Texture mapping

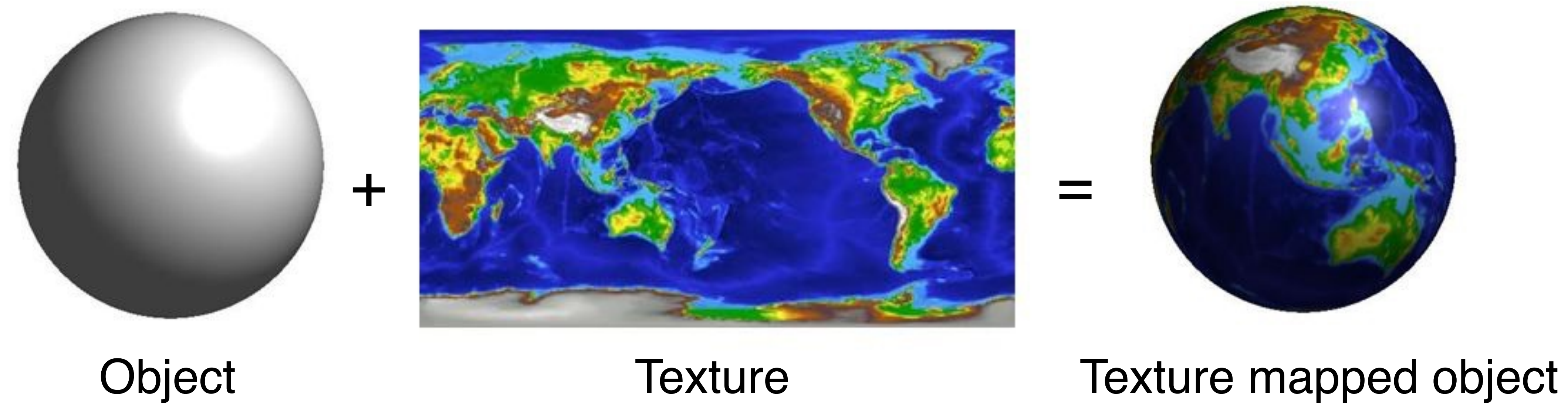
CS425: Computer Graphics I

Khairi Reda

Overview

- Texture mapping
- Texture filtering
- 1D, 2D, 3D textures
- Mipmapping

Texture mapping



Object Texture Texture mapped object

Textures

- Texturing is the process that applies an image, function, or other data source to a polygon.
- Why? Inexpensive way to add realism to a scene.
- Hardware support for 2D and 3D texturing.
- Simple example:
  - Say we want to show a brick wall. Could create geometry for brick, mortar, etc. Gets expensive pretty fast.
  - Easier to “paste” an image of a brick wall to a simple polygon.

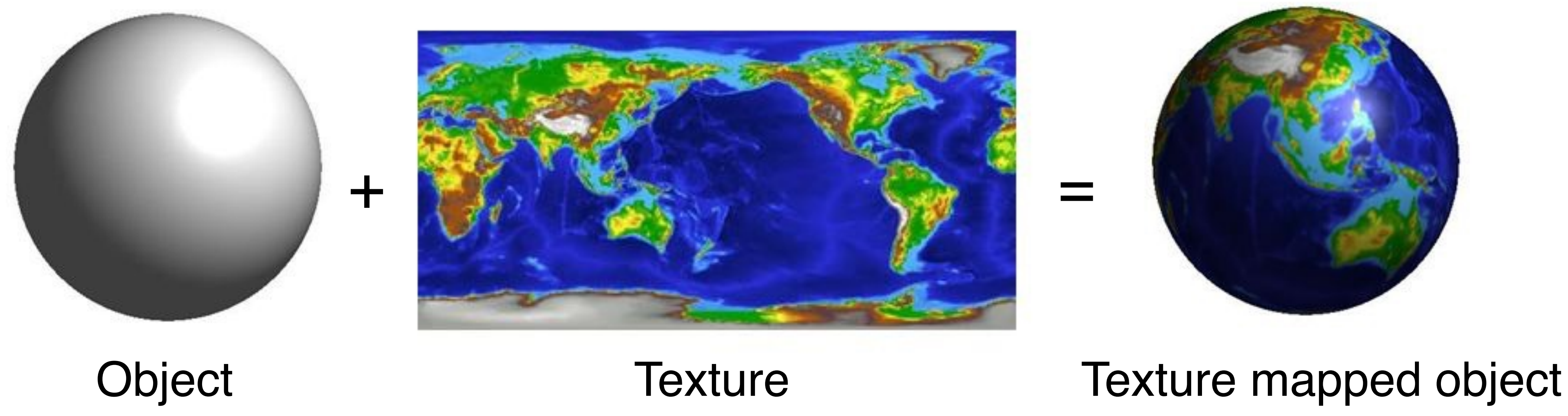
Textures

- **Terminology:**
  - Texture: an array of values
    - 1D, 2D, 3D.
    - Store color, alpha, depth, and even normal.
  - Texel: a single array element.
  - Texture mapping: process of mapping texture to geometry.
- **Source:**
  - Pixel maps: load from an image file.
  - Procedural textures: program generates texel values.

Relationship between textures and shading

- **Shading:** takes into account lighting, material, transparency, position of the viewer.
- **Texturing:** modulates the "base" properties used in the lighting calculation.
  - Replace color by texture color.
  - Modify shininess with specular texture.
  - Modify surface normal with bump maps.

Texture mapping



Object

Texture

Texture mapped object

Texture mapping

1

s

0

t

1

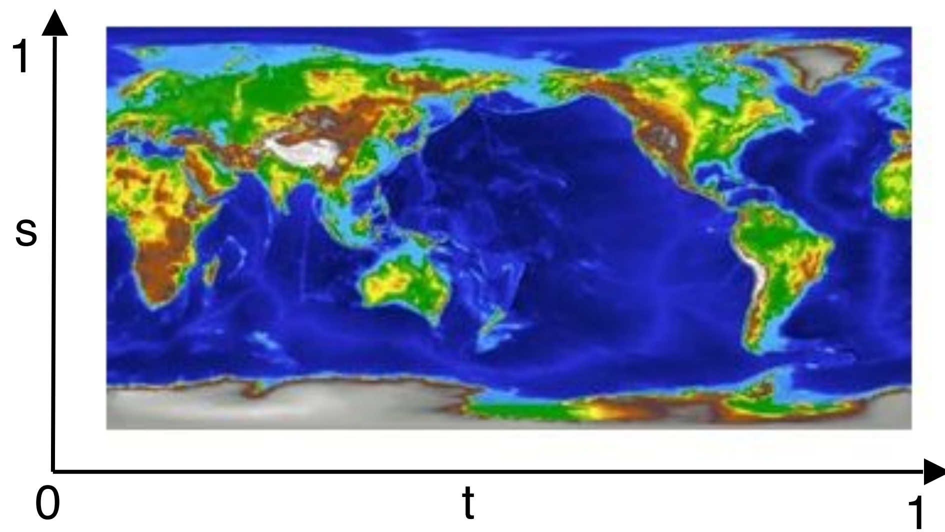
Build a mapping between the texture and the object

Texture mapping

- Problems:
  - Texture and objects are in two different spaces.
  - E.g., Object geometry specified in 3D, texture is in 2D typically
  - Where to specify this mapping?
    - Object space.
    - World space.
  - Complication: a point on the surface could map to a location between texels in the texture. How to handle that?

Texture space

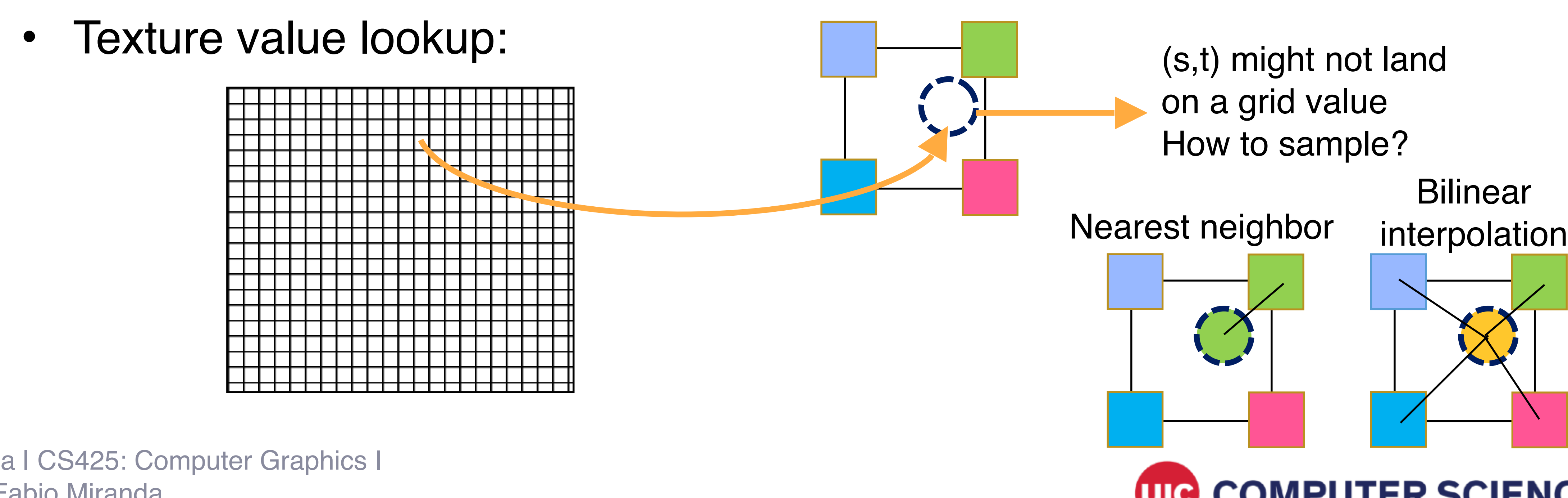
- A texture is defined in a normalized space.



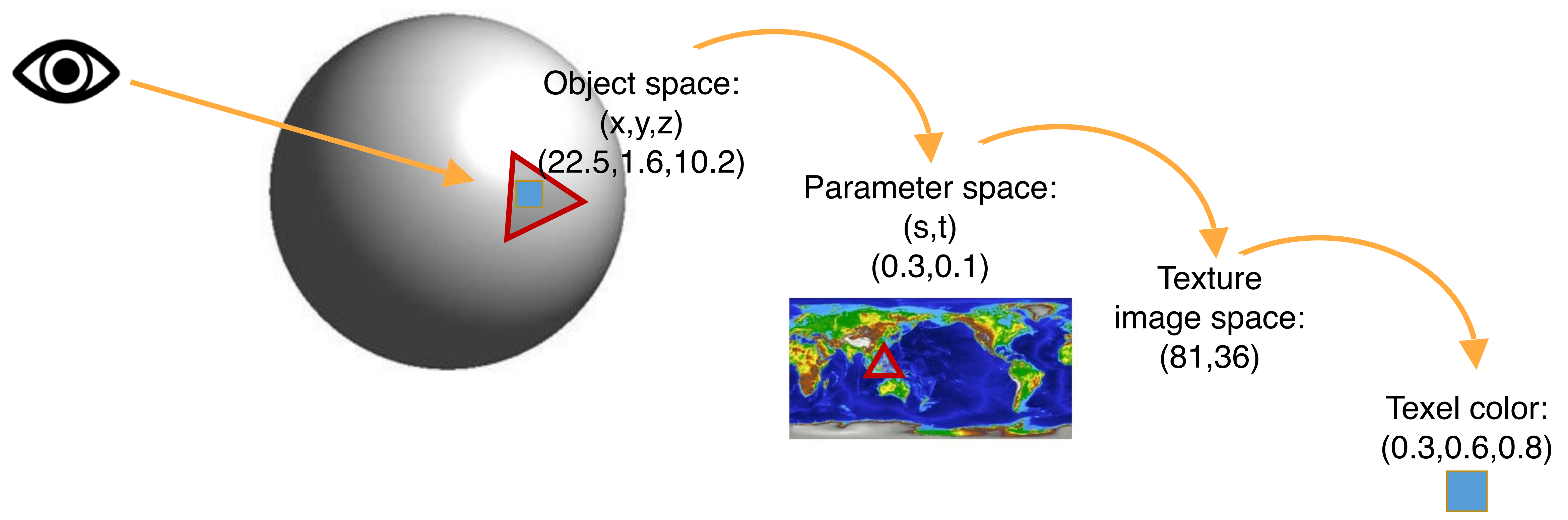
UIC COMPUTER SCIENCE

Texture space

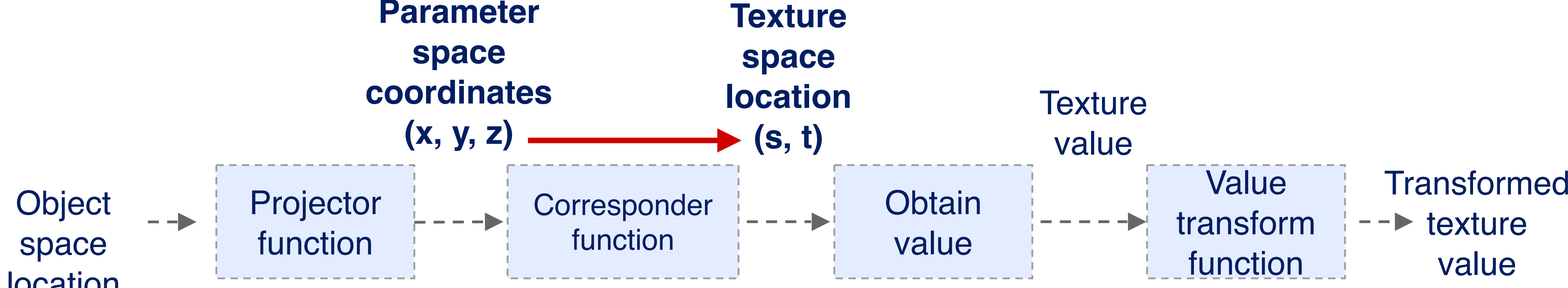
- Texture space is continuous, but textures are discrete arrays.
- Texel values are defined on a cartesian (continuous) grid.
- Texture value lookup:



Texturing a single fragment



Texture pipeline



Projector function

- Defines mapping between object (x,y,z) and parameter space (s,t).
- How to find this mapping function?
  - Given a fragment, we want to know to which point in the texture to map to.
  - We want a map in the form:

$$s = s(x, y, z)$$
$$t = t(x, y, z)$$

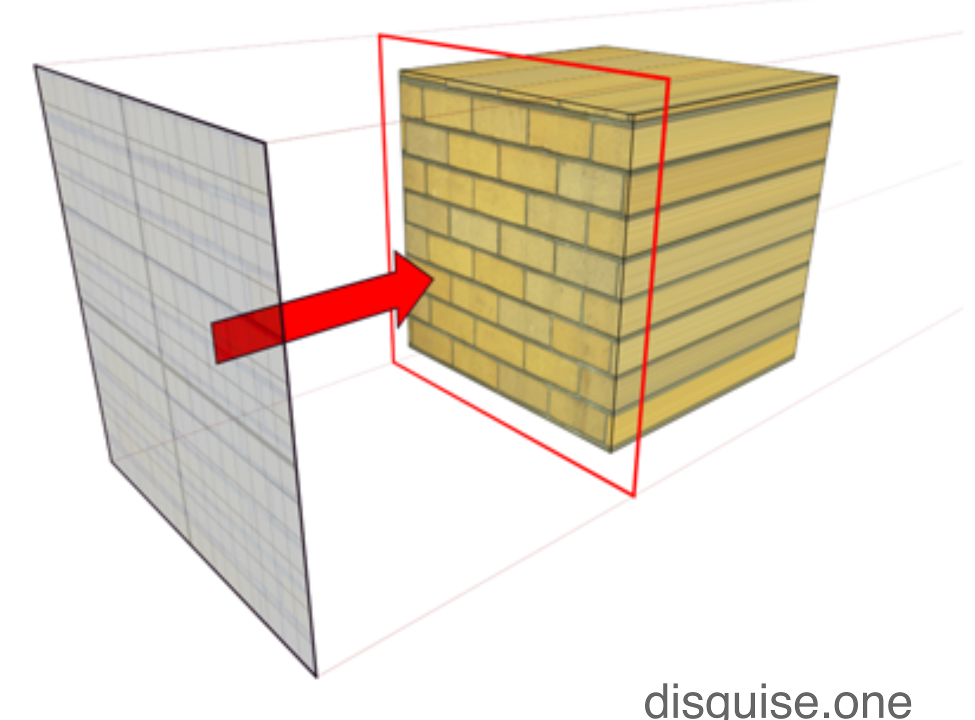
Projector function

- These mapping functions are difficult to find in general.
- One solution is to use an intermediate mapping:
  - Map the texture onto a simple intermediate surface.
  - Map the intermediate surface to the final object.
  - Intermediate objects:
    - Plane
    - Sphere
    - Cylinder

Planar mapping

- Convert (x,y,z) point to (x,y).
- Simply drop z coordinate:

$$s = x, t = y$$



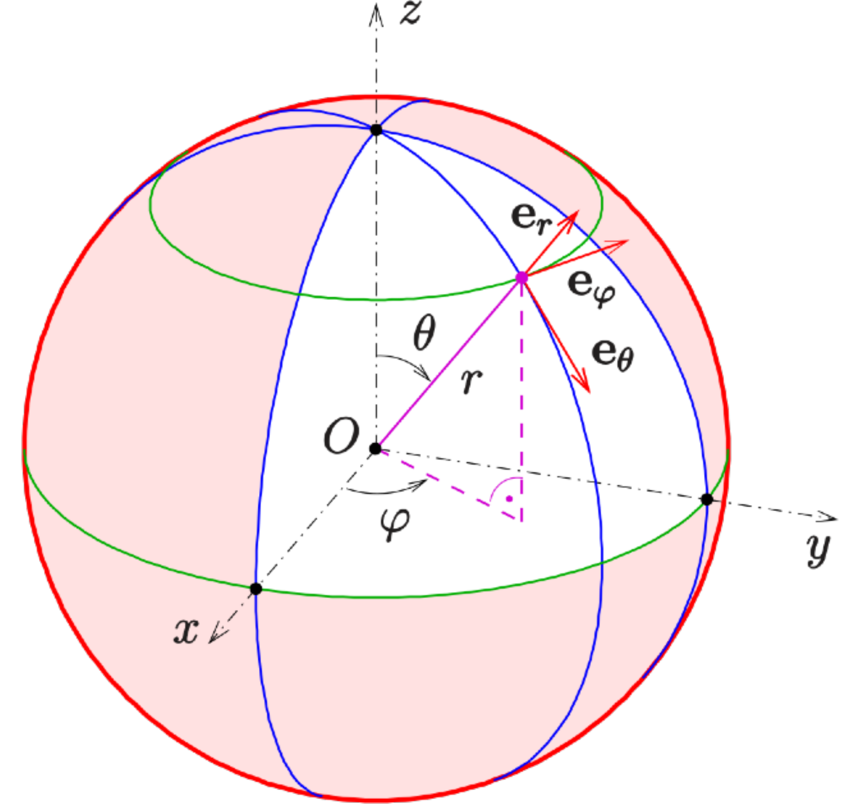
disguise.one

Spherical mapping

- Convert (x,y,z) point to spherical coordinate (θ, φ).

$$\theta = tan^{-1}\left(\frac{\sqrt{x^2 + y^2}}{z}\right)$$

$$\varphi = tan^{-1}\left(\frac{y}{x}\right)$$

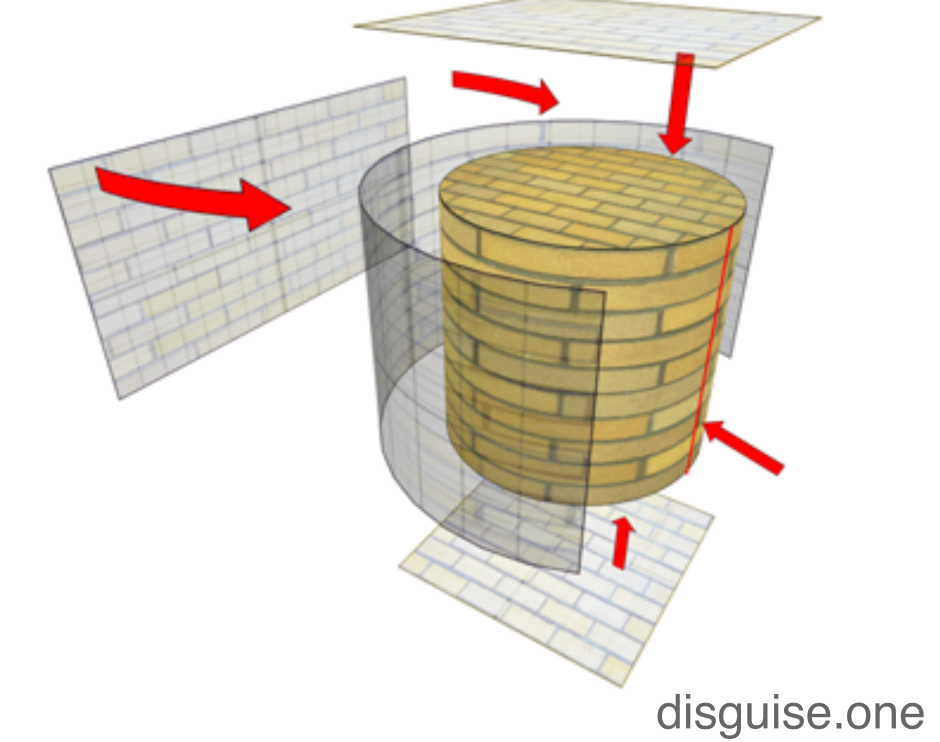


Cylindrical mapping

- Convert (x,y,z) point to cylindrical coordinates (θ, ρ).

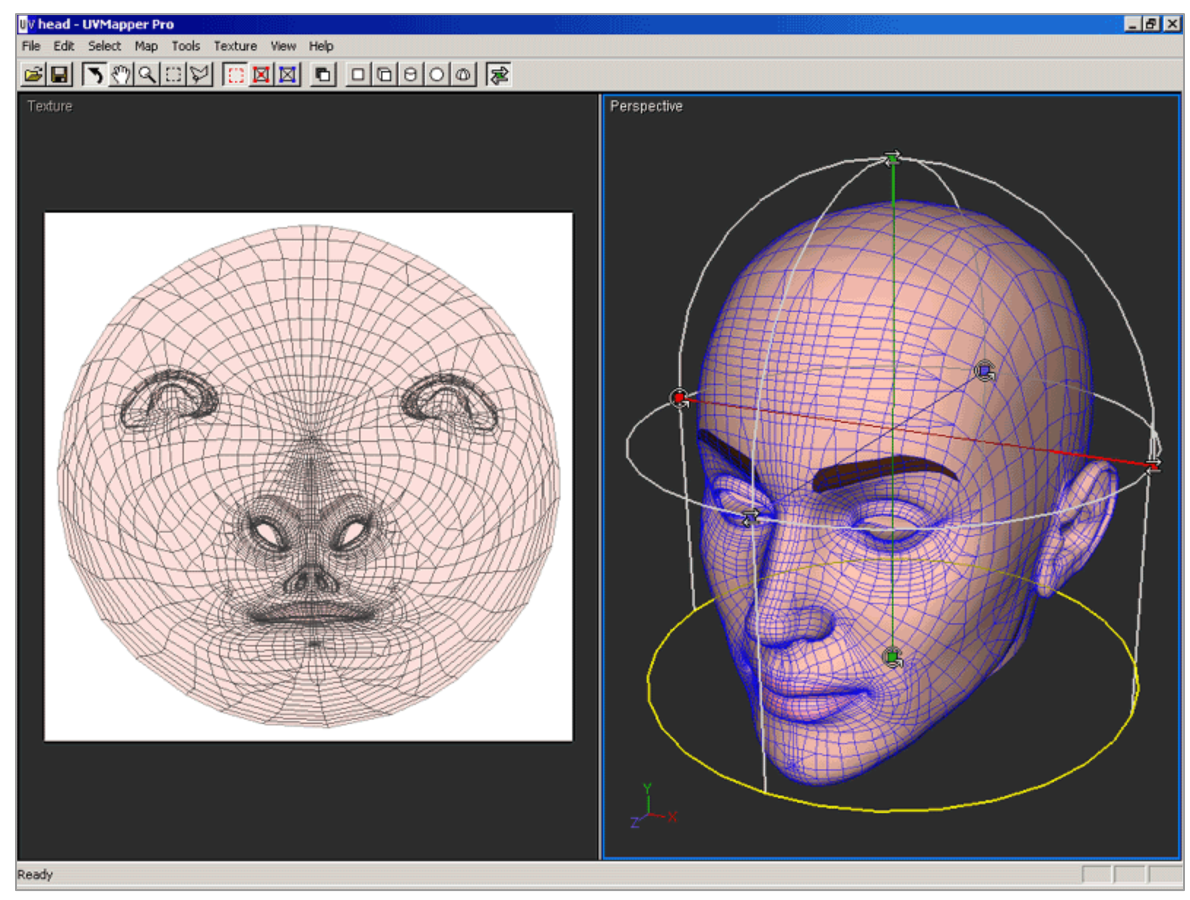
$$\theta = tan^{-1}\left(\frac{y}{x}\right)$$

$$\rho = \sqrt{x^2 + y^2}$$



disguise.one

Texture mapping for meshes

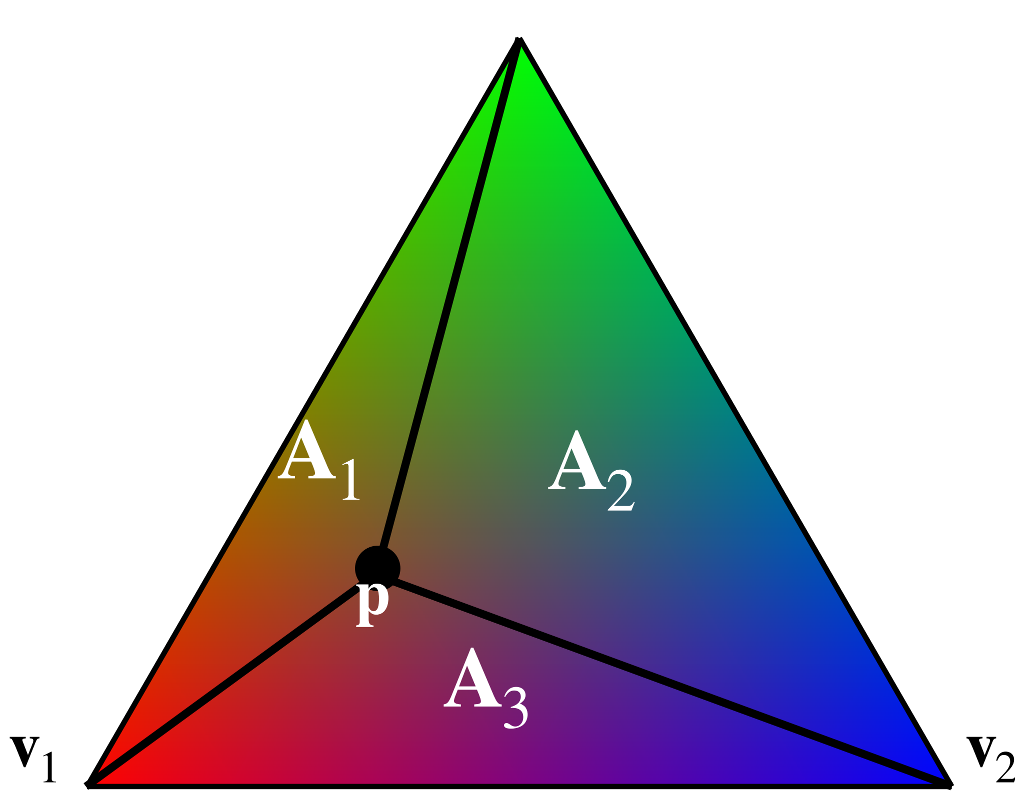


uvmapper.com

Texture mapping for triangles

- Assign texture coordinates to vertices.
- Interpolate within polygon.

Barycentric interpolation



$$f_p = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3$$

$$\alpha_1 = \frac{A_1}{A_{total}}$$

$$\alpha_2 = \frac{A_2}{A_{total}}$$

$$\alpha_3 = \frac{A_3}{A_{total}}$$

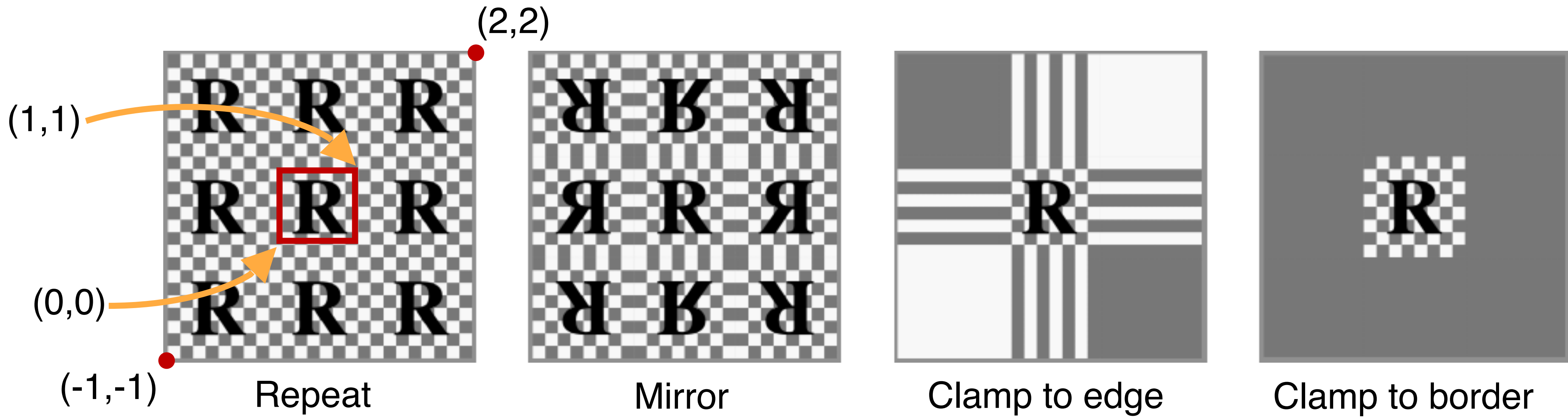
$$area(\mathbf{qrs}) = \frac{1}{2}\|\mathbf{qr} \times \mathbf{qs}\|$$

Corresponder functions

- Defines mapping between parameter-space values to texture-space locations.
- Flexibility in applying textures.
  - Boundary effects (wrapping modes).
  - Subset of an existing texture.

Wrapping modes

Wrapping modes:



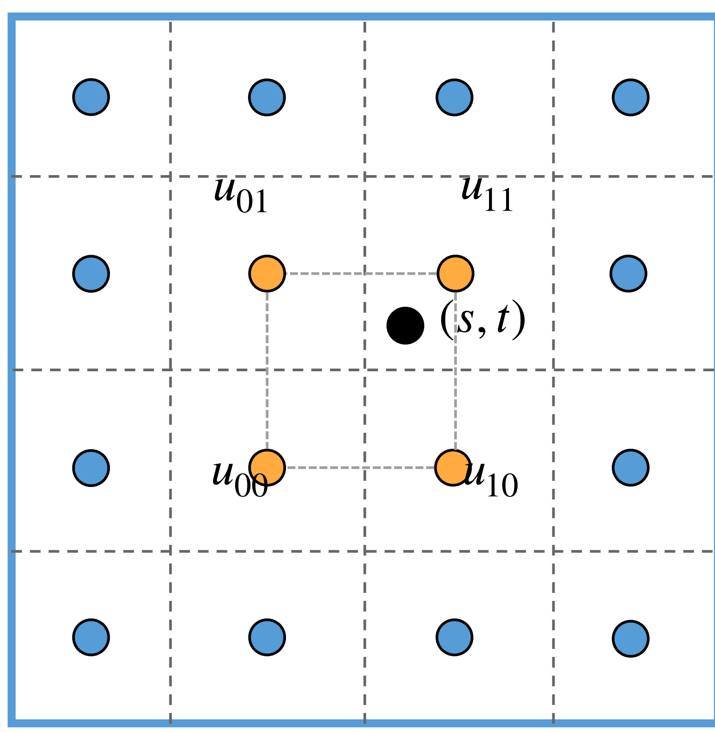
Texture value

- When the number of texels do not correspond to the number of fragments (area), then we have to perform texture **minification** or **magnification**.
- Example:
    - A 256x256 texture on a square.
    - Three cases:
        1. If the projected square on the screen is roughly the same size as the texture, the texture on the square will look almost like the original image.
        2. If the projected square covers ten times as many pixels of the original image, we need to perform **magnification**.
        3. If the projected square covers only a small part of the original image, we need to perform **minification**.

Texture magnification

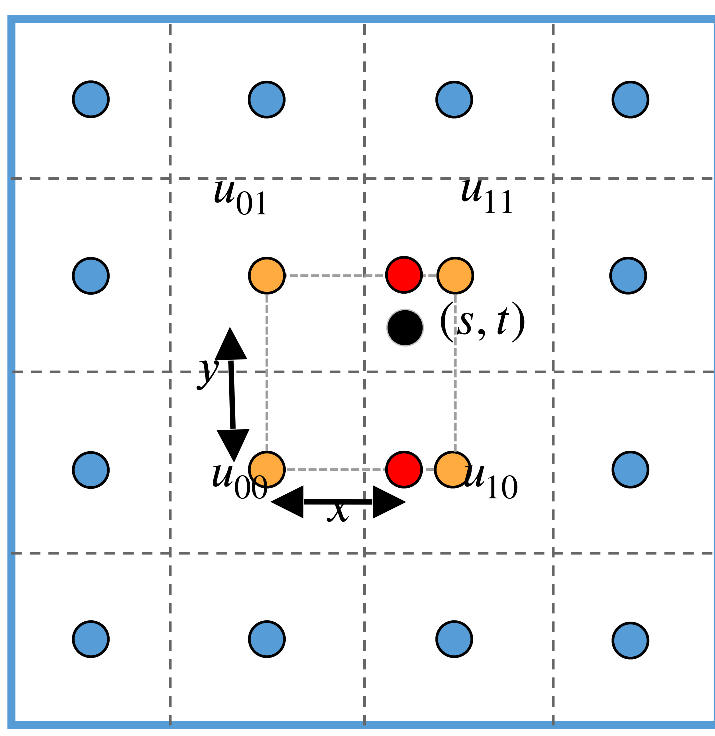
- When many fragments correspond to a single texel value, then we have texture magnification.
- Magnification filters:
  - Nearest: 1 texel per fragment.
  - Linear: 4 texels per fragment.
  - Cubic: 16 texels per fragment.

Bilinear interpolation



- Integral texel coordinates
- Texel centers
- Sample (s, t) coordinates

Bilinear interpolation



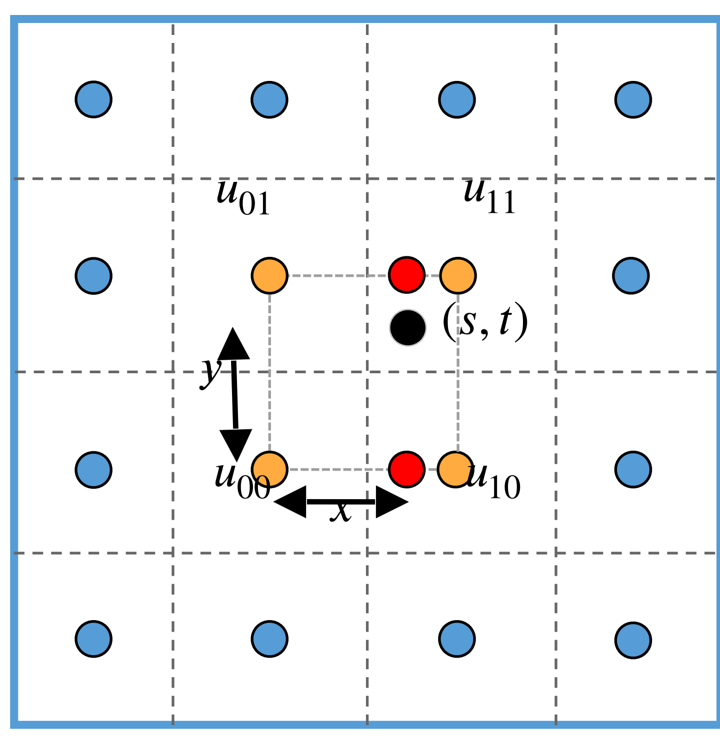
Linear interpolation (1D):

$$lerp(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two linear interpolations (horizontal):

$$u_0 = lerp(s, u_{00}, u_{10}) u_1 = lerp(s, u_{01}, u_{11})$$

Bilinear interpolation



Linear interpolation (1D):

$$lerp\left(x, v_0, v_1\right) = v_0 + x(v_1 - v_0)$$

Two linear interpolations (horizontal):

$$u_0 = lerp\left(s, u_{00}, u_{10}\right) u_1 = lerp(s, u_{01}, u_{11})$$

Final linear interpolation:

$$f(s, t) = lerp(t, u_0, u_1)$$

Texture magnification

From: “Real-Time Rendering”

48x48 image projected onto 320x320 pixels

- Left: nearest neighbor filtering.
- Middle: linear interpolation (4 nearest texels).
- Right: cubic filtering (16 nearest texels).

Texture minification

- When many texels correspond to a single fragment, then we have texture minification.
- Minification filters:
  - Nearest: 1 texel per fragment.
  - Linear: 4 texels per fragment.
- Other solution: mipmapping.

Texture minification

Pixel cells

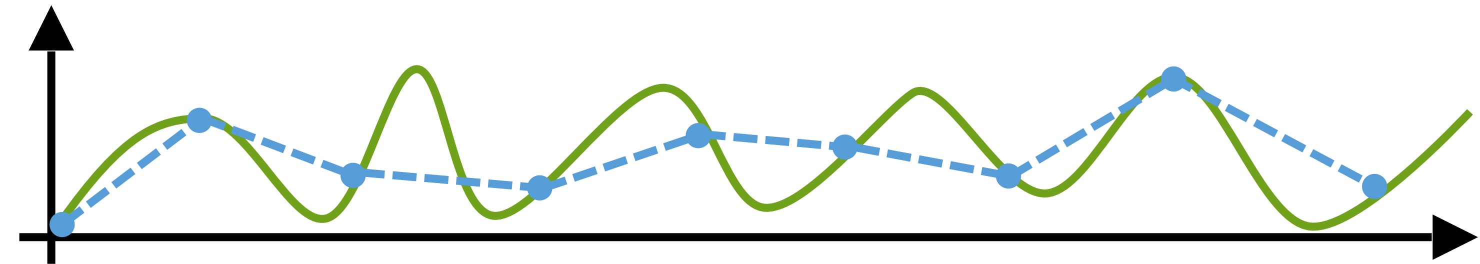
Texture

From: “Real-Time Rendering”

Problem: texture is insufficiently sampled, resulting in distortion and noise

Signal aliasing

- Under sampling a high-frequency signal can result in aliasing.



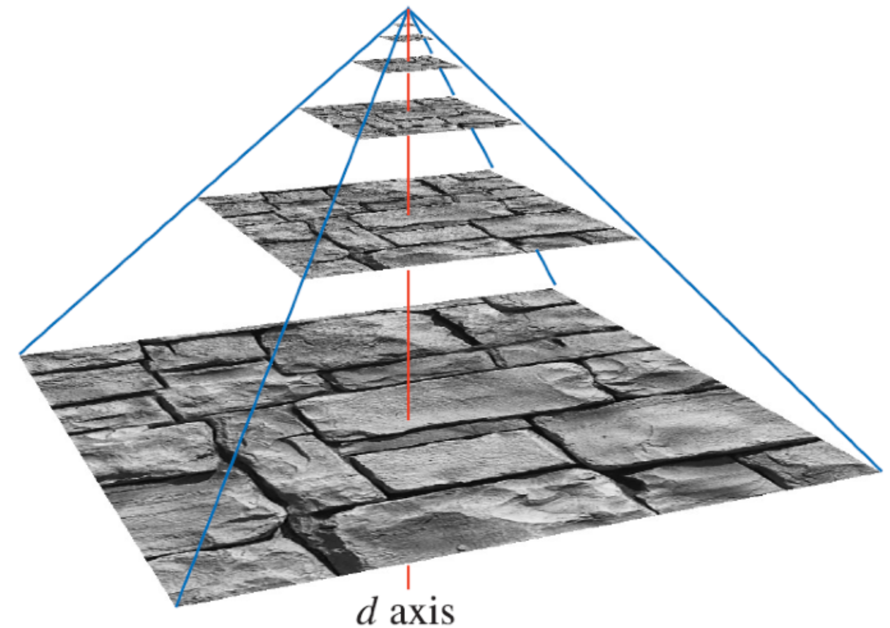
- Nyquist rate: sampling rate must be twice the maximum frequency of the signal.
- How to achieve the Nyquist rate?
  - Increase sampling rate.
  - Decrease signal frequency.

Signal aliasing

- Can we increase the sampling rate? No, number of samples if fixed (screen resolution).
- Can we decrease the signal frequency? Yes, decrease size of texture.

Mipmap

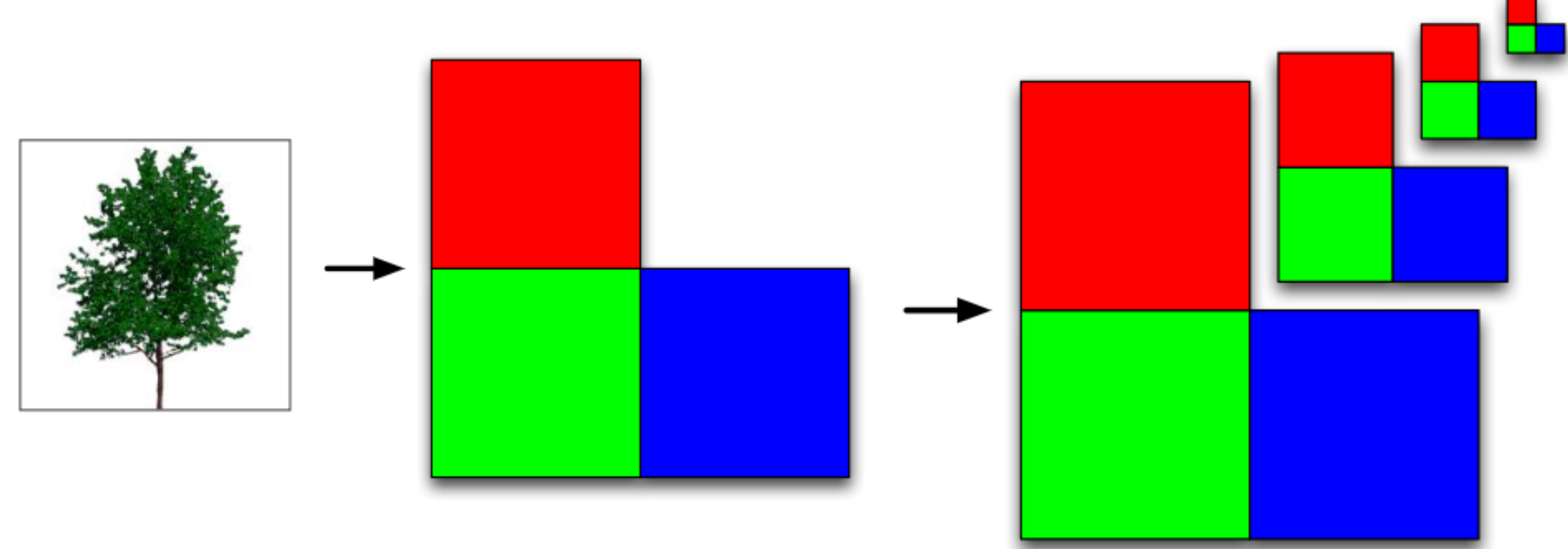
- Original texture is augmented with a set of smaller versions of the texture before rendering takes place.



Recursively until one or both of the dimensions of the texture equals one texel.

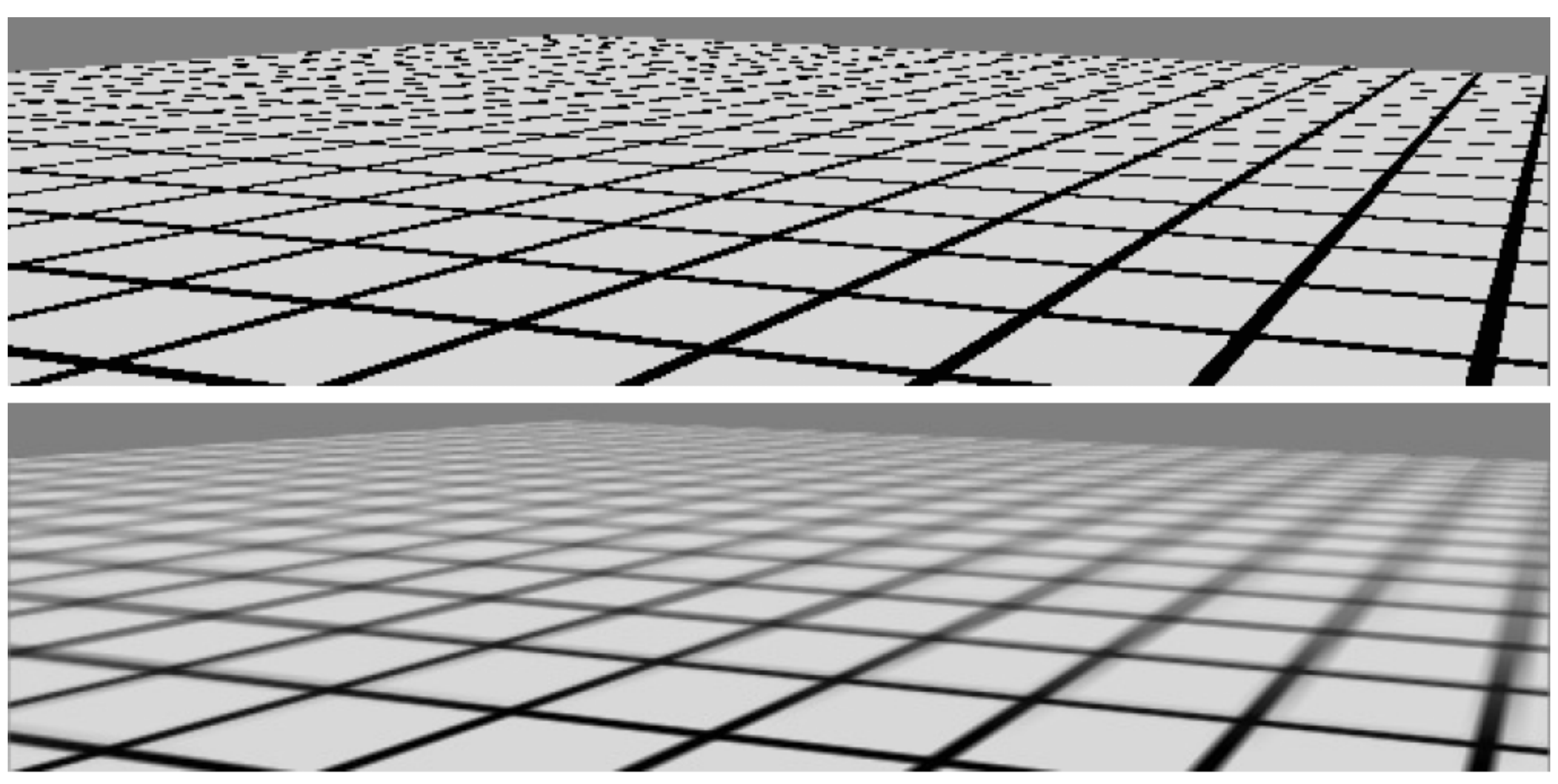
Mipmap

- What is the memory overhead to store the mipmap pyramid?



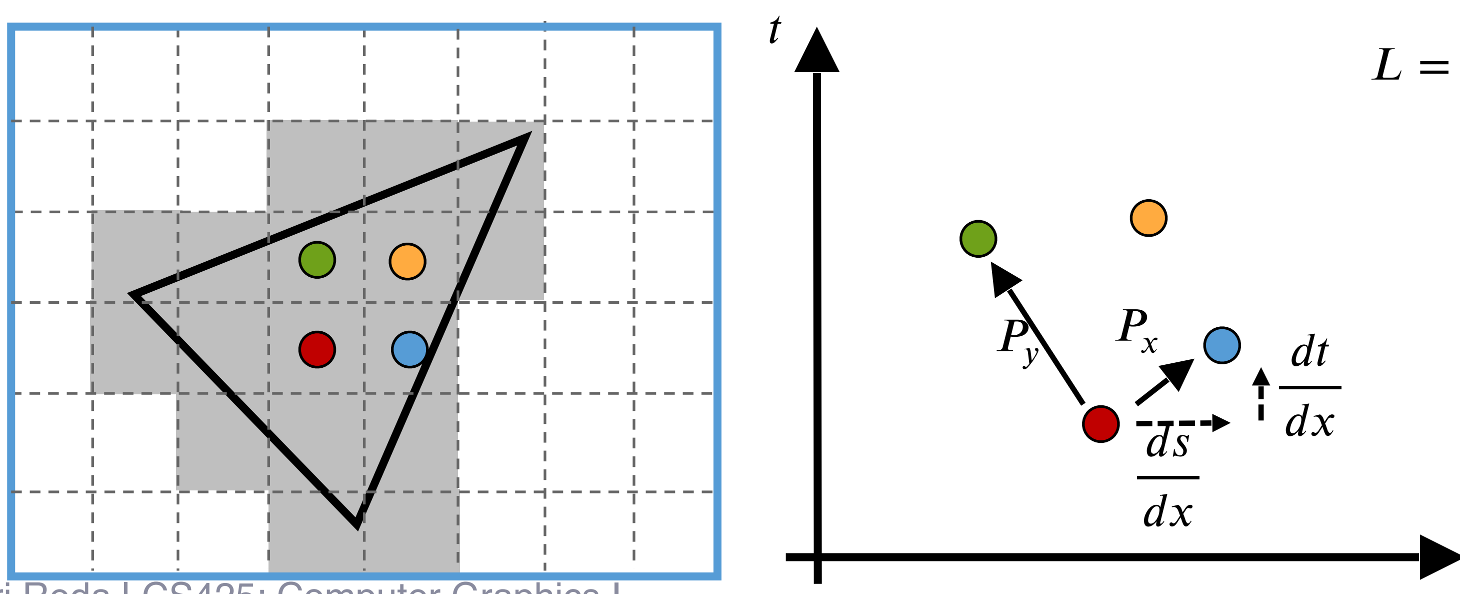
Overhead is 1/3 of the original size.

Mipmap



Mipmap

- *Which* texture in the mipmap pyramid to access when rendering?
  - Compute differences between texture coordinate values of neighboring screen samples.

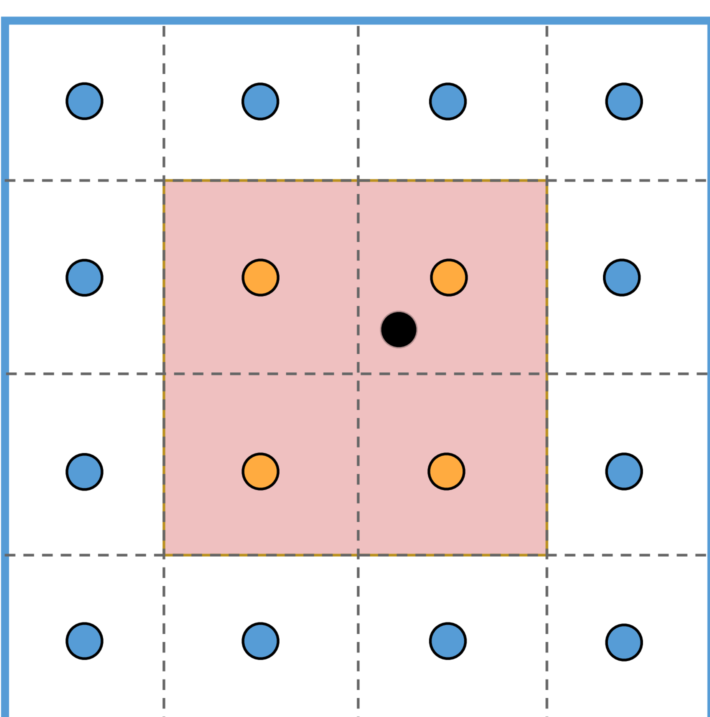


$$L = \max\left(P_x, P_y\right)$$

$$L = \max\left(\sqrt{\left(\frac{ds}{dx}\right)^2 + \left(\frac{dt}{dx}\right)^2}, \sqrt{\left(\ldots\right.}\right.$$

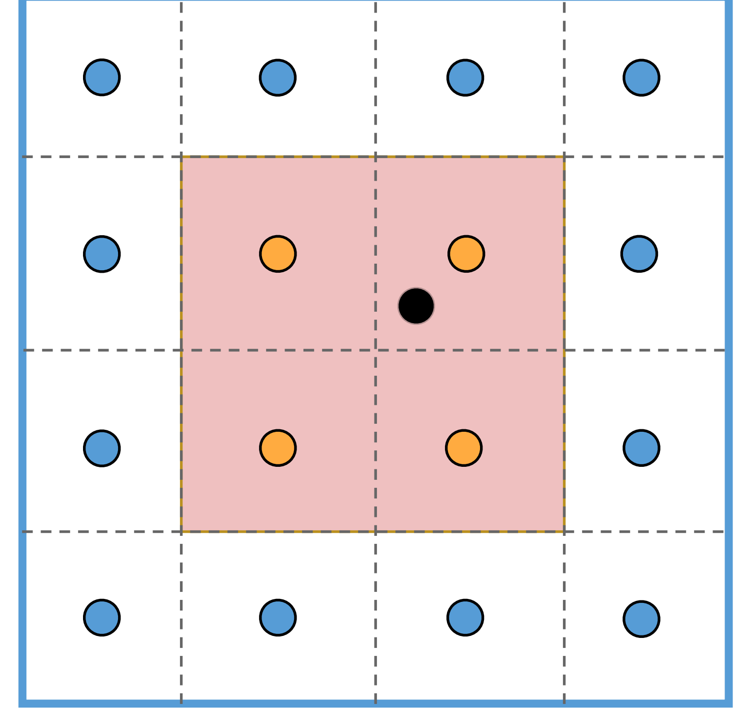
Mipmap: isotropic filtering

Bilinear filtering

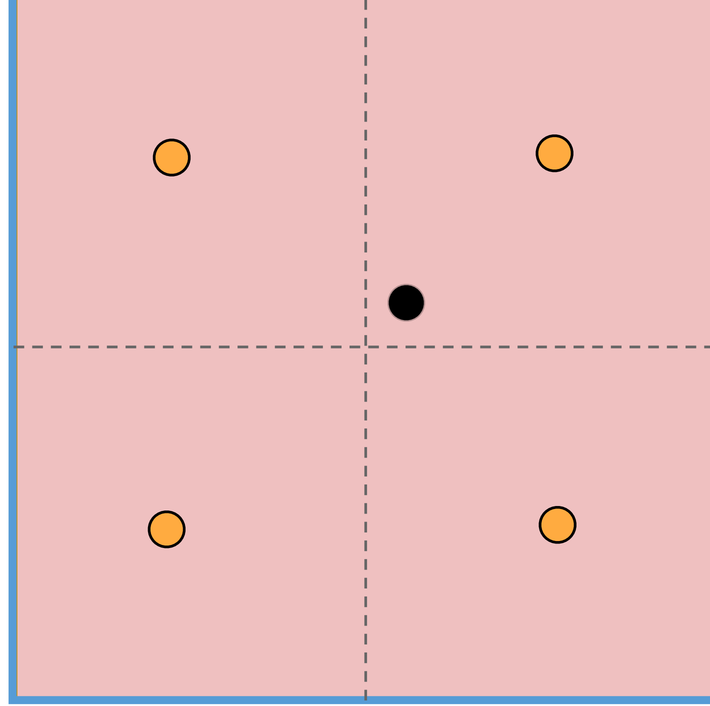


Mipmap level d

Trilinear filtering



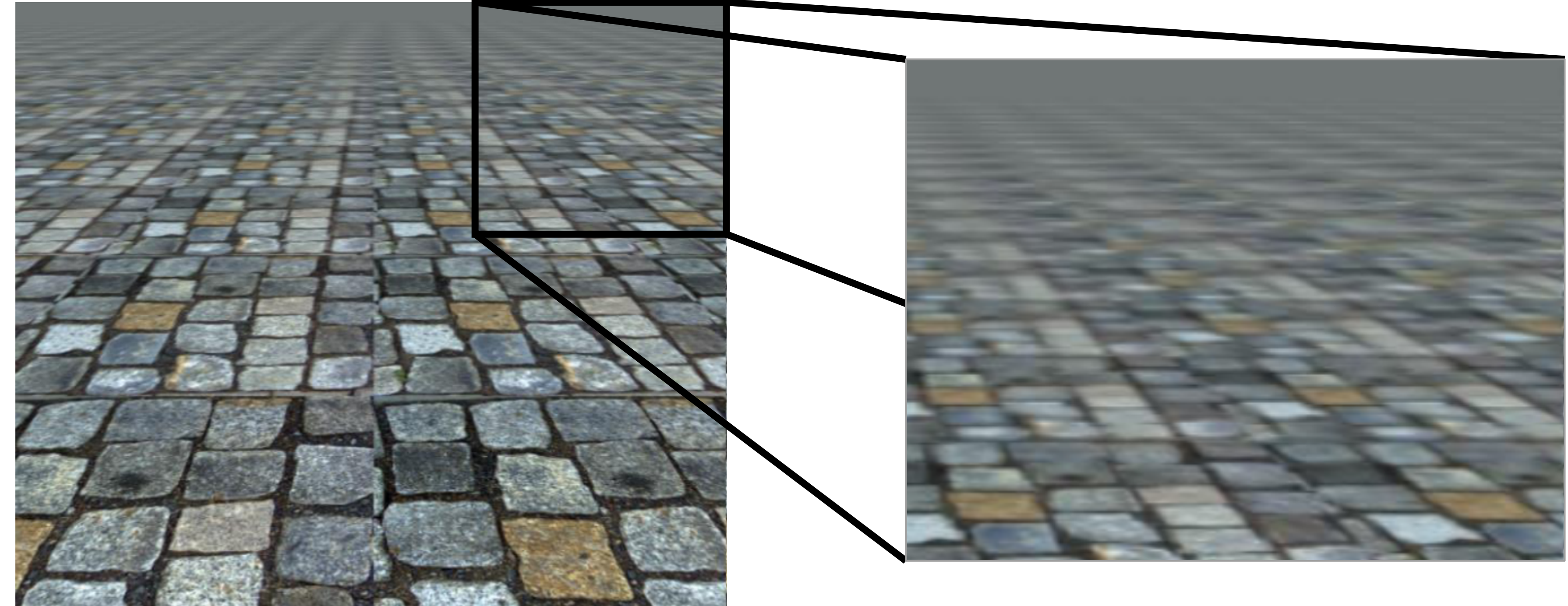
Mipmap level d



Mipmap level d+1

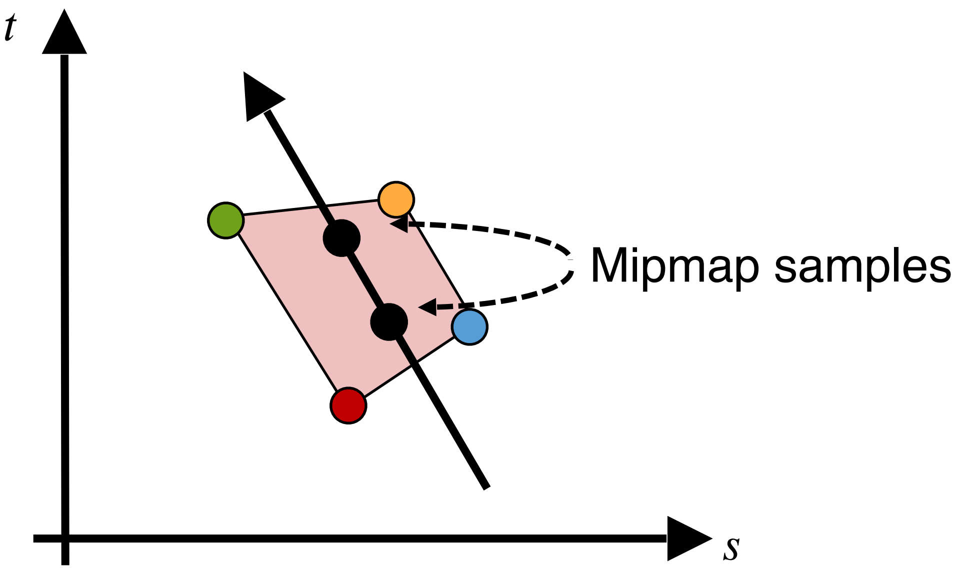
Mipmap: isotropic filtering

- Isotropic approach might lead to over blurring at oblique viewing angles.



Mipmap: anisotropic filtering

- Replace the uniformly-sized filter with multiple smaller mipmap samples.



Mipmap: anisotropic filtering

Mipmap

polycount.com

Textures + WebGL

1. Load image file, create texture.
2. Create texture coordinates buffer, attach to VAO.
3. Texture sampler.
4. Vertex shader: send texture coordinates to fragment shader.
5. Fragment shader: sample texture.

Load texture image

```
function loadTexture(gl, url) {
    var texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    var image = new Image();
    image.onload = function () {
        var level = 0;
        var internalFormat = gl.RGBA;
        var srcFormat = gl.RGBA;
        var srcType = gl.UNSIGNED_BYTE;
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D,level,internalFormat,srcFormat,srcType,image);
        gl.generateMipmap(gl.TEXTURE_2D);
    };
    image.src = url;

    return texture;
}
```

Texture parameters

```
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,gl.LINEAR);// NEAREST LINEAR
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,gl.LINEAR_MIPMAP_LINEAR);
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S,gl.REPEAT);// REPEAT CLAMP_TO_EDGE
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T,gl.REPEAT);// REPEAT CLAMP_TO_EDGE
gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, srcFormat, srcType, image);
gl.generateMipmap(gl.TEXTURE_2D);
```

Texture coordinates

Create buffer:

```
var texCoordsBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, texCoordsBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(tri['texCoord']), gl.STATIC_
DRAW);
```

Texture coordinate attribute variable location:

```
texCoordAttribLoc = gl.getAttribLocation(program, "texCoord");
```

Attach to VAO:

```
gl.enableVertexAttribArray(texCoordAttribLoc);
size = 2;
type = gl.FLOAT;
gl.bindBuffer(gl.ARRAY_BUFFER, buffers['texCoord']);
gl.vertexAttribPointer(texCoordAttribLoc, size, type, false, 0, 0);
```

Texture sampler

Sampler location:

```
samplerLoc = gl.getUniformLocation(program, 'uSampler'),
```

Draw function:

```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(samplerLoc, 0);
```

Shaders

Vertex shader:

```
uniform mat4 uModel;
uniform mat4 uProj;
uniform mat4 uView;

in vec2 texCoord;
in vec3 position;
in vec4 color;

out vec4 vColor;
out vec2 vTexCoord;

void main() {
  vColor = color;
  vTexCoord = texCoord;
  gl_Position = uProj * uView * uMode
l * vec4(position, 1);
}
```

Fragment shader:

```
precision highp float;

in vec2 vTexCoord;
in vec4 vColor;
out vec4 outColor;
uniform vec4 uColor;

uniform sampler2D uSampler;

void main() {
  outColor = texture(uSampler, vTexCoor
d);
}
```