units: GPU Pipeline



Texture Units: GPU Pipeline



Texture Units: GPU Pipeline



Textures in WebGL


```
function initTextures(textureFile) {
  var tex = gl.createTexture();
  tex.image = new Image();
  tex.image.src = textureFile;
  tex.image.onload = function () {
    handleTextureLoaded(tex);
  };
  return tex;
}
```

Textures in WebGL


```
function handleTextureLoaded(texture) {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(
    gl.TEXTURE_2D,          // 2D texture
    0,                      // mipmap level
    gl.RGBA,                // Internal format
    gl.RGBA,                // format
    gl.UNSIGNED_BYTE,       // type
    texture.image           // array or <img>
  );
  gl.generateMipmap(gl.TEXTURE_2D);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR
  );
}
```

Textures in WebGL: Binding


```
// for texture binding
gl.activeTexture(gl.TEXTURE0); // set texture unit 0 to use
gl.bindTexture(gl.TEXTURE_2D, sampleTexture); // bind the texture object to the texture unit
```

Textures in WebGL: Fragment Shader


```
const fragShader = `#version 300 es
precision highp float;

out vec4 fragColor;
in vec2 fragTexCoord;
uniform sampler2D uTexture;

void main() {
  fragColor = vec4(0,0,0,1);
  vec4 textureColor =  texture(uTexture, fragTexCoord);
  fragColor = textureColor;
}`;
```

Textures in WebGL: Pass Texture to Shader


```
// lookup texture location in shader
uTextureLocation = gl.getUniformLocation(shaderProgram, "uTexture");
// pass the texture unit to the shader
gl.uniform1i(uTextureLocation, 0);
```