Fall 2015

CSCI 420: **Computer Graphics**

6.1 Texture Mapping

Hao Li

http://cs420.hao-li.com

Outline

- Introduction
- Texture mapping in OpenGL
- Filtering and Mipmaps
- Example
- Non-color texture maps

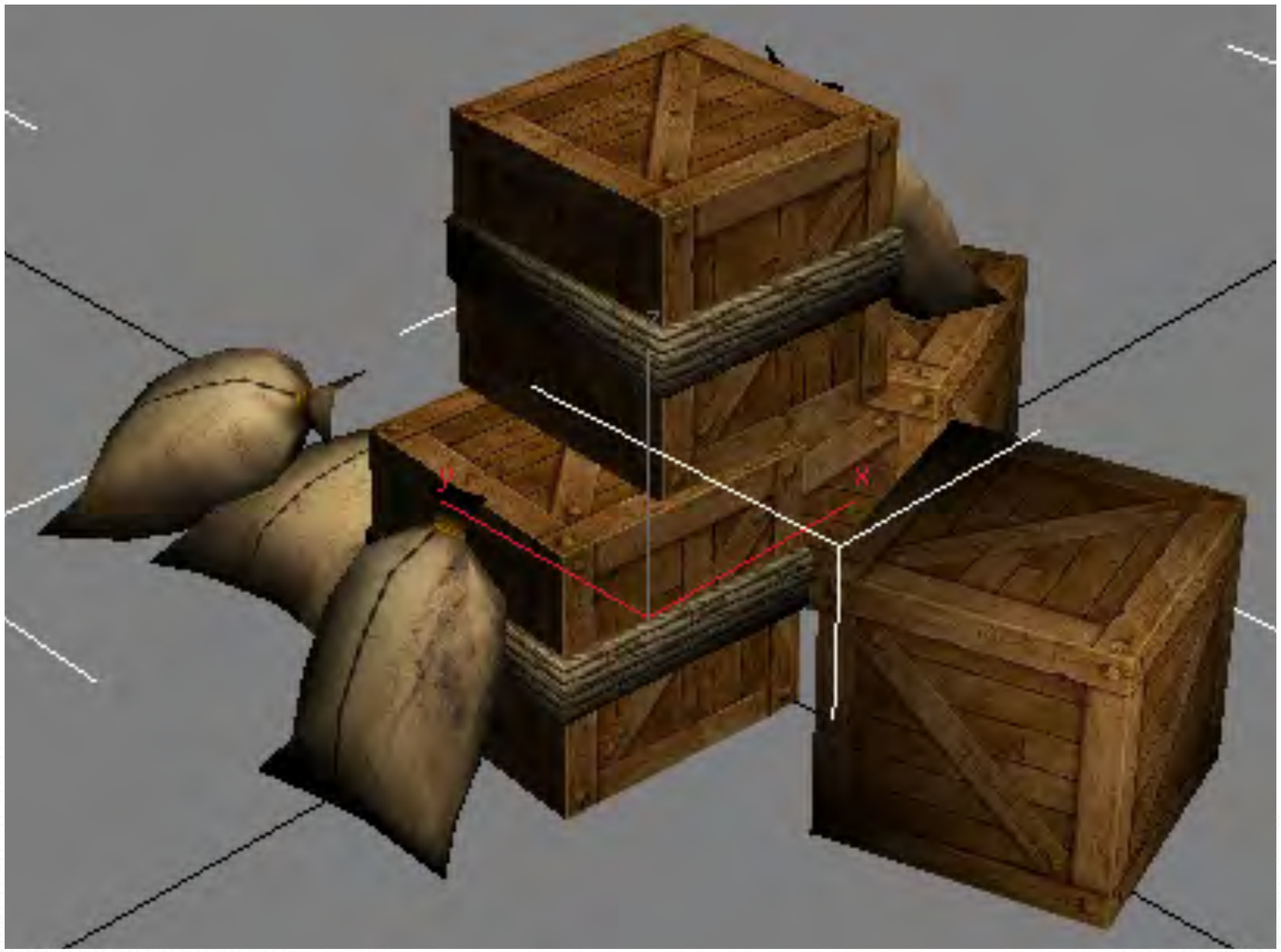
Many uses of texture mapping

Define variation in surface reflectance

Pattern on ball

Wood grain on floor

How Do You Add Detail to a Cube?



six sides - six colors?

Texture Mapping

- A way of adding surface details

- Two ways can achieve the goal:

  - Model the surface with more polygons

    - Slows down rendering speed
    - Hard to model fine features

  - Map a texture to the surface

    - This lecture
    - **Image complexity does not affect complexity of processing**

- Efficiently supported in hardware

Trompe L'Oeil ("Deceive the Eye")

Jesuit Church, Vienna, Austria

- Windows and columns in the dome are painted, not a real 3D object

- Similar idea with texture mapping:

  Rather than modeling the intricate 3D geometry, replace it with an image !

Map textures to surfaces

an image

texture map

image mapped
to a 3D polygon
The polygon can
have arbitrary size,
shape and 3D position

The Texture

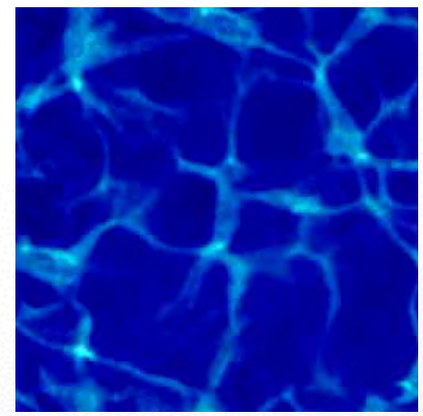
- Texture is a bitmap image
  - Can use an image library to load image into memory
  - Or can create images yourself within the program

- 2D array:
  unsigned char texture[height][width][4]

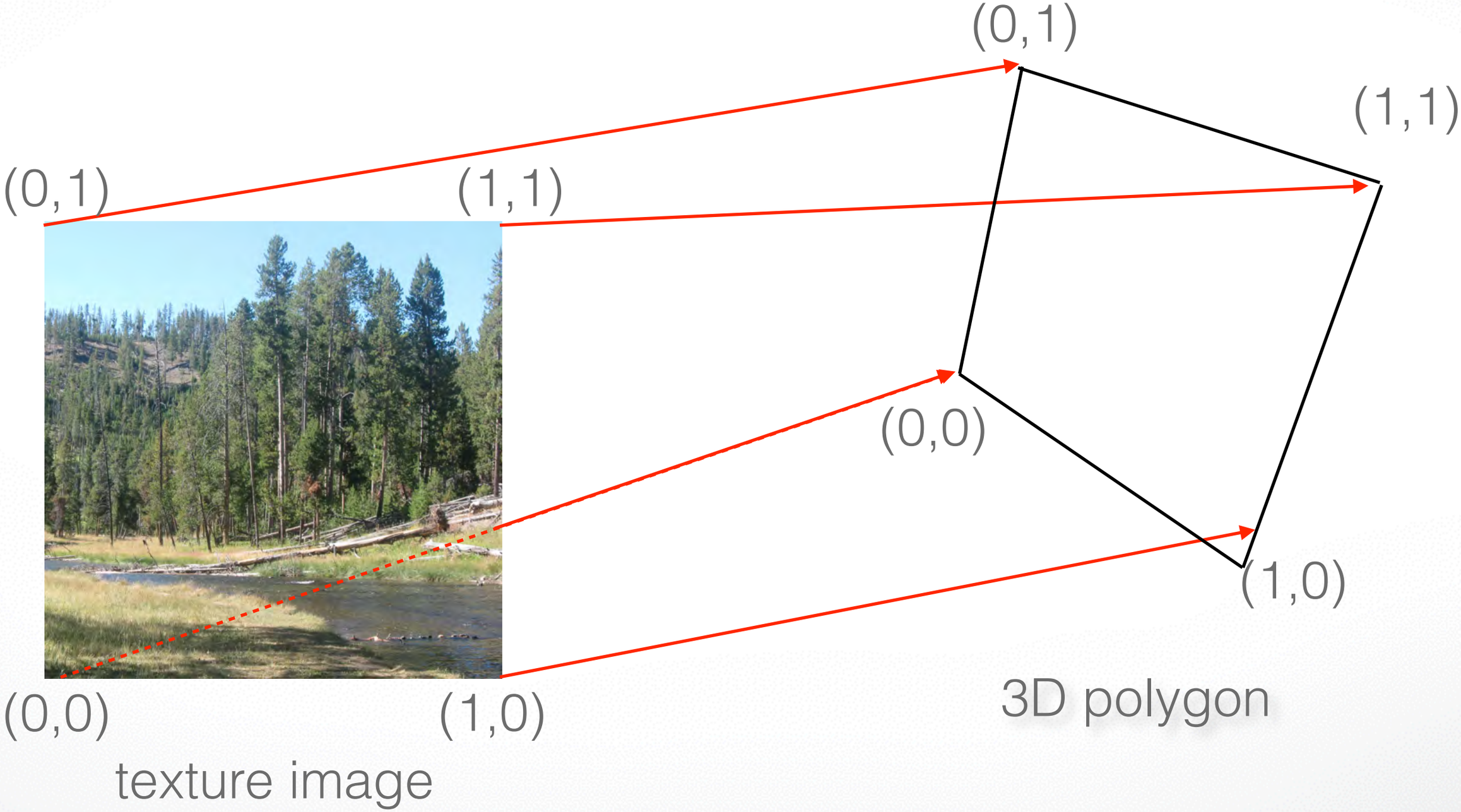
- Or unrolled into 1D array:
  unsigned char texture[4*height*width]

- Pixels of the texture are called *texels*

- Texel coordinates (s,t) scaled to [0,1] range



Texture map

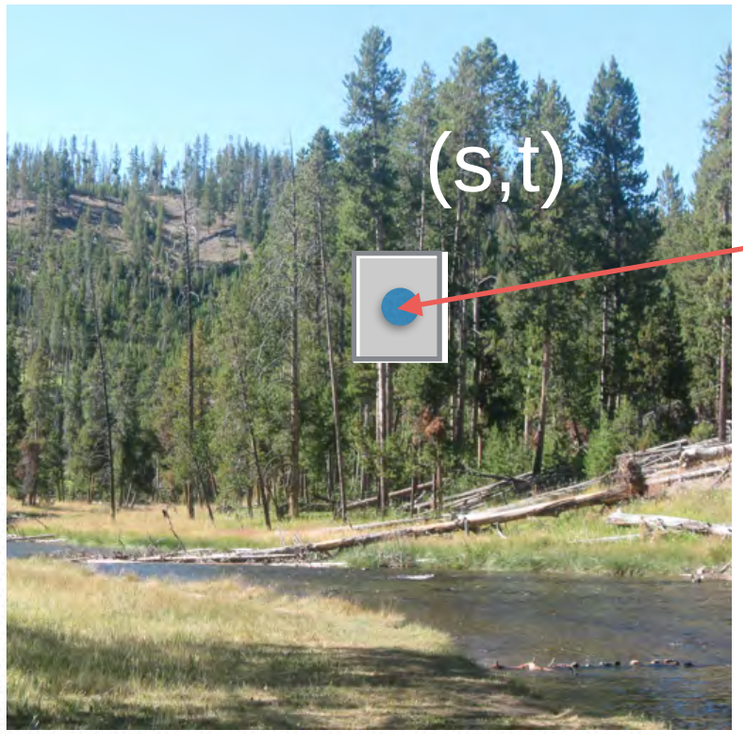


Texture map

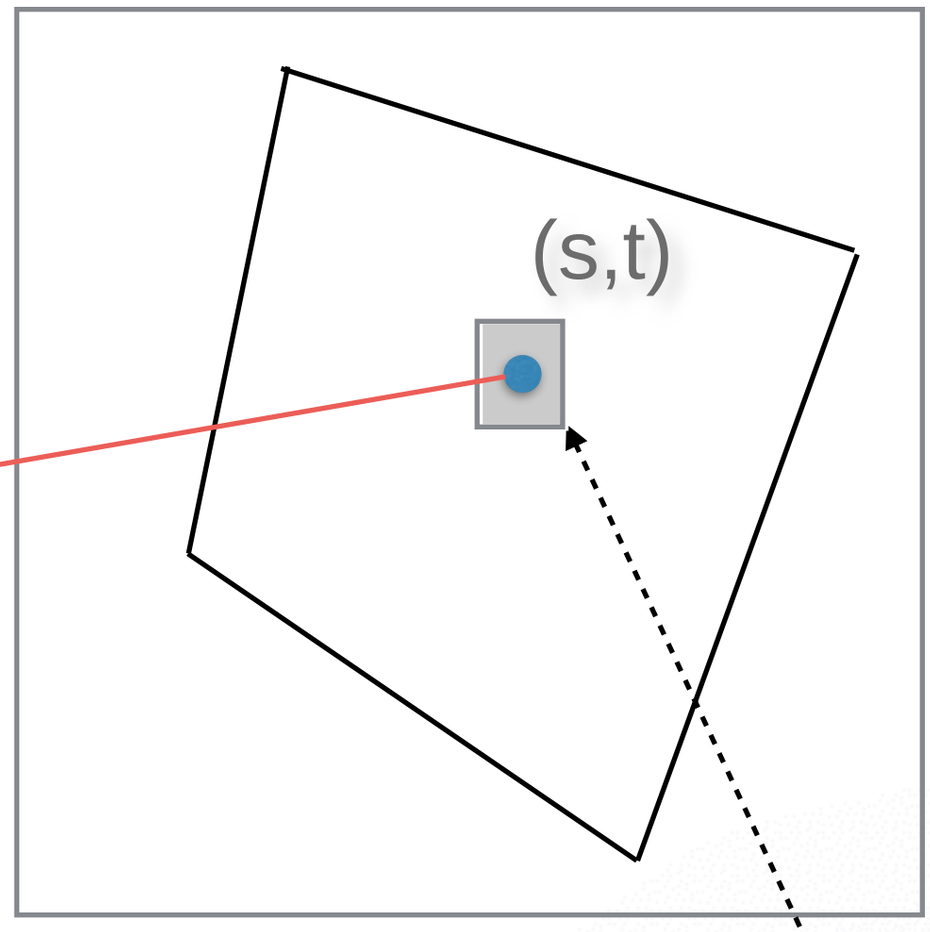
texture image

3D polygon

Inverse texture map



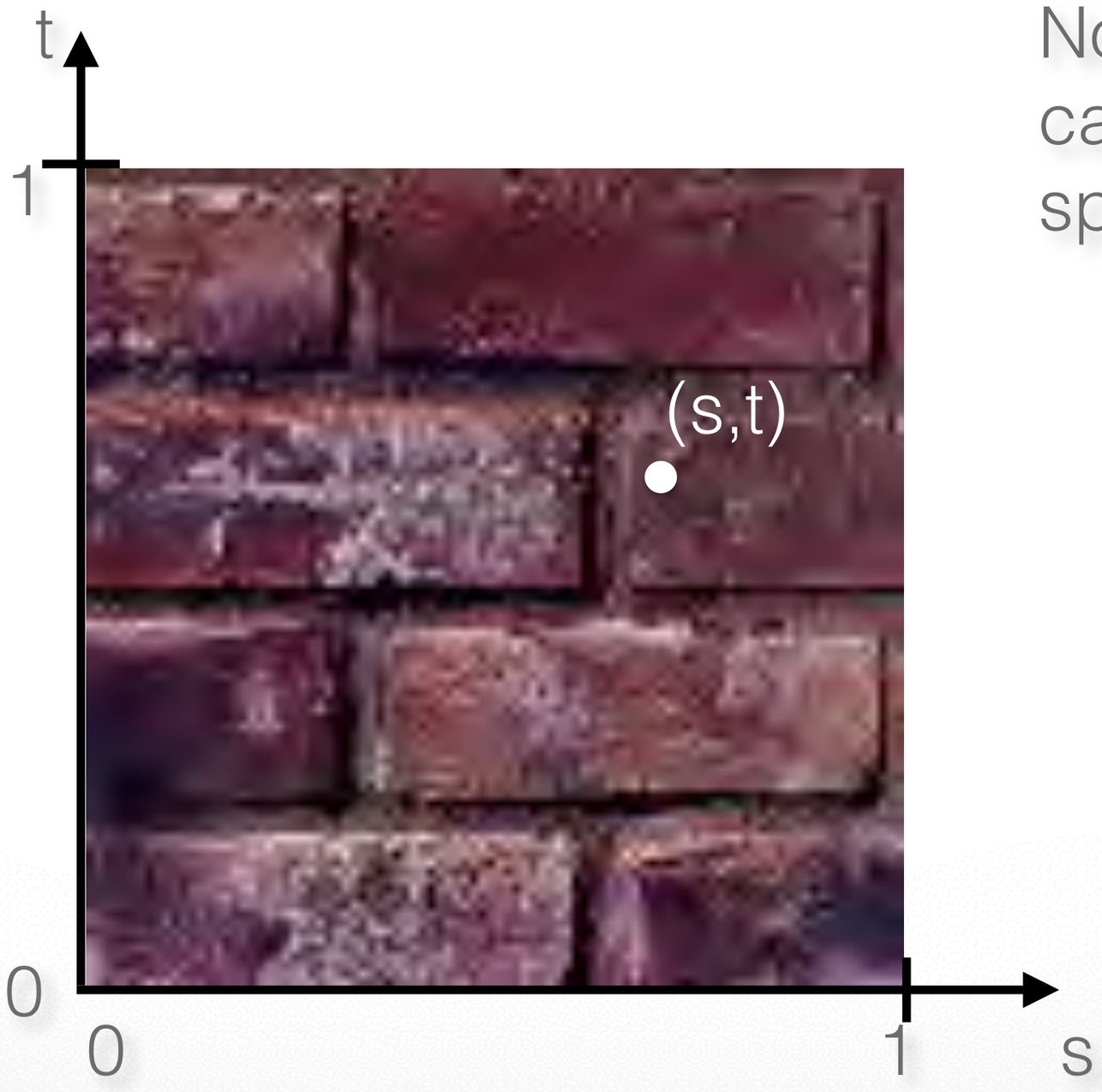
texture image



screen image

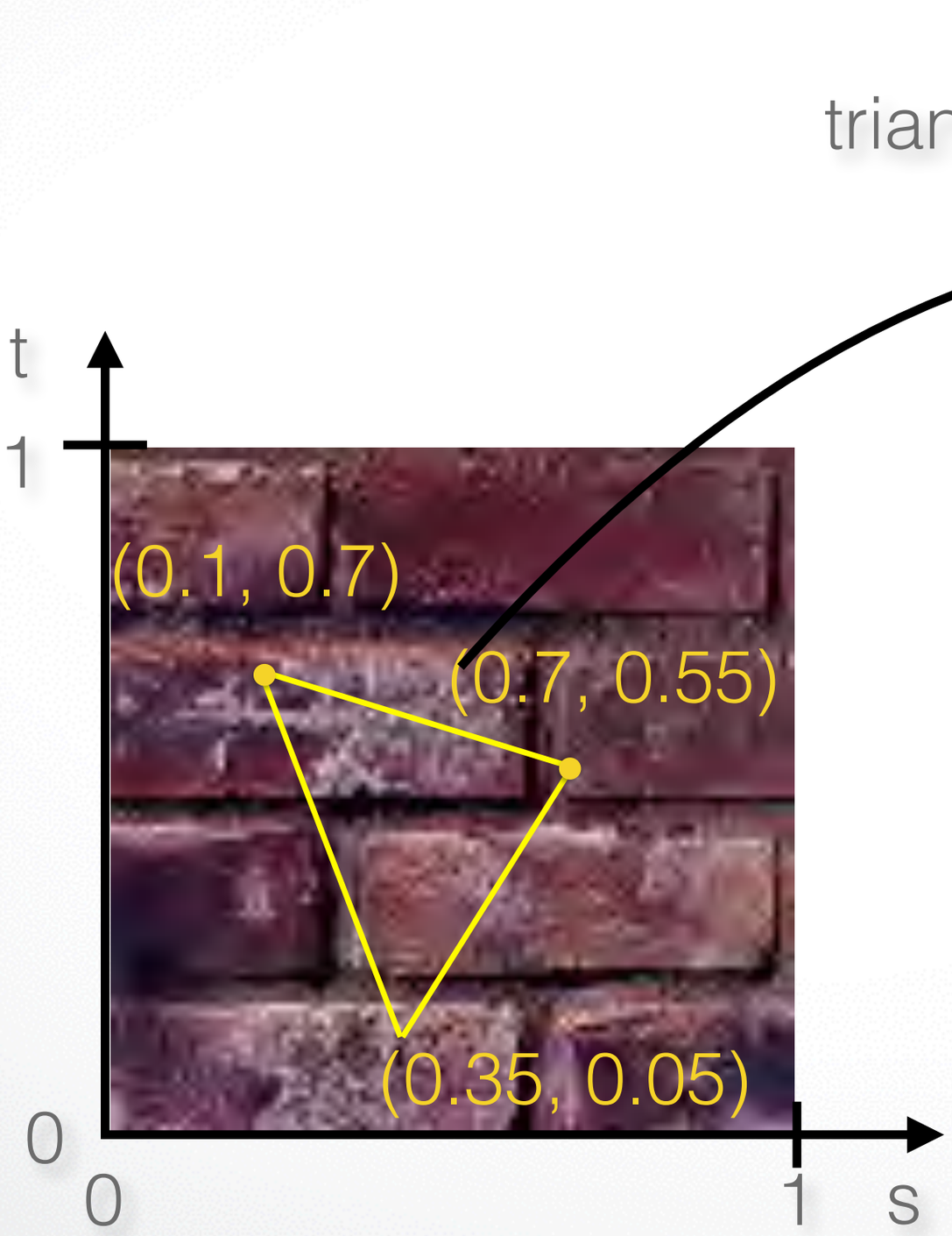
For each pixel, lookup into the texture image to obtain color

The “st” coordinate system

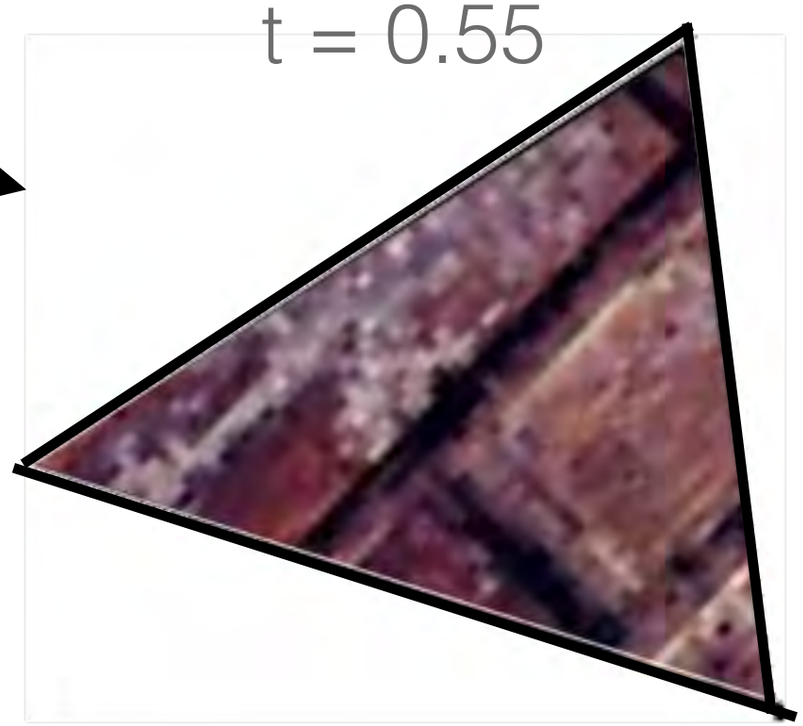


Note: also called “uv” space

Texture mapping: key slide



triangle in 3D



s = 0.7
t = 0.55

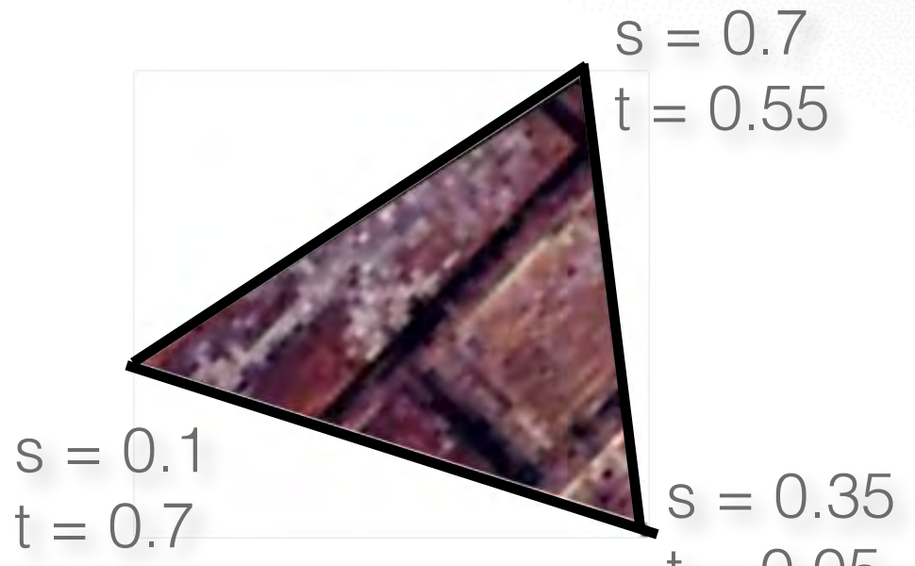
(-2,1,0)

(0,1,0)
s = 0.1
t = 0.7

(2,-1,0)
s = 0.35
t = 0.05

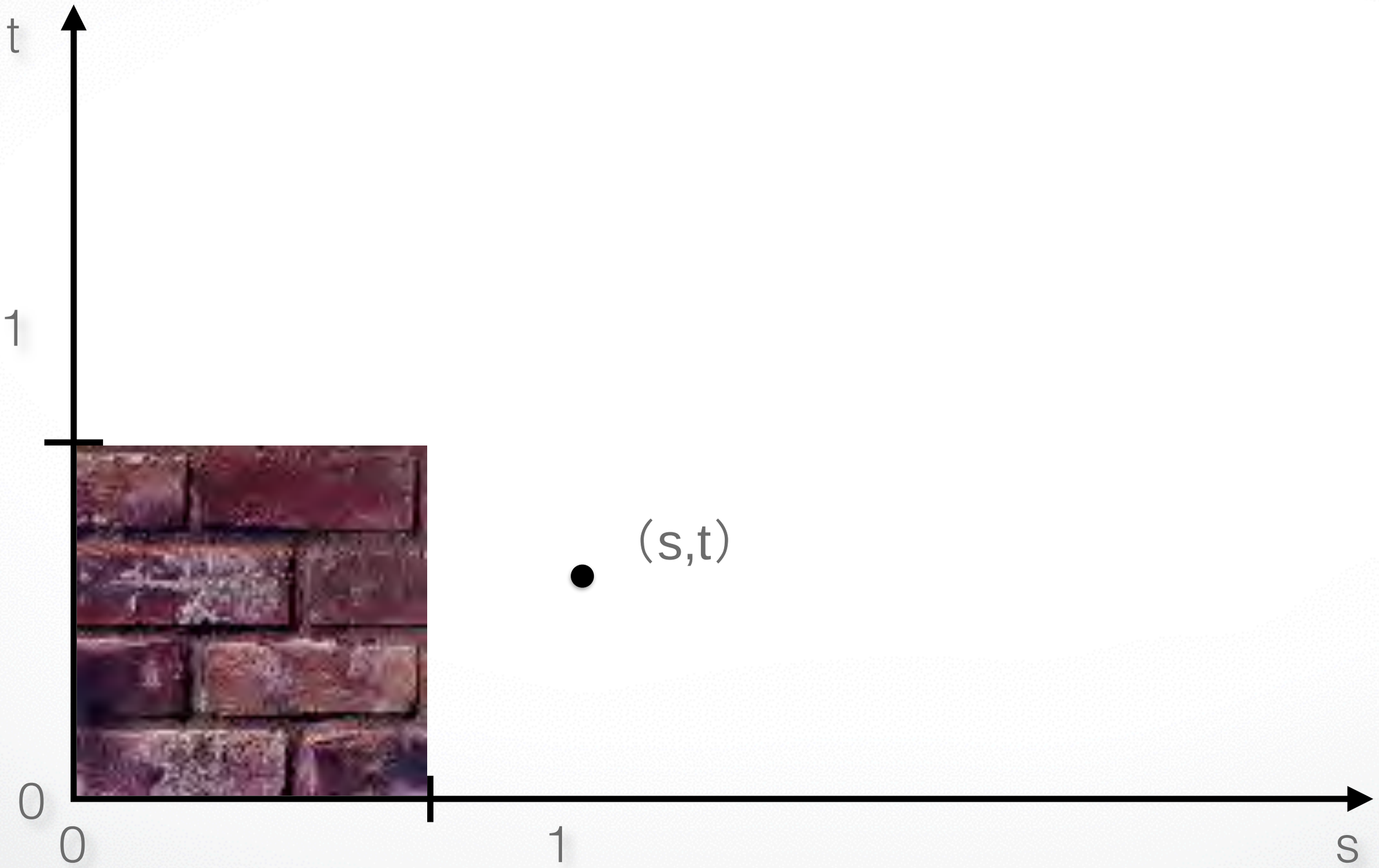
Specifying texture coordinates in OpenGL

- Use glTexCoord2f(s,t)
- State machine: Texture coordinates remain valid until you change them
- Example (from previous slide) :



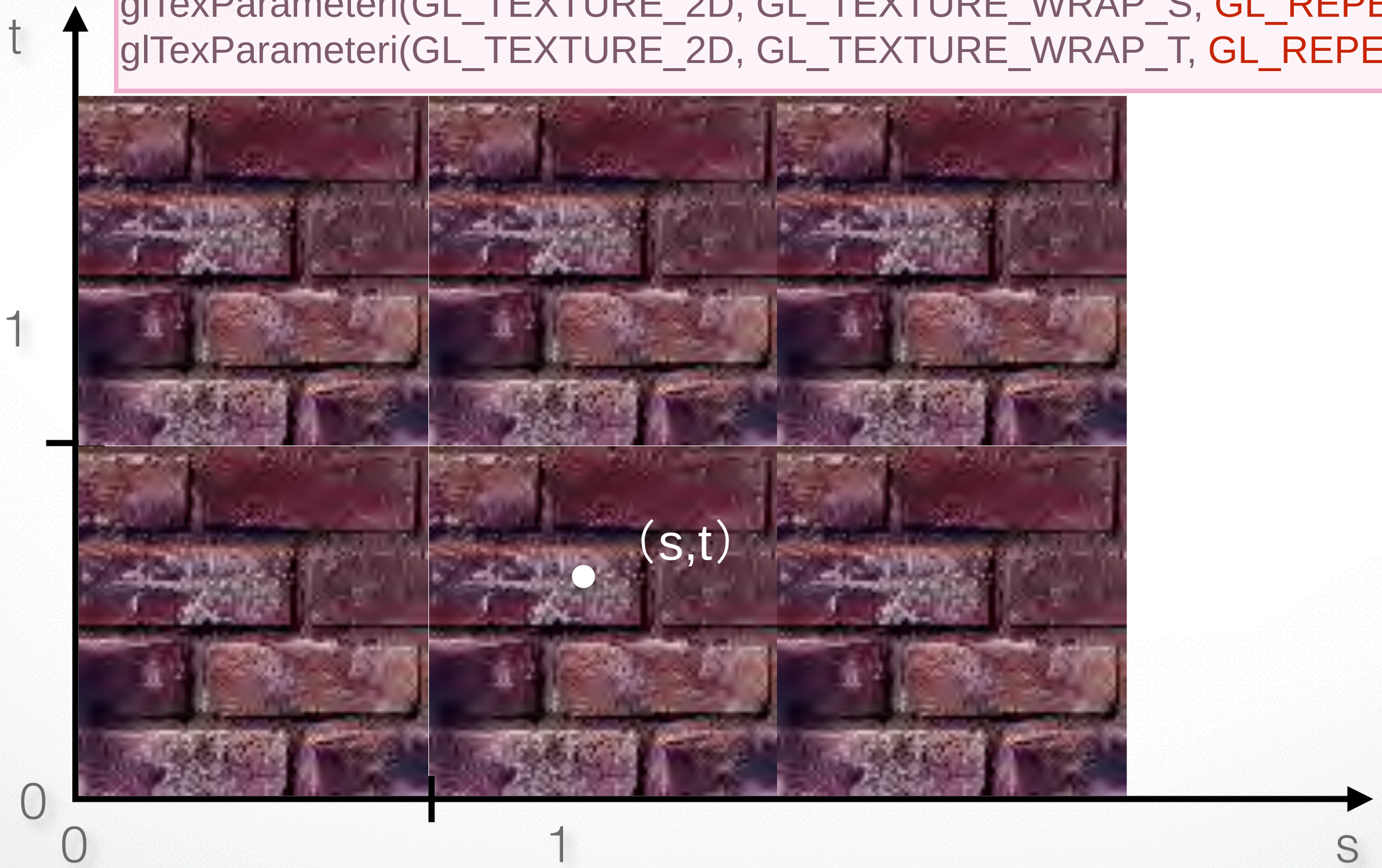

```
glEnable(GL_TEXTURE_2D); // turn texture mapping on
glBegin(GL_TRIANGLES);
  glTexCoord2f(0.35,0.05); glVertex3f(2.0,-1.0,0.0);
  glTexCoord2f(0.7,0.55); glVertex3f(-2.0,1.0,0.0);
  glTexCoord2f(0.1,0.7); glVertex3f(0.0,1.0,0.0);
glEnd();
glDisable(GL_TEXTURE_2D); // turn texture mapping off
```

What if texture coordinates are outside of [0,1] ?



Solution 1: Repeat texture

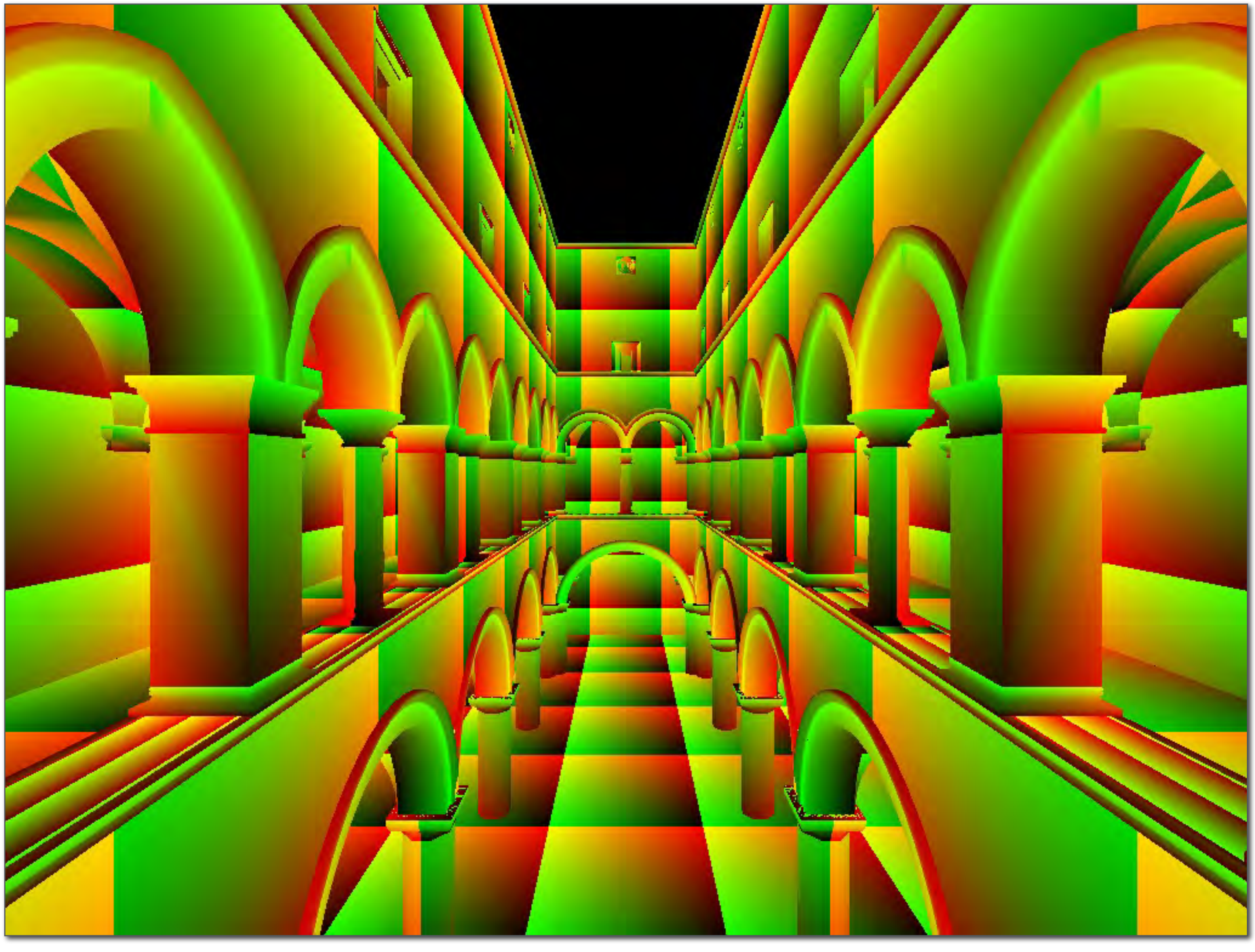
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
```



Solution 2: Clamp to [0,1]

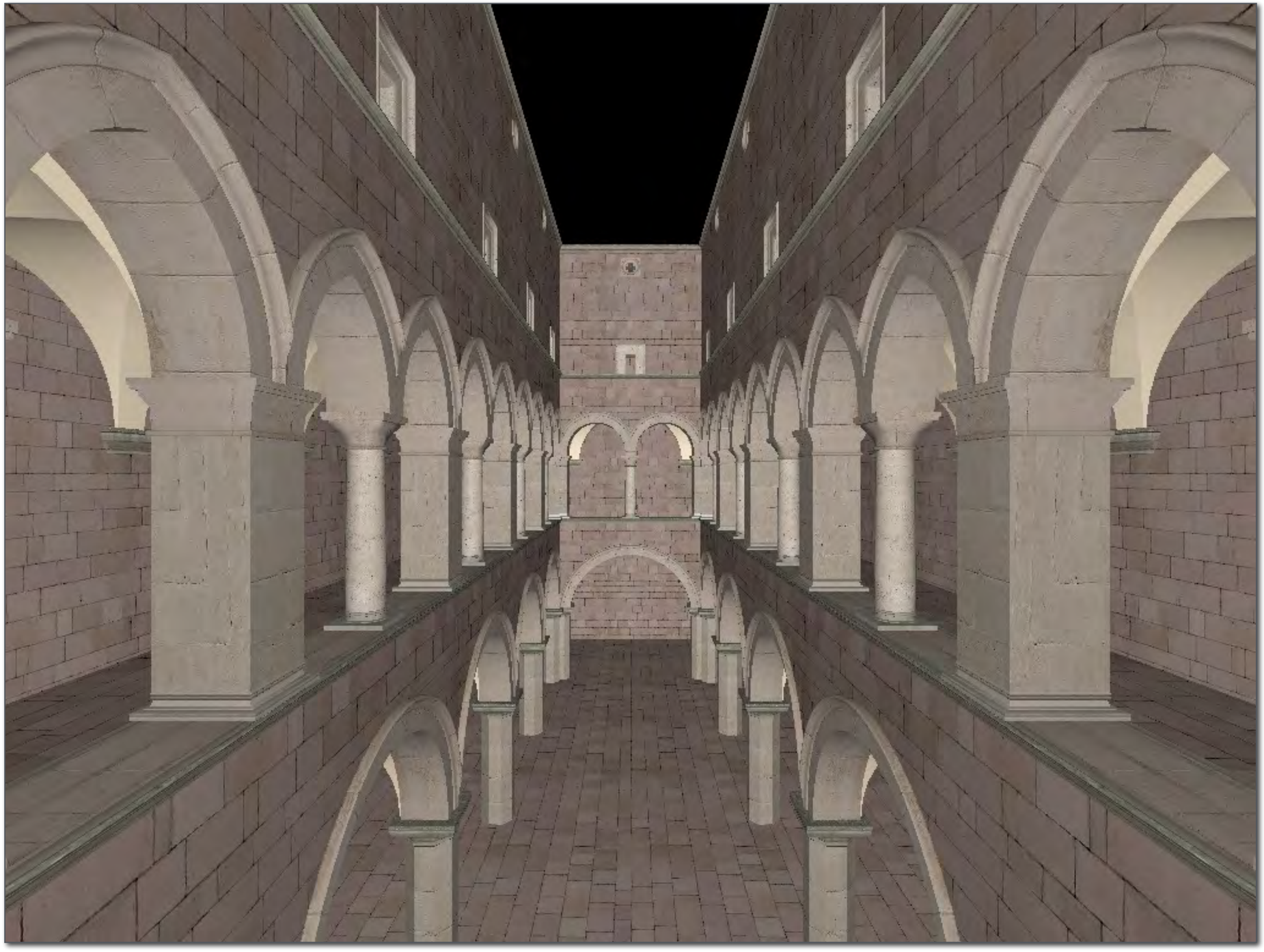
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
```

Another example: Sponza



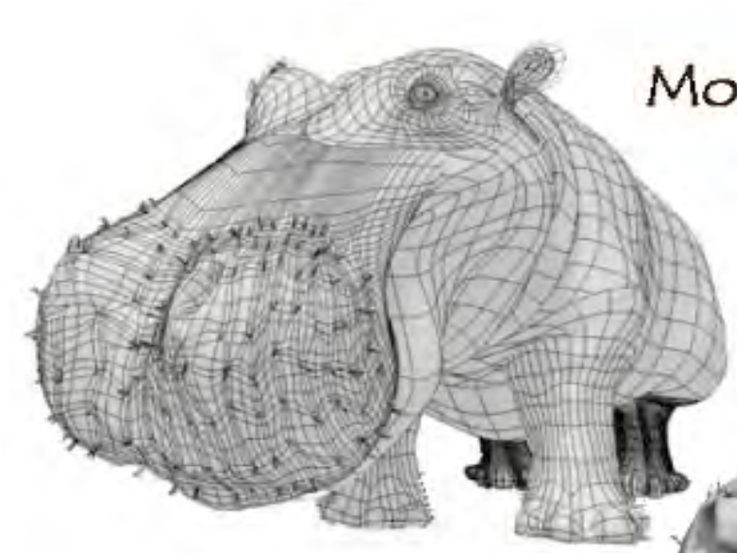
Notice texture coordinates repeat over surface.

Textured Sponza



Example textures used in Sponza

Combining texture mapping and shading



Model

Model + Shading

Model + Shading
+ Textures

At what point
do things start
looking real?

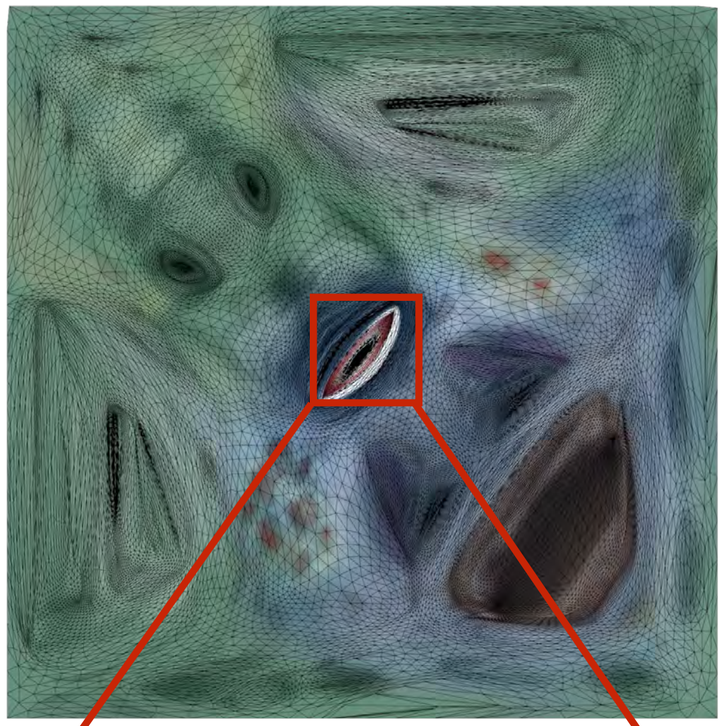
For more info on the computer artwork of Jeremy Birn
see http://www.3drender.com/jbirn/productions.html

Texture mapping adds detail

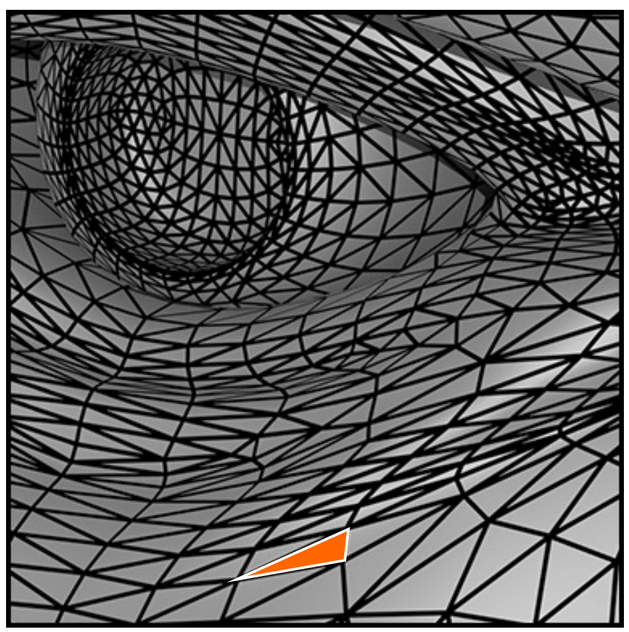
rendering without texture

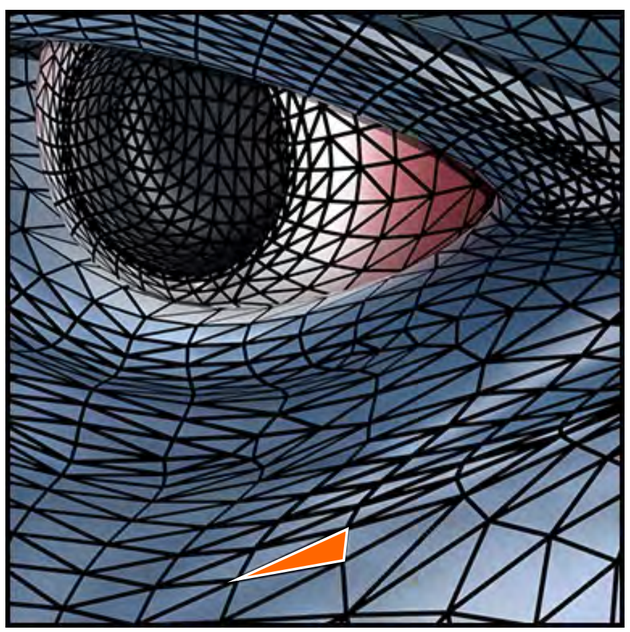
rendering with texture

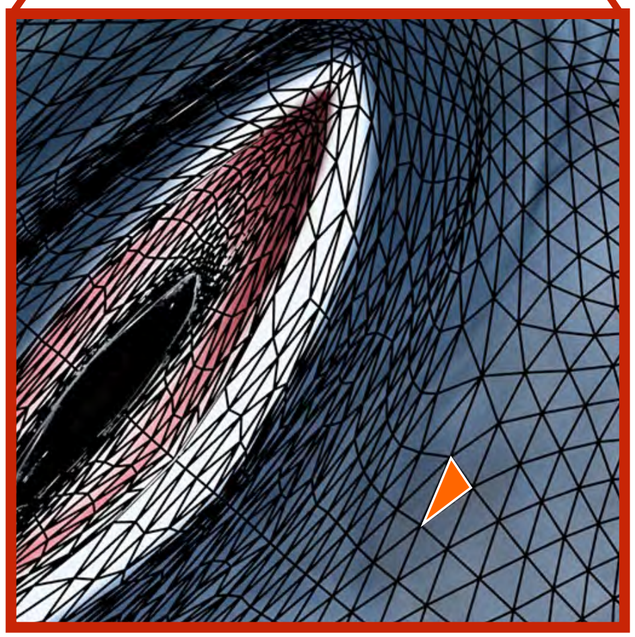
texture image



zoom







Each triangle "copies" a piece of the image back to the surface.

Describe surface material properties

Multiple layers of texture maps for color, logos, scratches, etc.

Combining texture mapping and shading

- Final pixel color = a combination of texture color and color under standard OpenGL Phong lighting
- GL_MODULATE:
  multiply texture and Phong lighting color
- GL_BLEND:
  linear combination of texture and Phong lighting color
- GL_REPLACE:
  use texture color only (ignore Phong lighting)
- Example:

```
glTexEnvf(GL_TEXTURE_ENV,
    GL_TEXTURE_ENV_MODE,   GL_REPLACE);
```

Outline

- Introduction
- Texture mapping in OpenGL
- Filtering and Mipmaps
- Example
- Non-color texture maps

Texture mapping in OpenGL

- **During your initialization:**
  1. Read texture image from file into an array in memory,
     or generate the image using your program
  2. Specify texture mapping parameters
     - Wrapping, filtering, etc.
  3. Initialize and activate the texture

- **In display():**
  1. Enable OpenGL texture mapping
  2. Draw objects: Assign texture coordinates to vertices
  3. Disable OpenGL texture mapping

Initializing the texture

- Do once during initialization, for each texture image in the scene, by calling glTexImage2D

- The dimensions of texture images must be powers of 2
  - if not, rescale image or pad with zero
  - or can use OpenGL extensions

- Can load textures dynamically if GPU memory is scarce

glTexImage2D

- glTexImage2D(GL_TEXTURE_2D, level, internalFormat, width, height, border, format, type, data)
- GL_TEXTURE_2D: specifies that it is a 2D texture
- Level: used for specifying levels of detail for mipmapping (default:0)
- InternalFormat
  - Often: GL_RGB or GL_RGBA
  - Determines how the texture is stored internally
- Width, Height
  - The size of the texture must be powers of 2
- Border (often set to 0)
- Format, Type
  - Specifies what the input data is (GL_RGB, GL_RGBA, …)
  - Specifies the input data type (GL_UNSIGNED_BYTE, GL_BYTE, …)
  - Regardless of Format and Type, OpenGL converts the data to internalFormat
- Data: pointer to the image buffer

Enable/disable texture mode

- Must be done before rendering any primitives that are to be texture-mapped
  glEnable(GL_TEXTURE_2D)
  glDisable(GL_TEXTURE_2D)

- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons

- Changing textures:
  - Only one texture is active at any given time
    (with OpenGL extensions, more than one can be used simultaneously; this is called *multitexturing*)
  - Use glBindTexture to select the active texture

Outline

- Introduction
- Texture mapping in OpenGL
- Filtering and Mipmaps
- Example
- Non-color texture maps

Texture interpolation

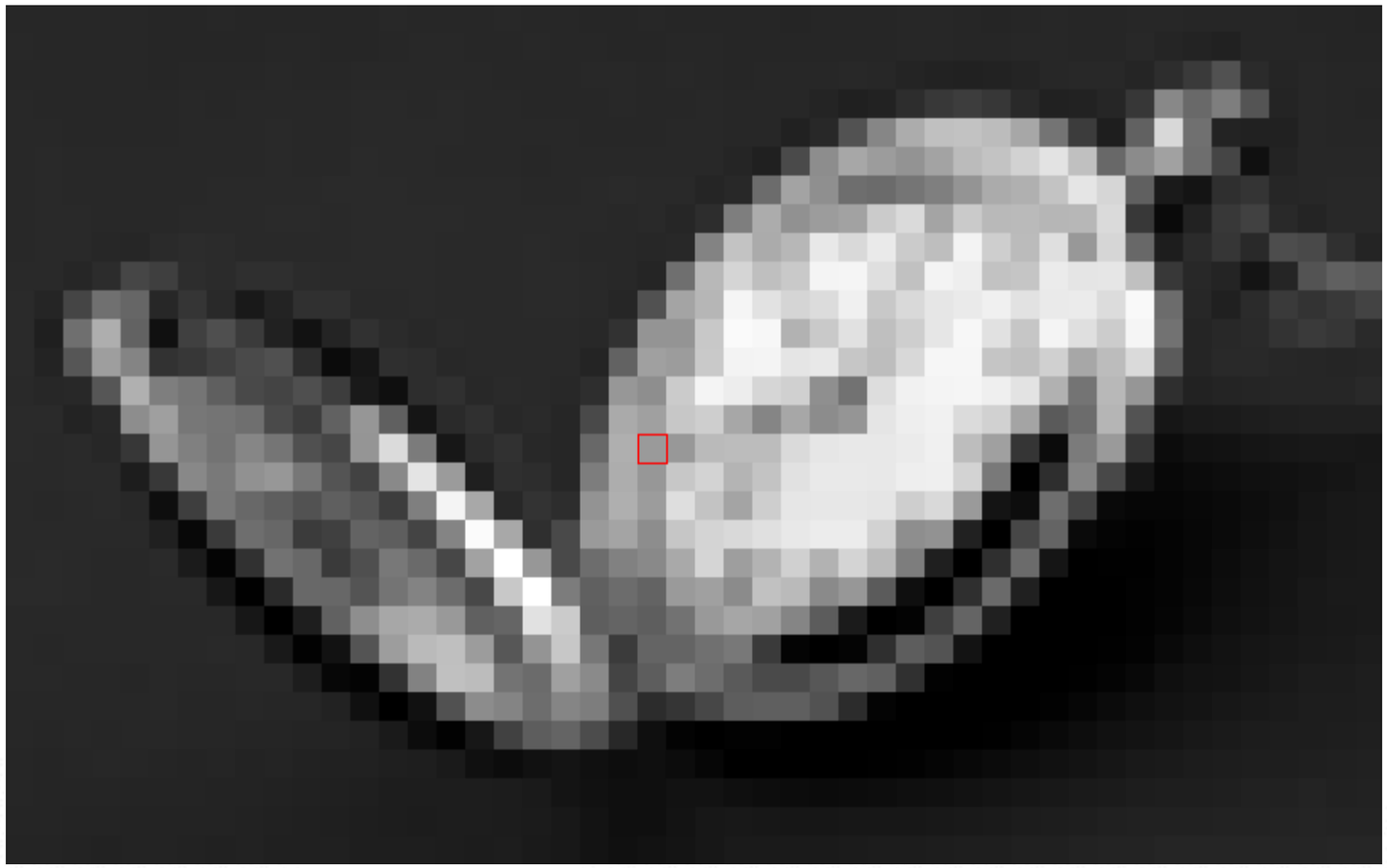
- This photo is too small

Zooming

- First consider a black and white image

- We want to blow it up to poster size (zoom by a factor of 16)
- Firs try: repeat each row 16 times, then each column 16 times

Zooming: Nearest Neighbor Interpolation



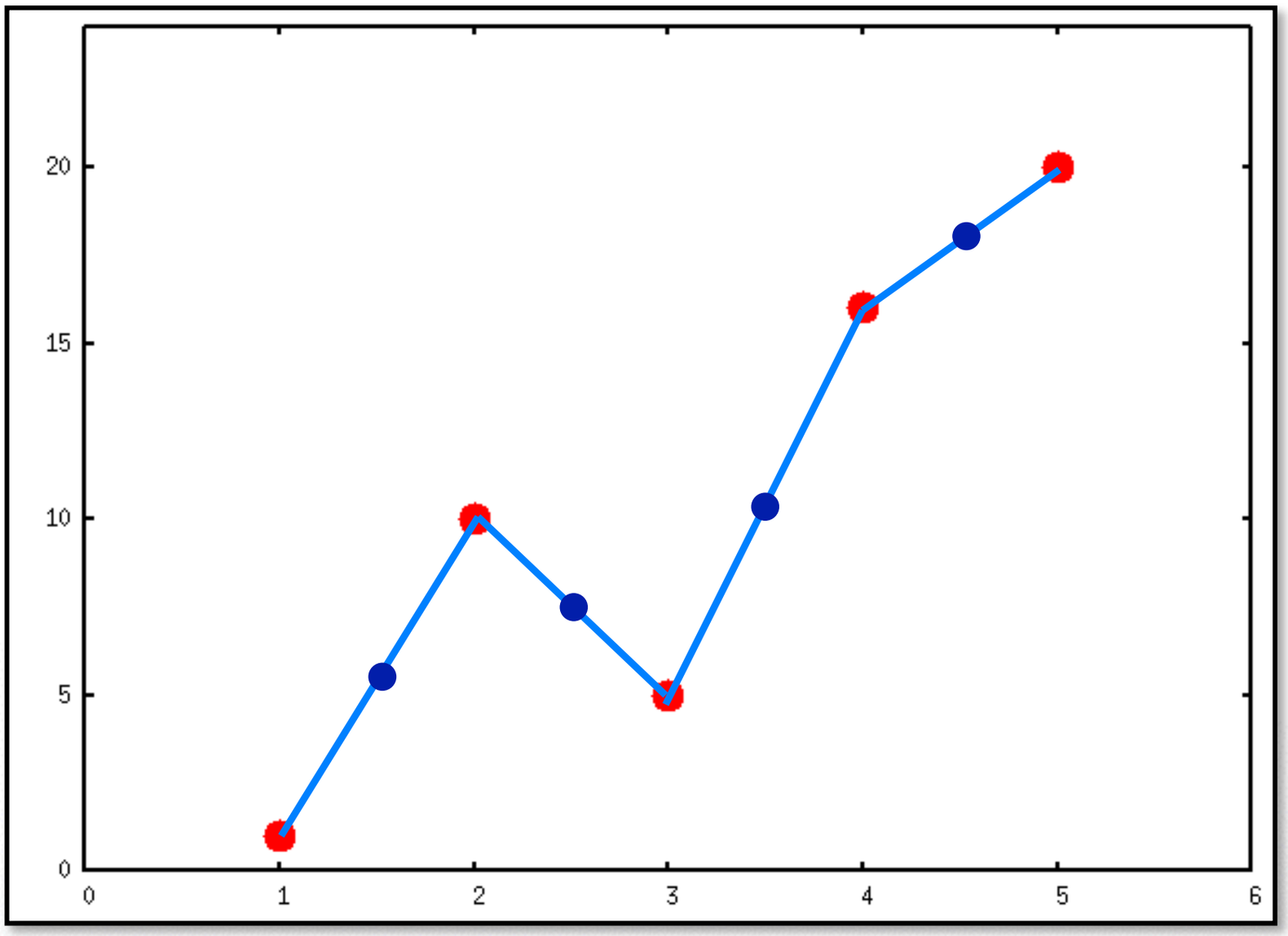
Zooming: First Attempt

- That didn't work so well
- We need a better way to find the in between values
- Let's consider one horizontal slice through the image (one scanline)

Interpolation

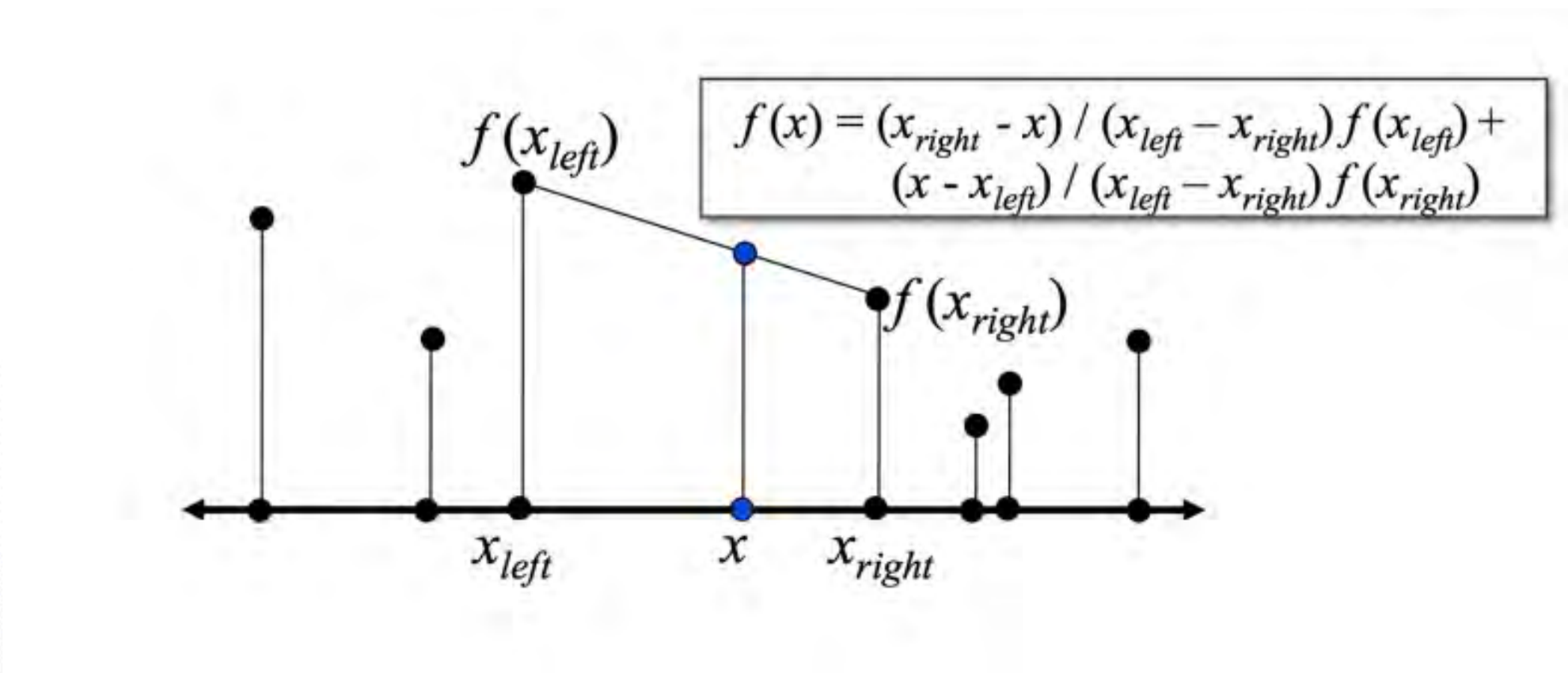
- Problem statement:
  - Given the values of a function f at a few locations, e.g. f(1), f(2), f(3), …
  - Find the rest of the values: what is f(1.5)?
- This is called **Interpolation**
- We need some models that predicts how the function behaves

Linear Interpolation (LERP)



Linear Interpolation (LERP)

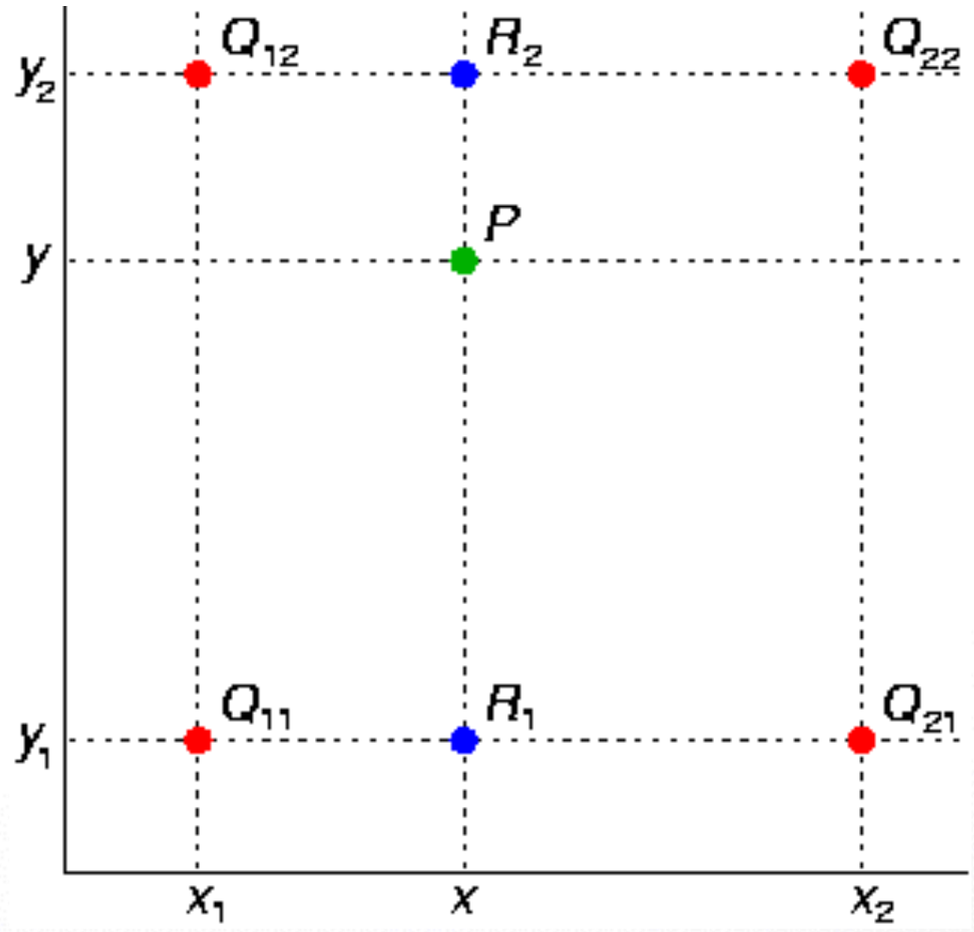
- To compute f(x), find the two points x_{left} and x_{right} that x lies between



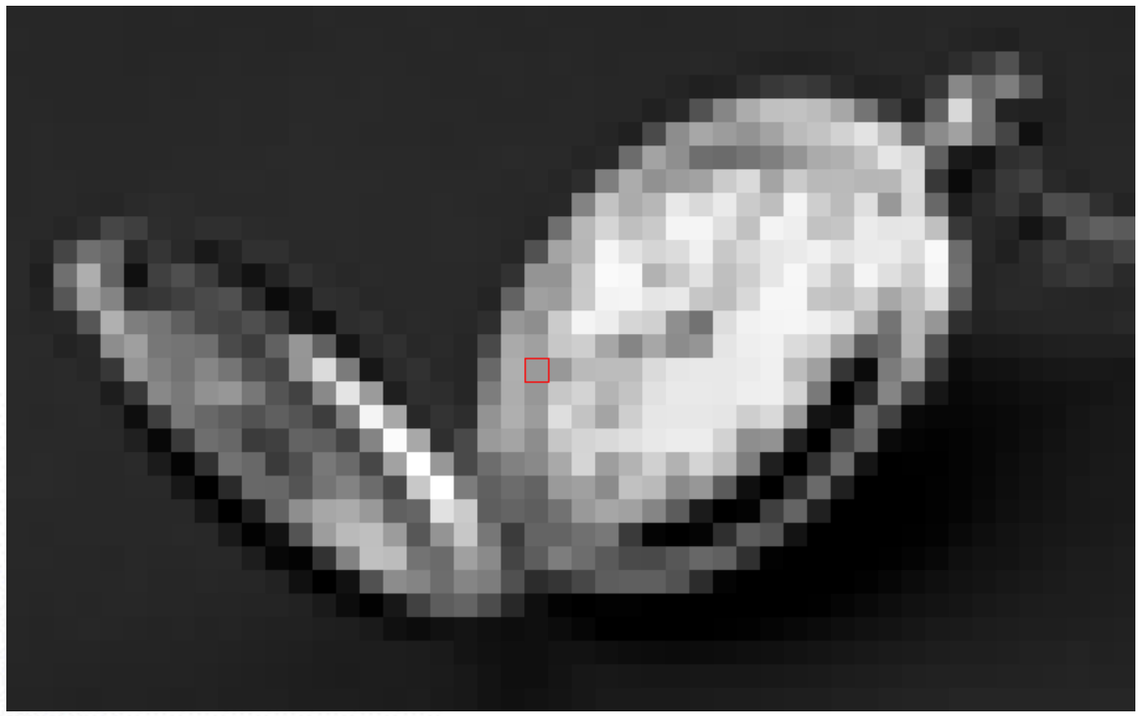
Bilinear Interpolation (in 2D)

- Interpolate in x then in y

$$
\begin{aligned}
f(x, y) \approx & \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) \\
& + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) \\
& + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) \\
& + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).
\end{aligned}
$$



Comparison

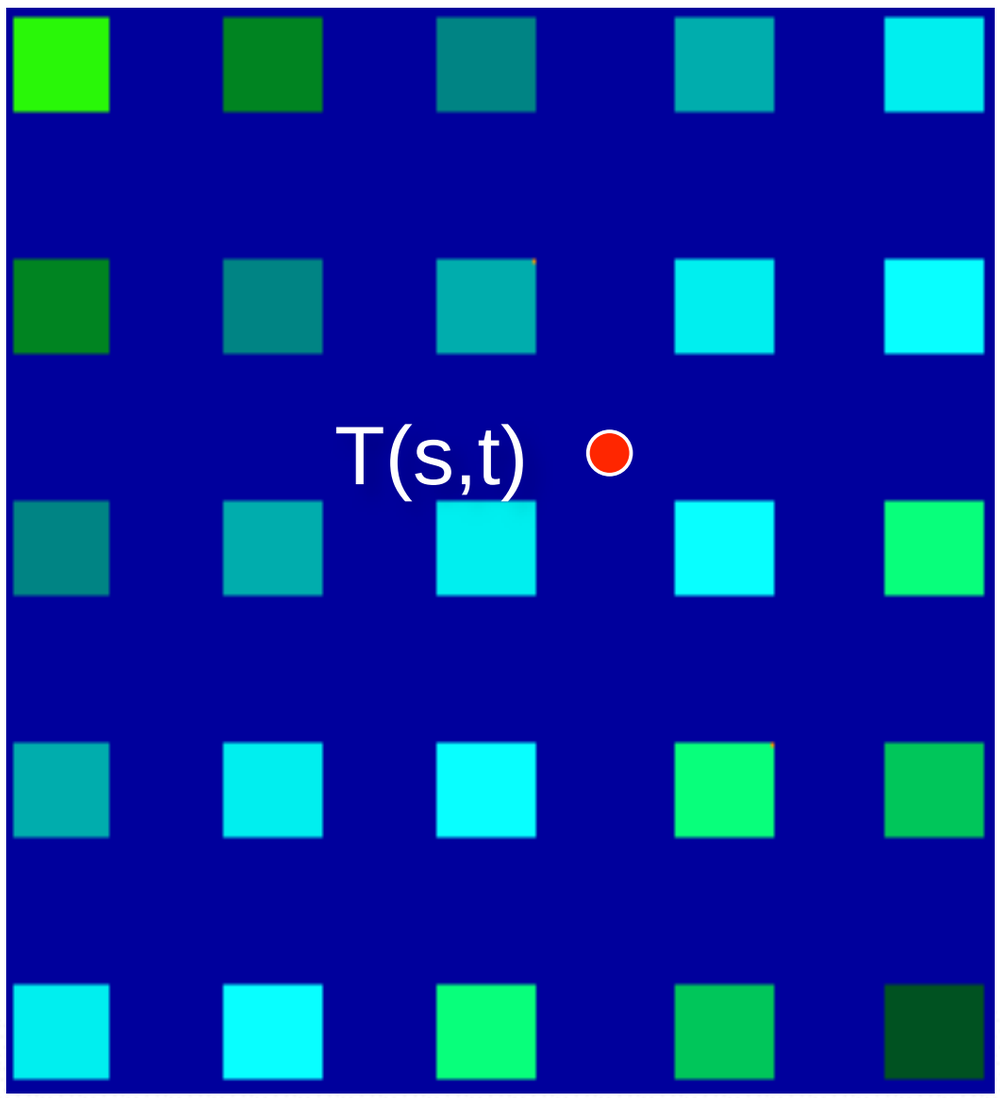


Nearest Neighbor

Bilinear

Texture interpolation

5 x 5 texture



(s,t) coordinates typically not directly at pixel in the texture, but in between

Texture Interpolation in OpenGL

- (s,t) coordinates typically not directly at pixel in the texture, but in between
- Solutions:
  - Use the nearest neighbor to determine color
    - Faster, but worse quality

      glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER, GL_NEAREST)
  - Linear interpolation
    - Incorporate colors of several neighbors to determine color
    - Slower, better quality

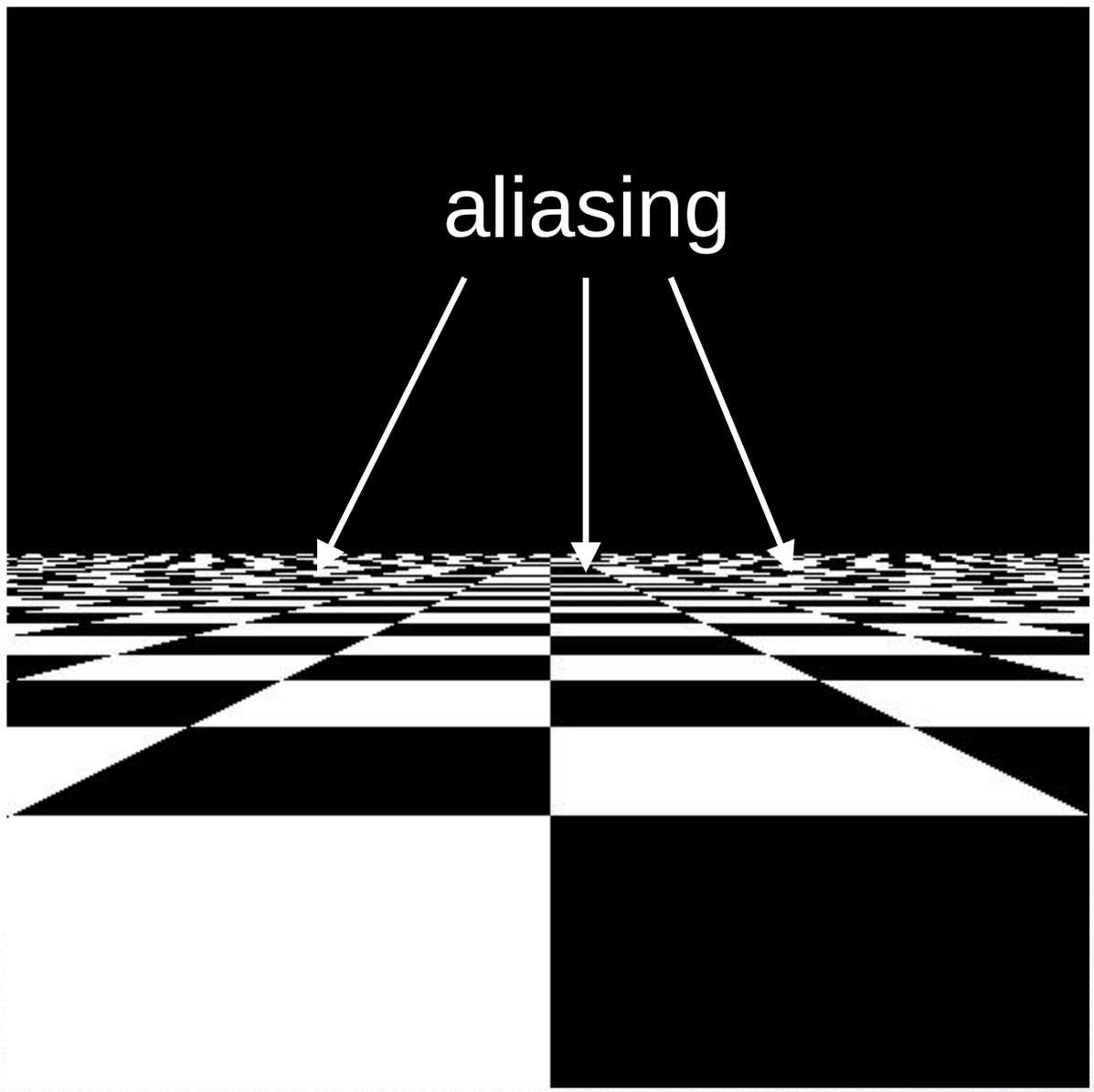
      glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER, GL_LINEAR)

Filtering

- Texture image is shrunk in distant parts of the image

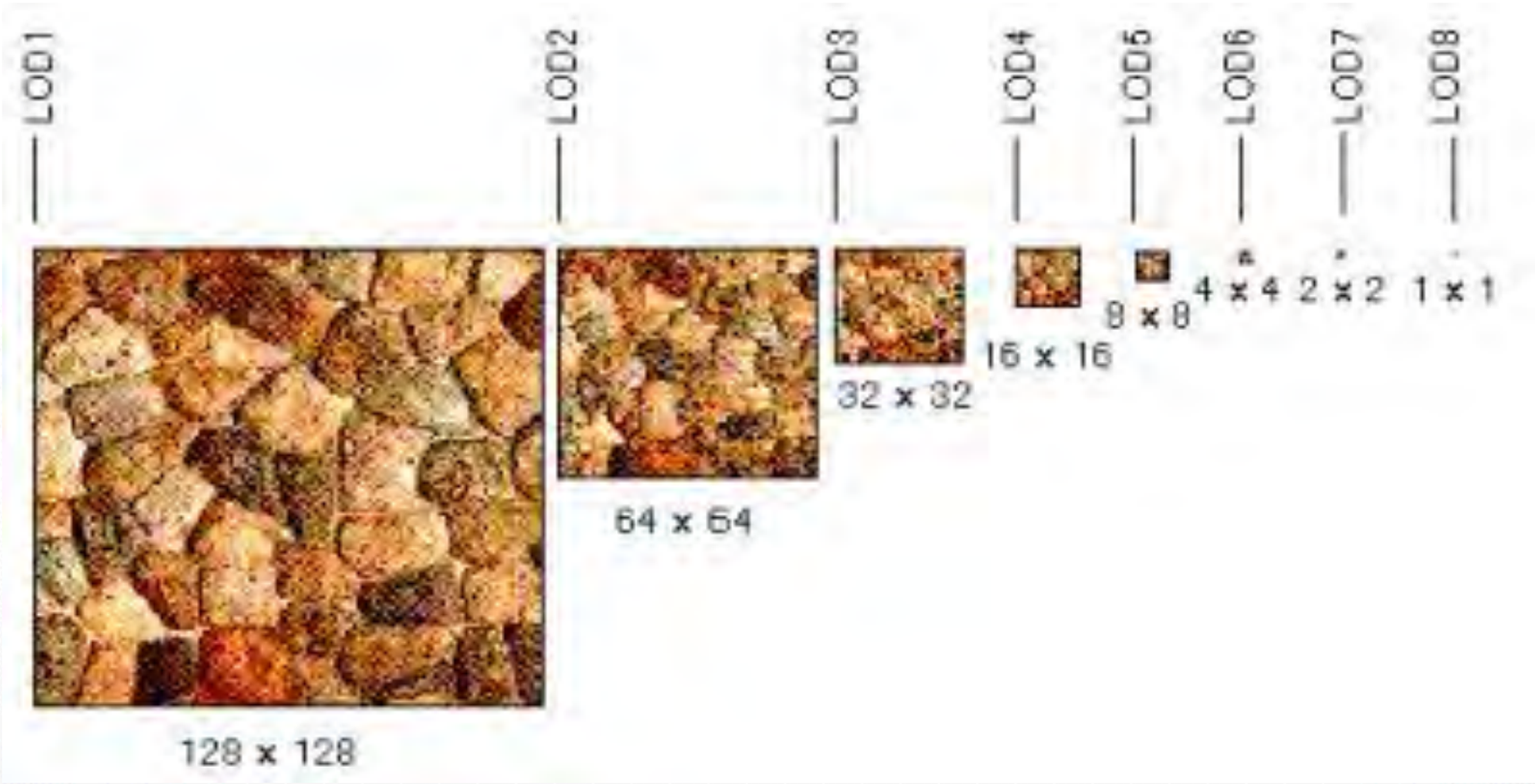
- This leads to aliasing

- Can be fixed with *filtering*
  - bilinear in space
  - trilinear in space and level of detail (mipmapping)



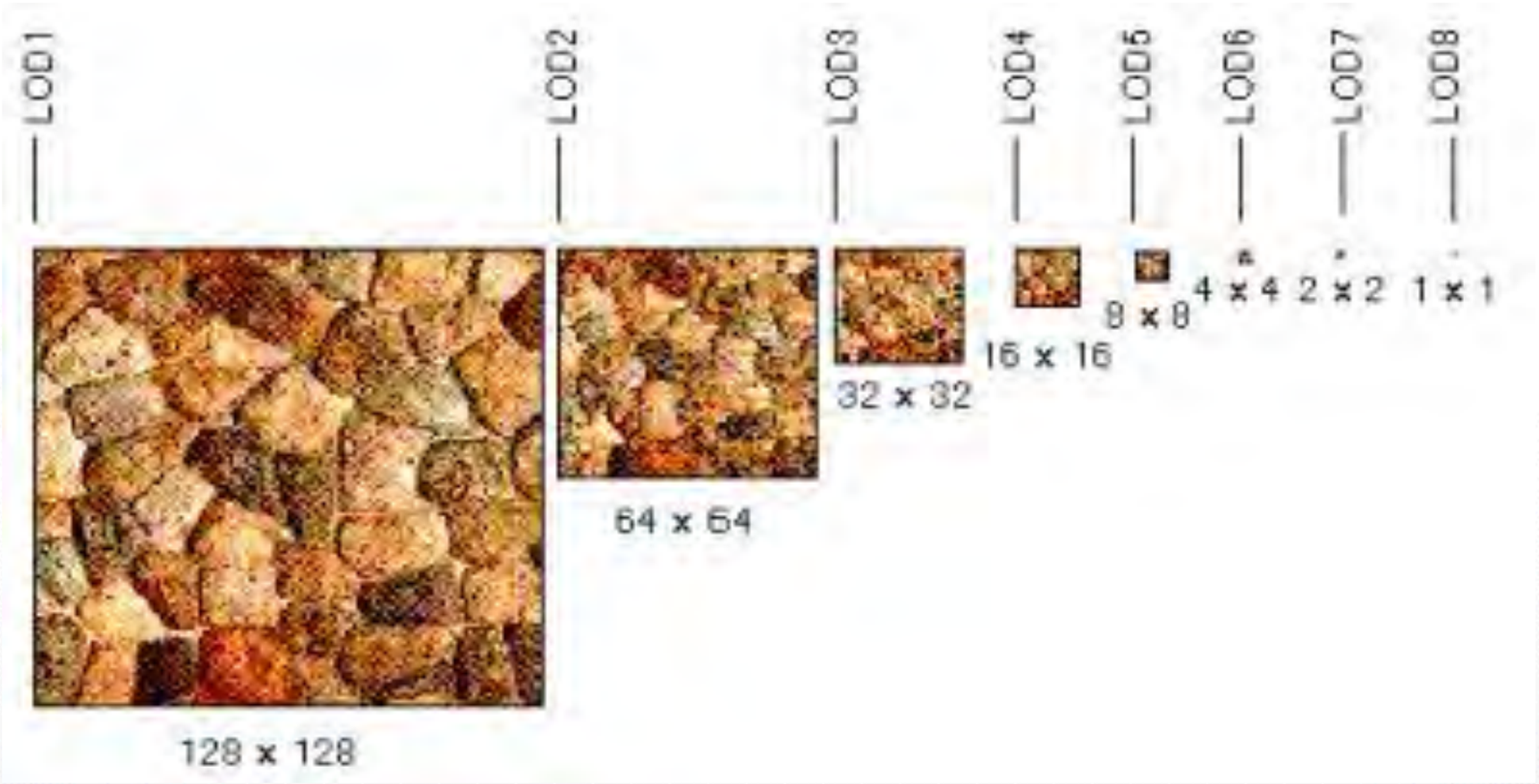
Mipmapping

- Pre-calculate how the texture should look at various distances, then use the appropriate texture at each distance
- Reduces / fixes the aliasing problem



Mipmapping

- Each mipmap (each image below) represents a level of depth (LOD).
- Powers of 2 make things much easier.



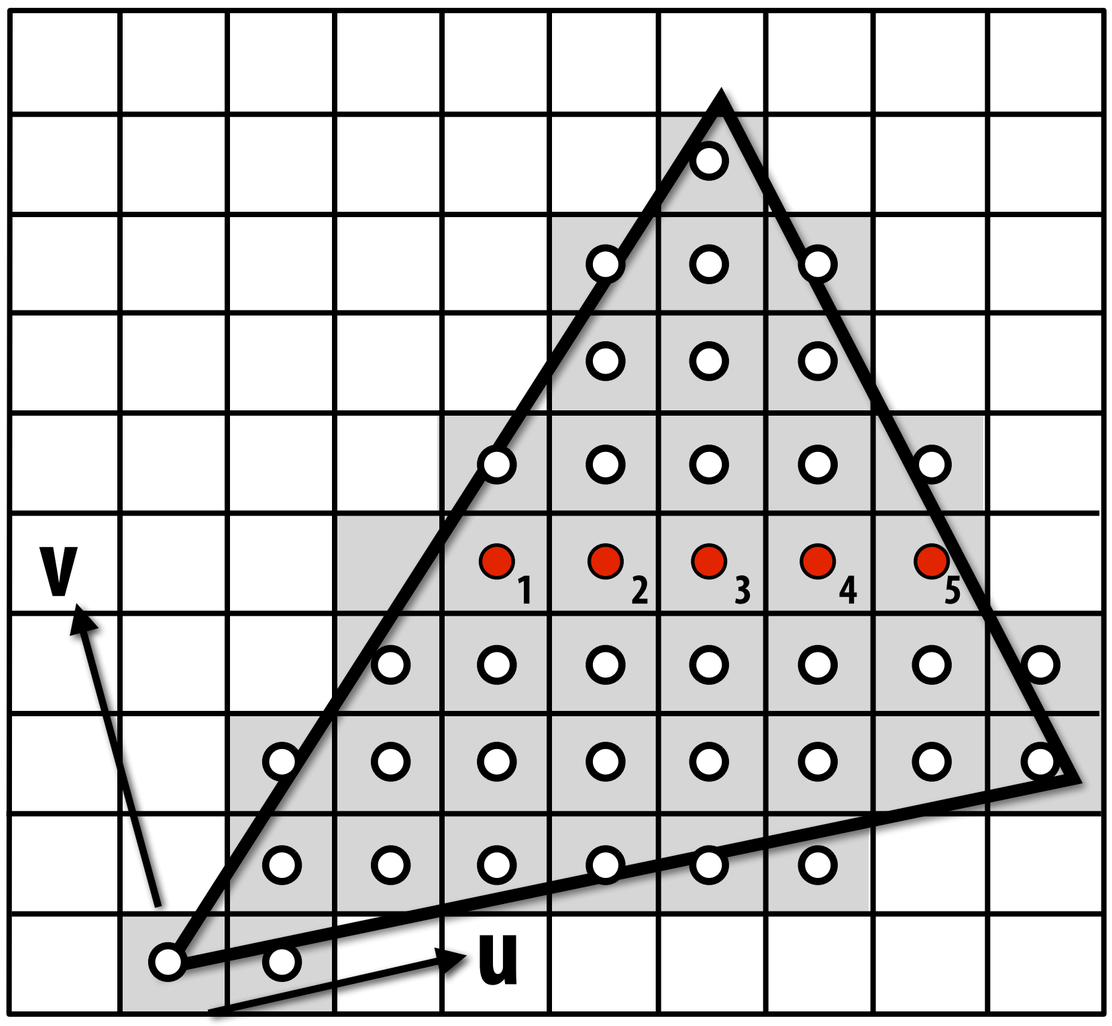
Mipmapping in OpenGL

- gluBuild2DMipmaps(GL_TEXTURE_2D, components, width, height, format, type, data)
  - This will generate all the mipmaps automatically

- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST)
  - This will tell GL to use the mipmaps for the texture

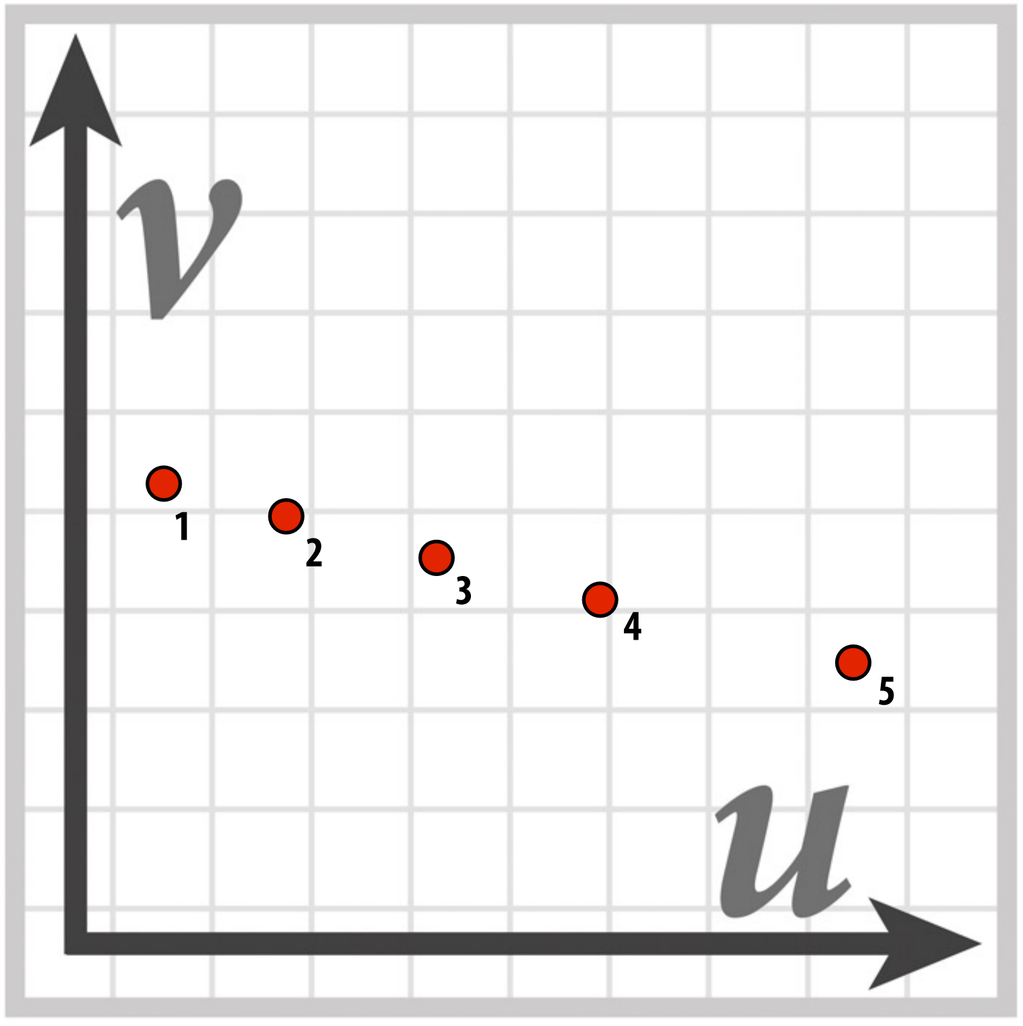
Texture space samples

Sample positions in XY screen space



Sample positions are uniformly distributed in screen space (rasterizer samples triangle's appearance at these locations)

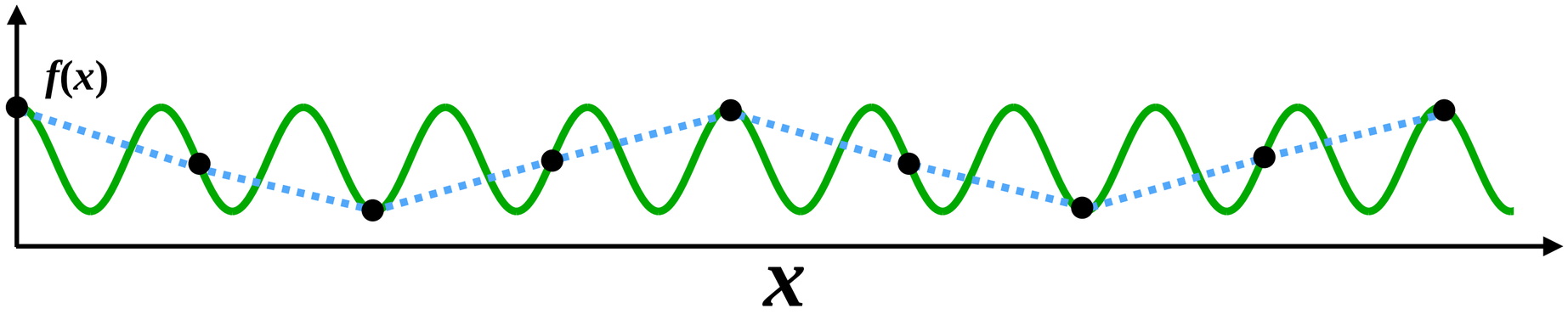
Sample positions in texture space



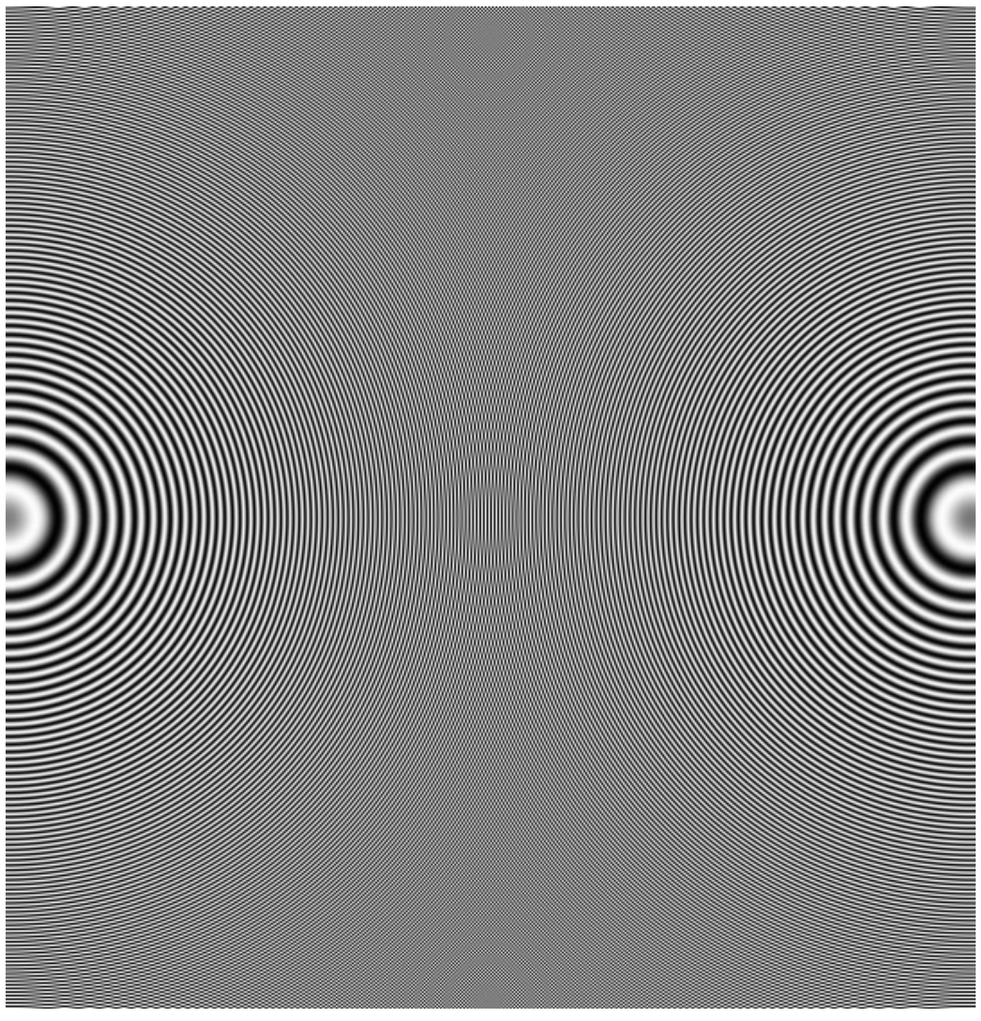
Texture sample positions in texture space (texture function is sampled at these locations)

Recall: aliasing

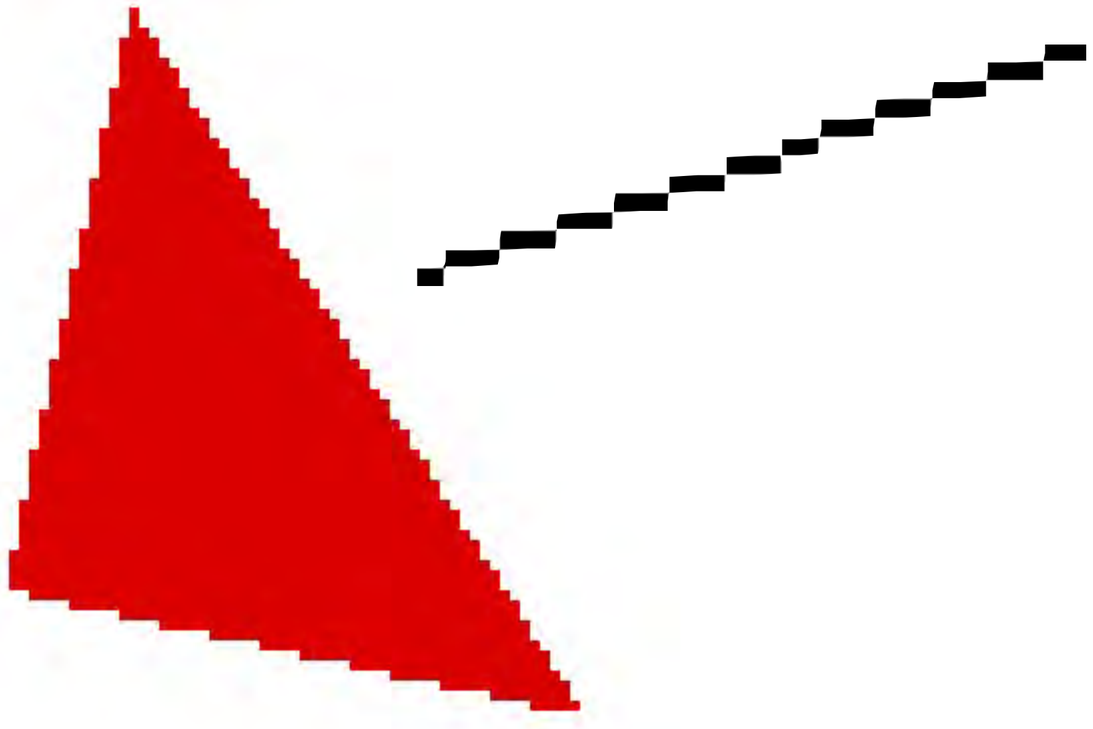
Undersampling a high-frequency signal can result in aliasing



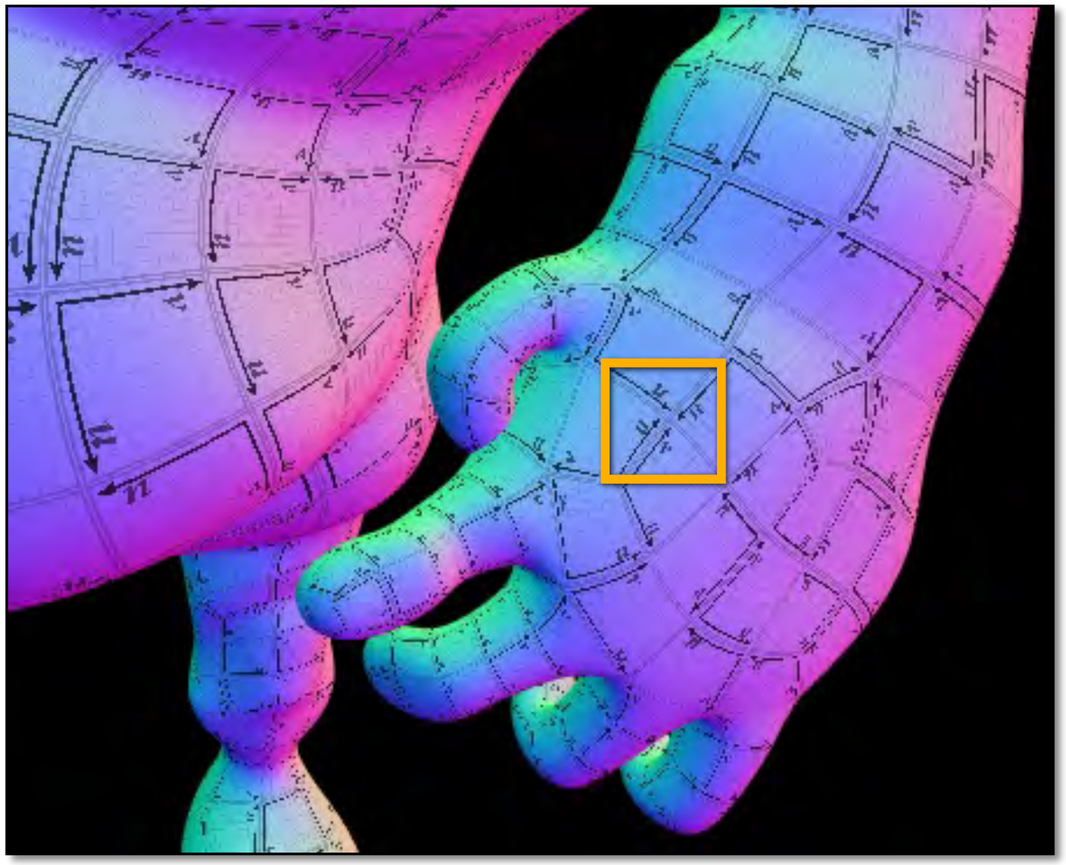
1D example



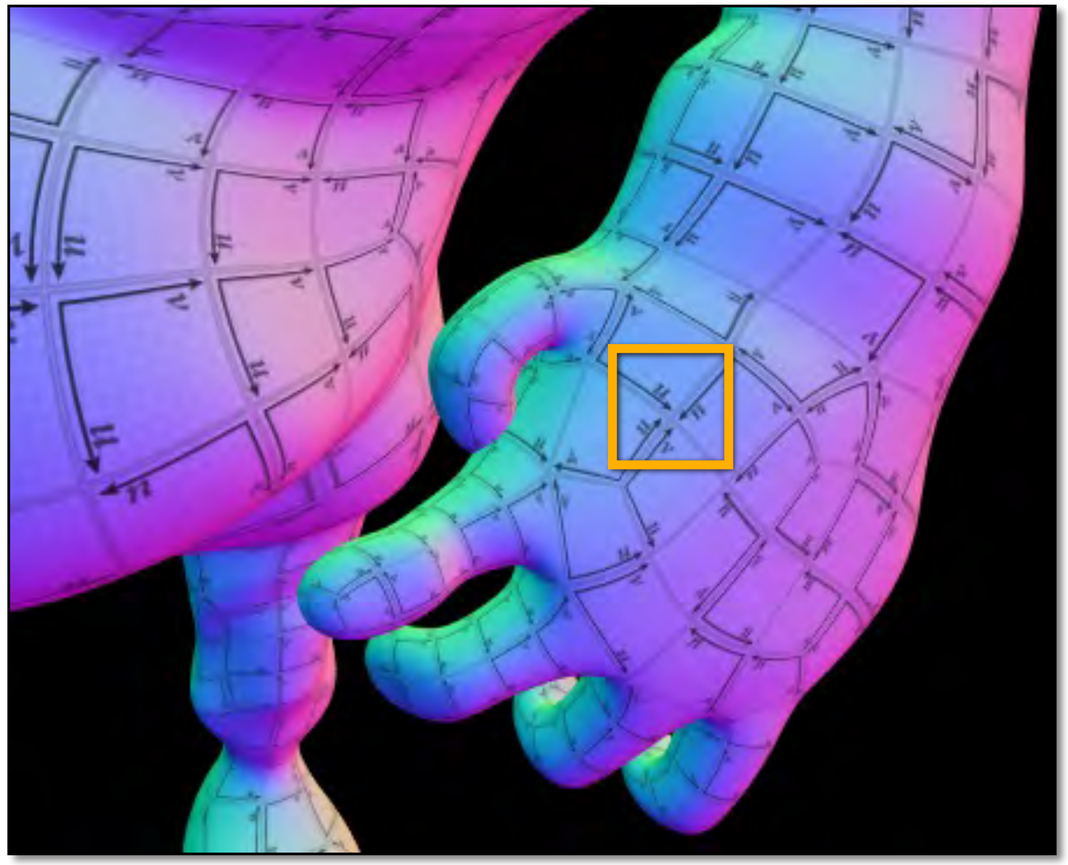
2D examples:
Moiré patterns, jaggies



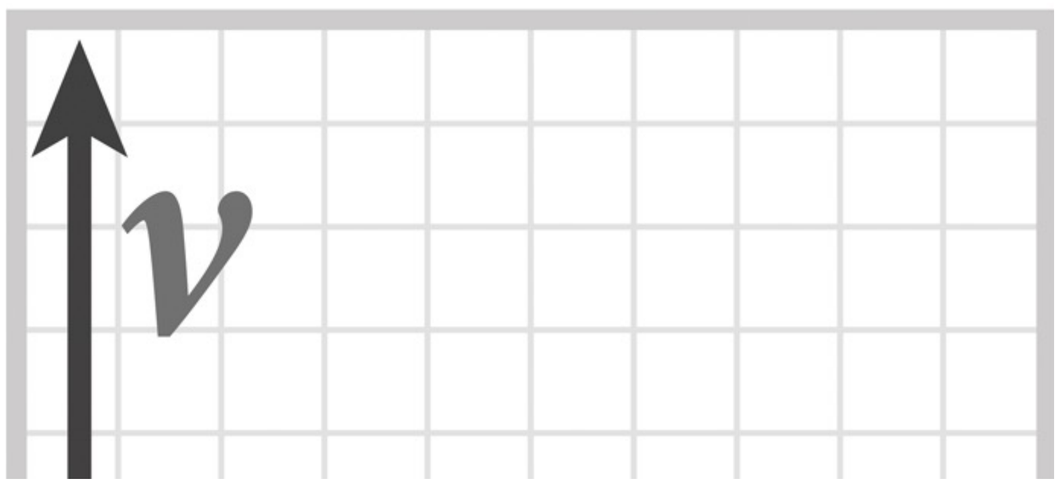
Aliasing due to undersampling texture



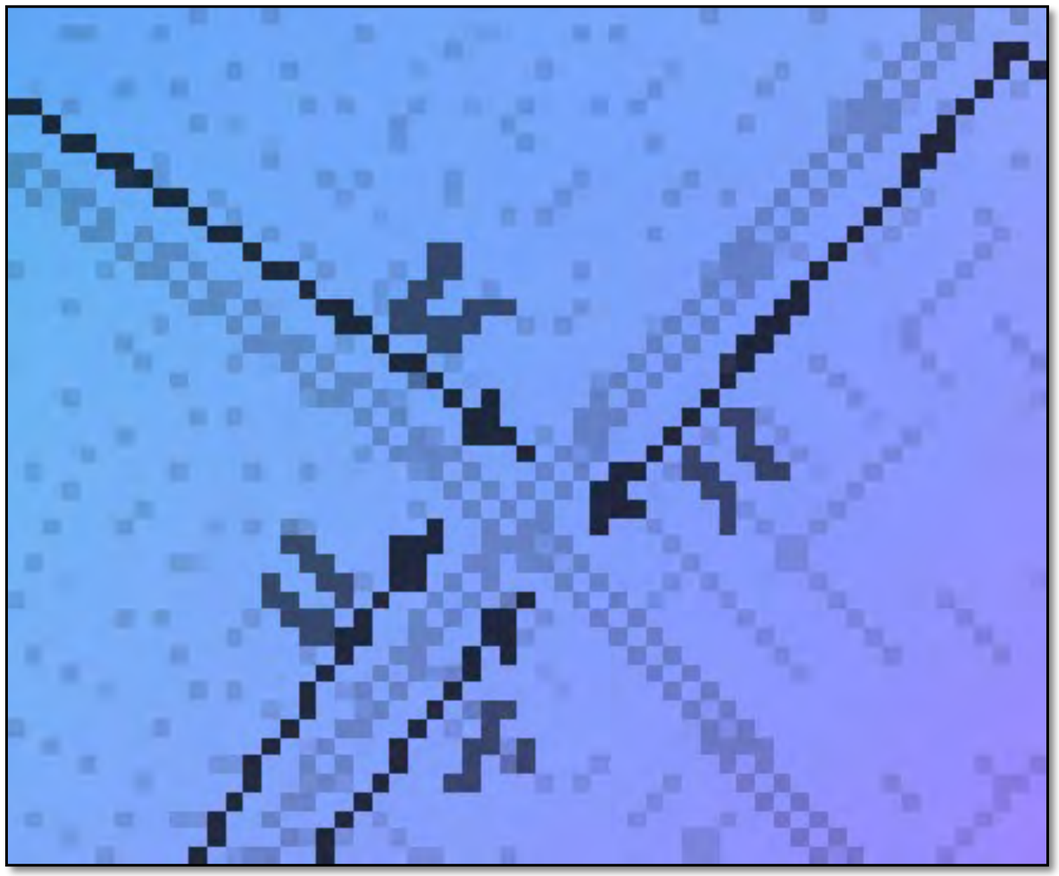
No pre-filtering of texture data
(resulting image exhibits aliasing)



Rendering using pre-filtered texture data

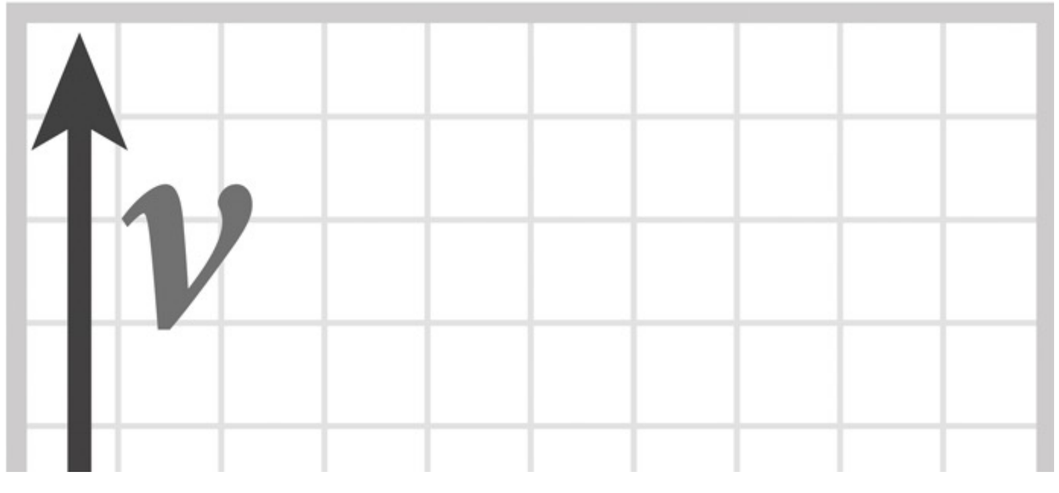


Aliasing due to undersampling (zoom)

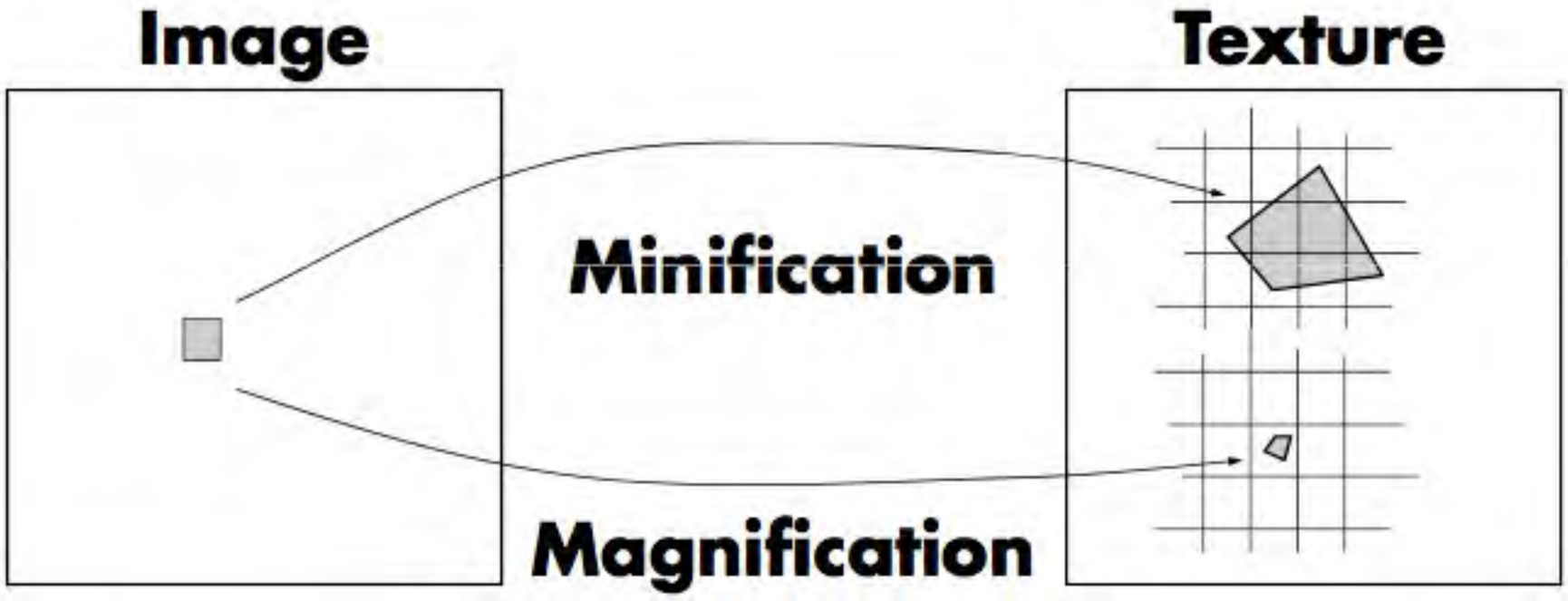


No pre-filtering of texture data
(resulting image exhibits aliasing)

Rendering using pre-filtered texture data



Filtering textures

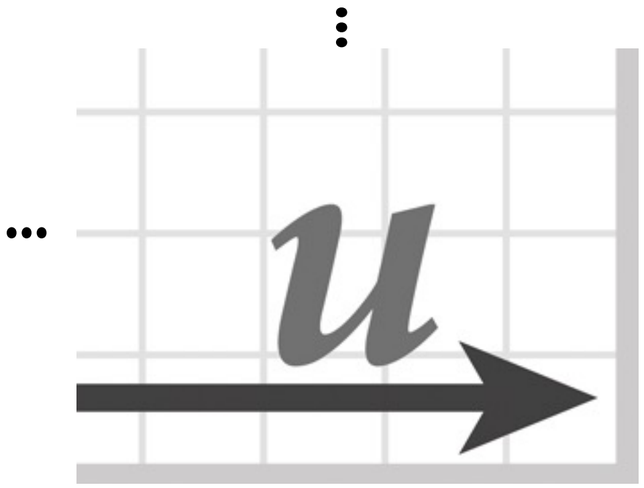


- **Minification:**
  - Area of screen pixel maps to large region of texture (filtering required -- averaging)
  - One texel corresponds to far less than a pixel on screen
  - Example: when scene object is very far away

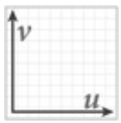
- **Magnification:**
  - Area of screen pixel maps to tiny region of texture (interpolation required)
  - One texel maps to many screen pixels
  - Example: when camera is very close to scene object (need higher resolution texture map)

Figure credit: Akeley and Hanrahan

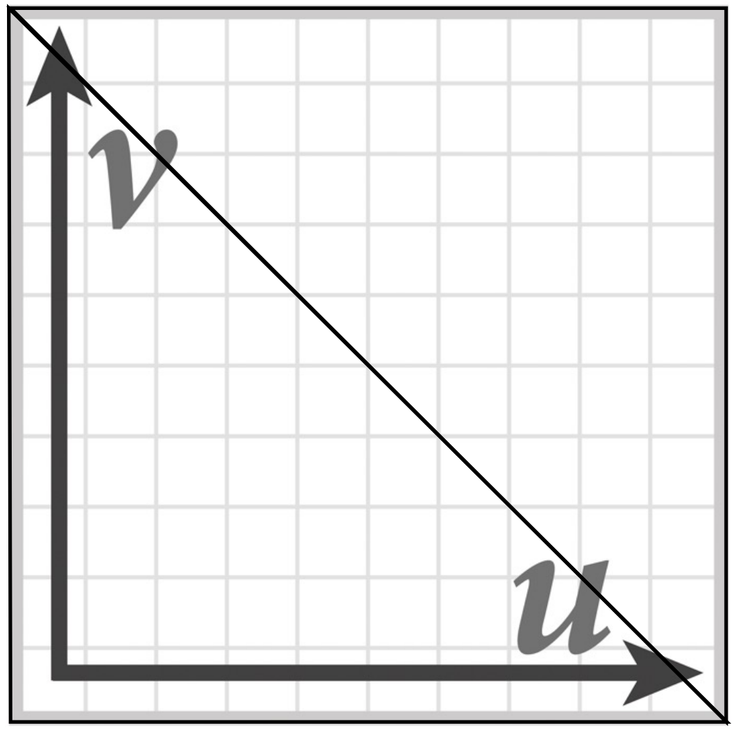
Filtering textures



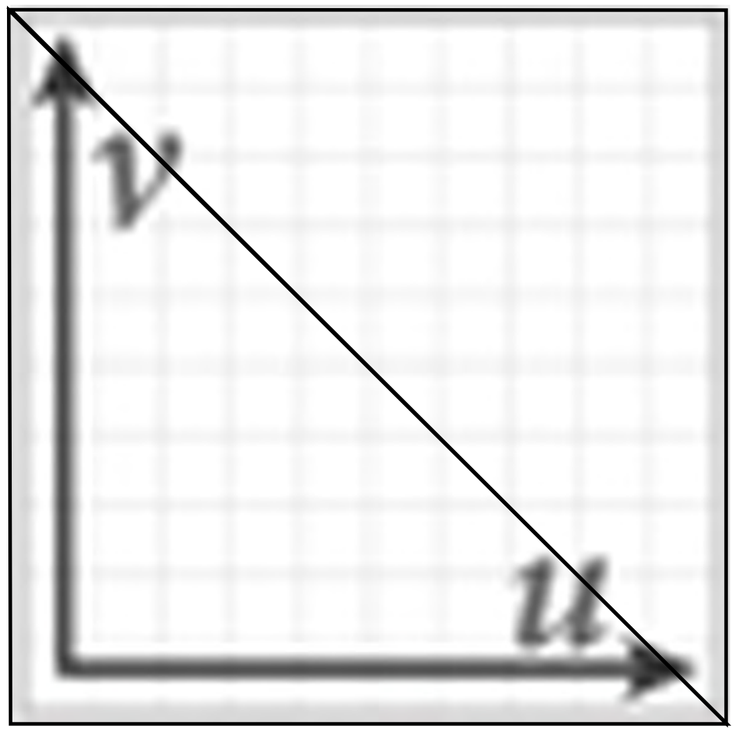
Actual texture: 700x700 image
(only a crop is shown)



Actual texture: 64x64 image

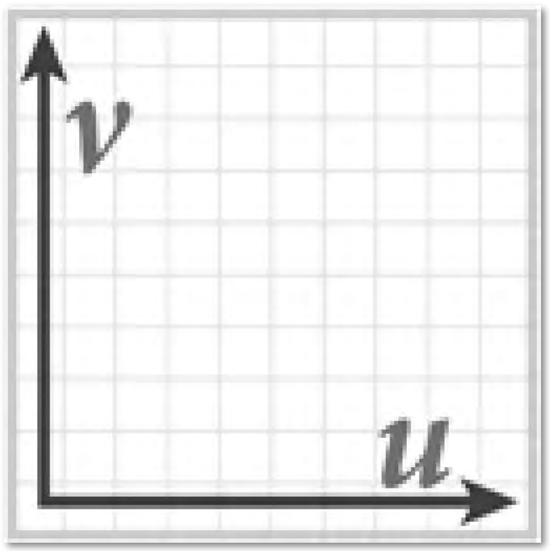


Texture minification

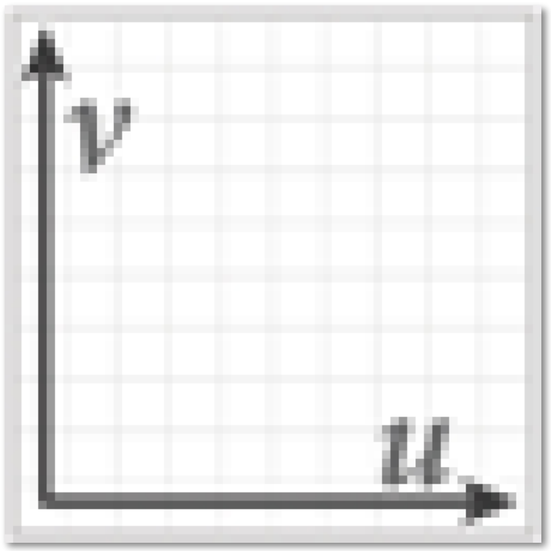


Texture magnification

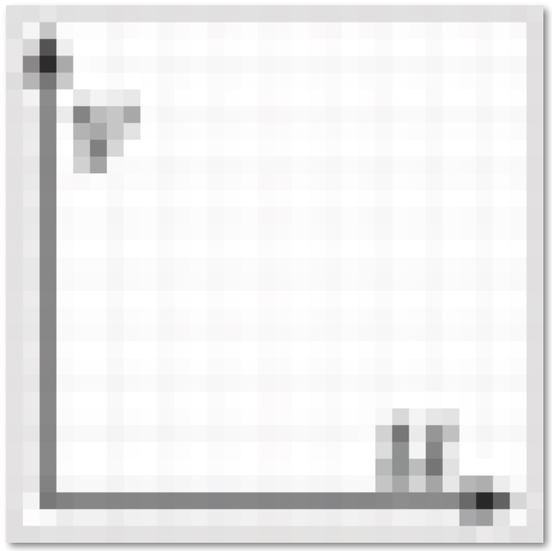
Mipmap (L. Williams 83)



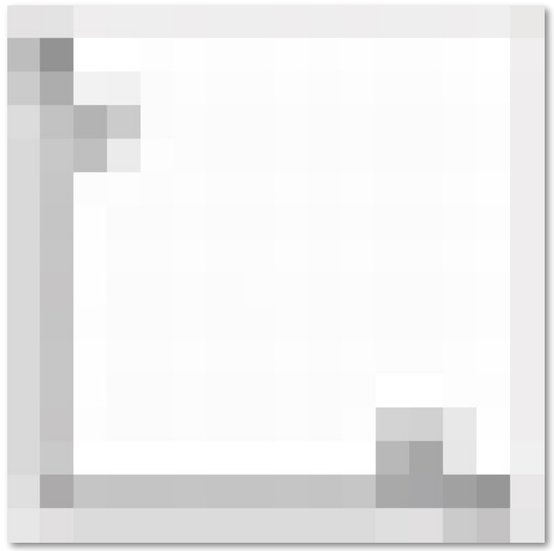
Level 0 = 128x128



Level 1 = 64x64

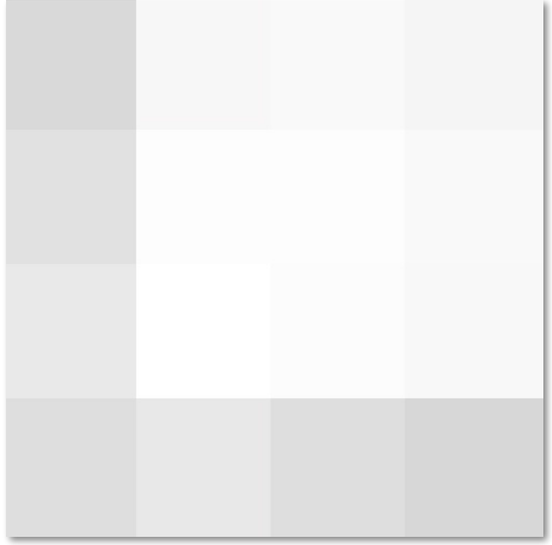


Level 2 = 32x32

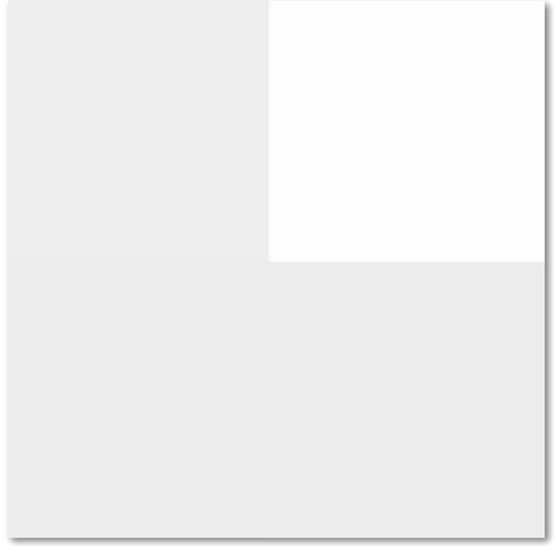


Level 3 = 16x16

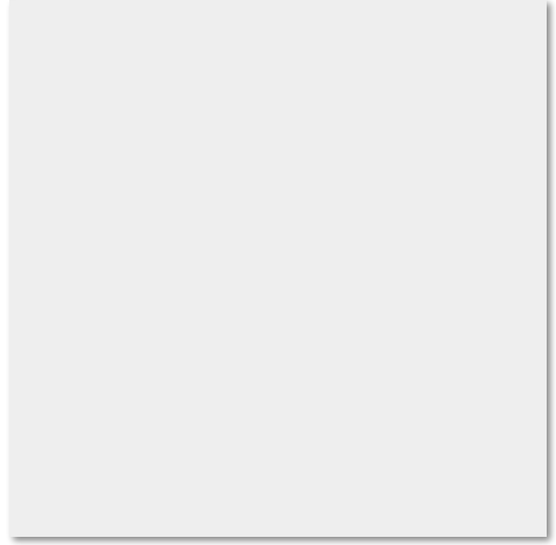
Level 4 = 8x8



Level 5 = 4x4



Level 6 = 2x2



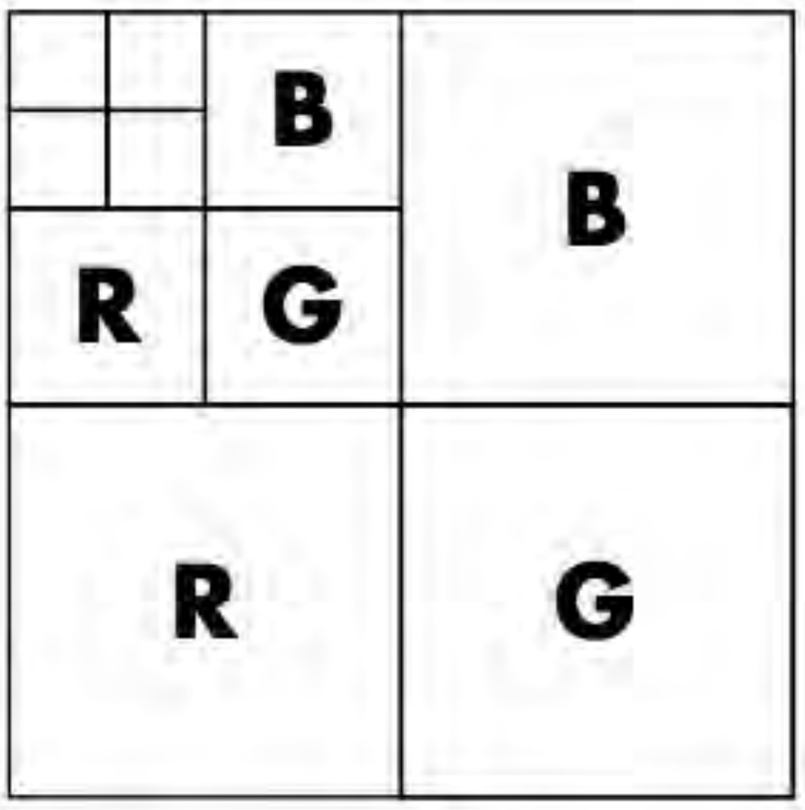
Level 7 = 1x1

Idea: prefilter texture data to remove high frequencies

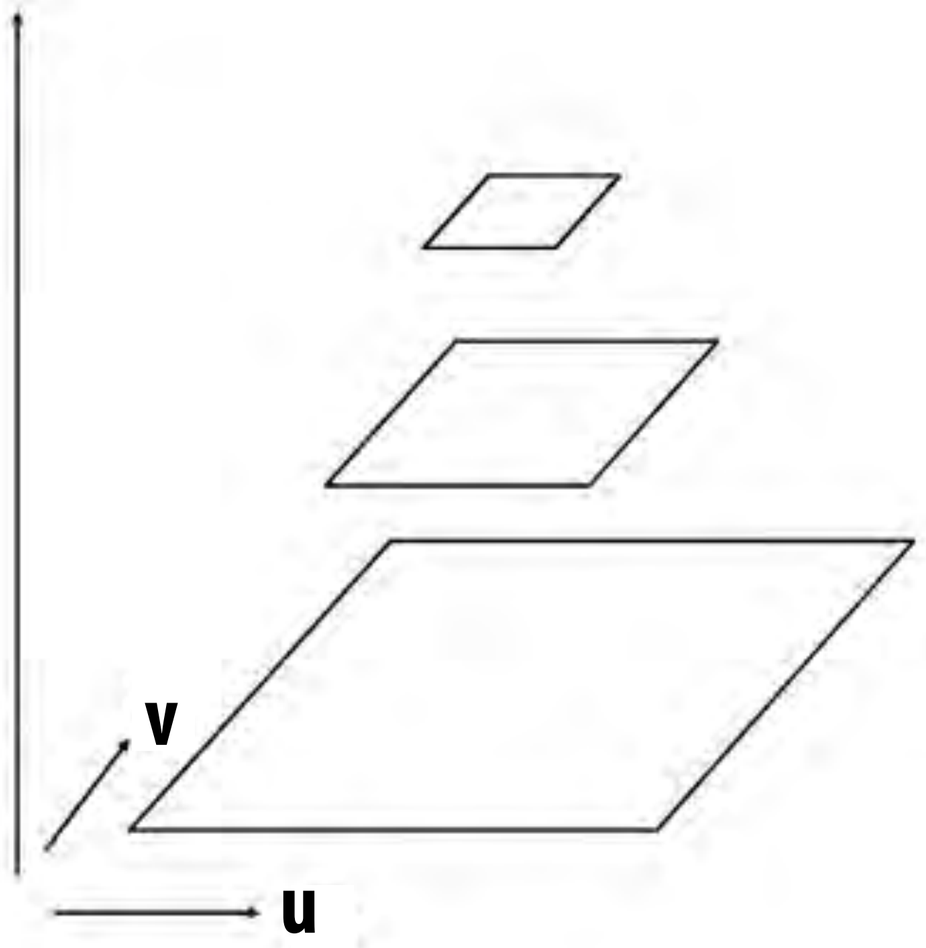
Texels at higher levels store integral of the texture function over a region of texture space (downsampled images)

Texels at higher levels represent low-pass filtered version of original texture signal

Mipmap (L. Williams 83)



Williams' original proposed mip-map layout

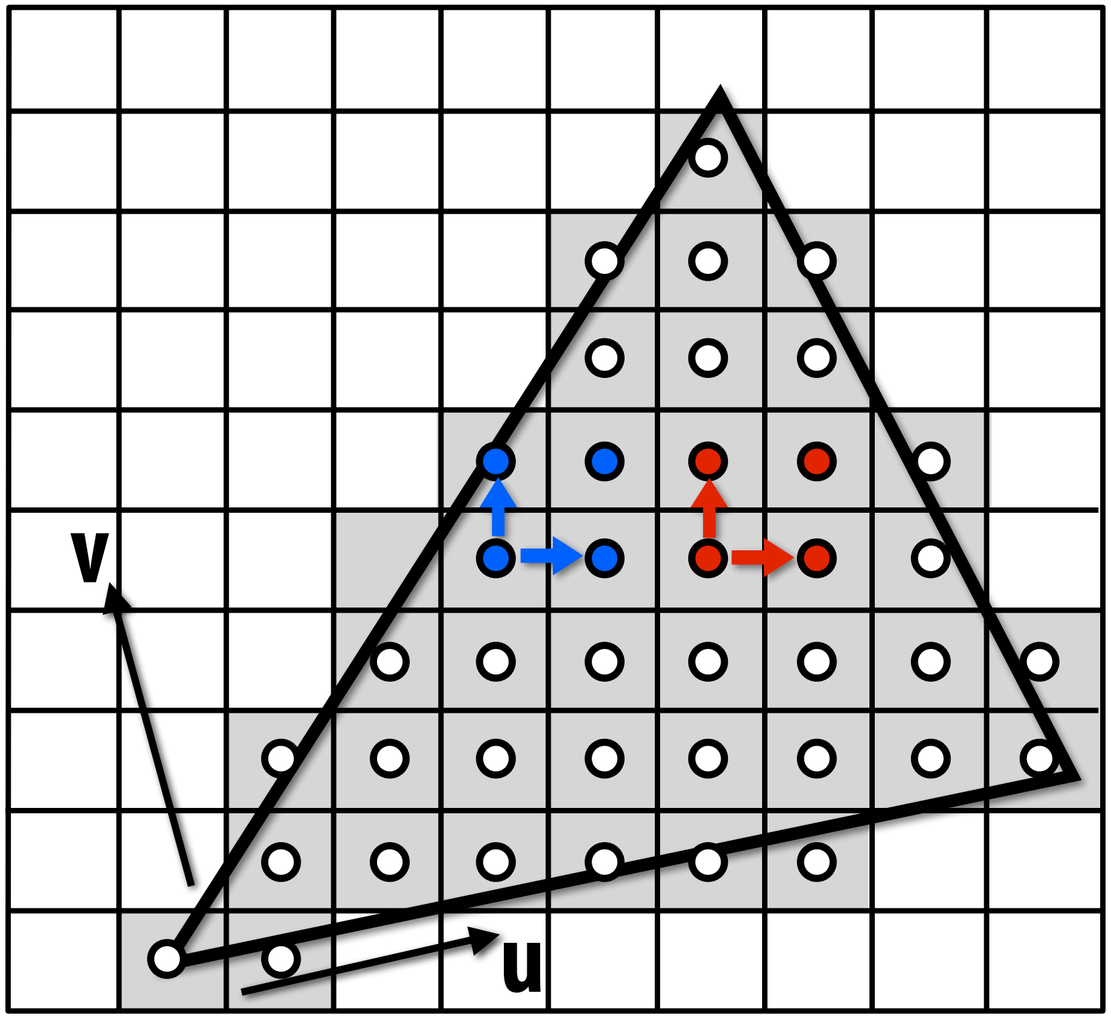


"Mip hierarchy" level = *d*

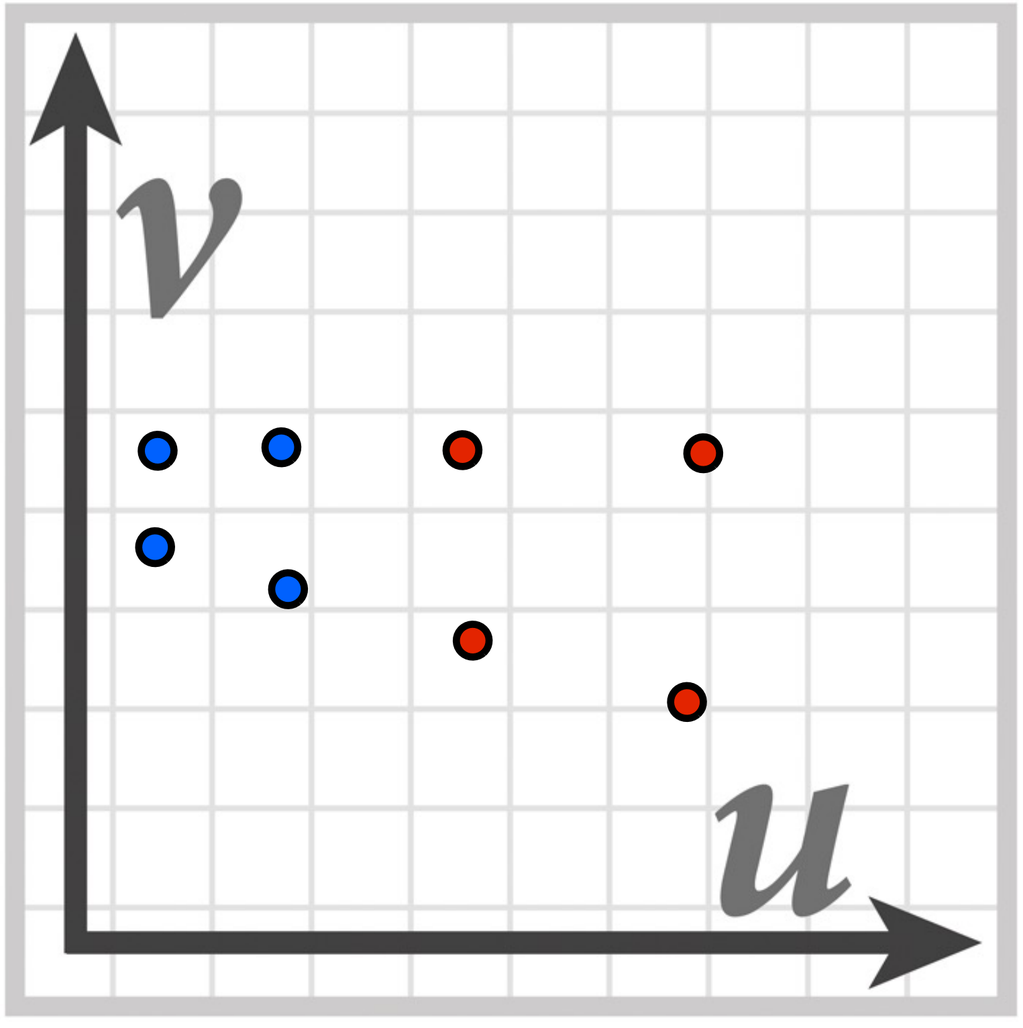
What is the storage overhead of a mipmap?

Computing *d*

Compute differences between texture coordinate values of neighboring screen samples



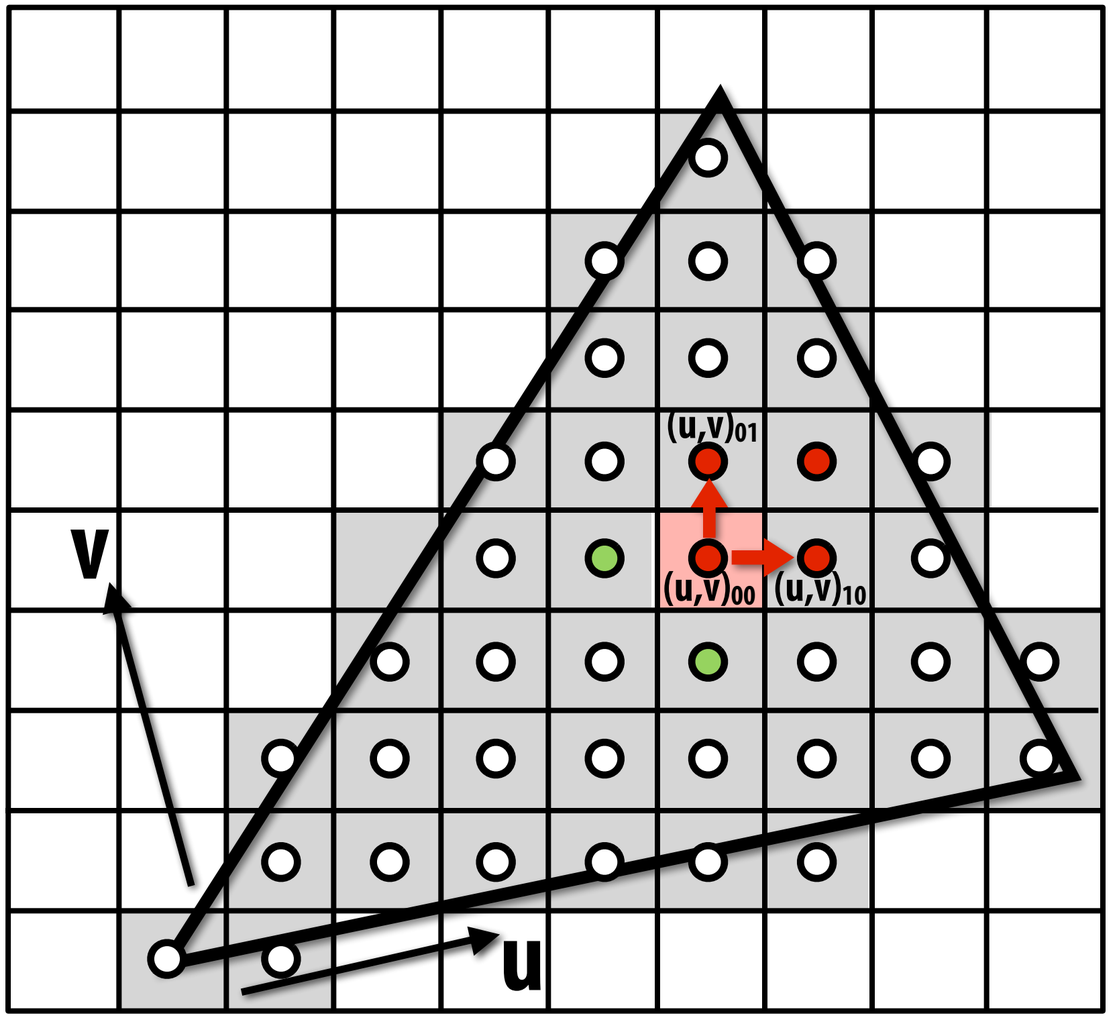
Screen space

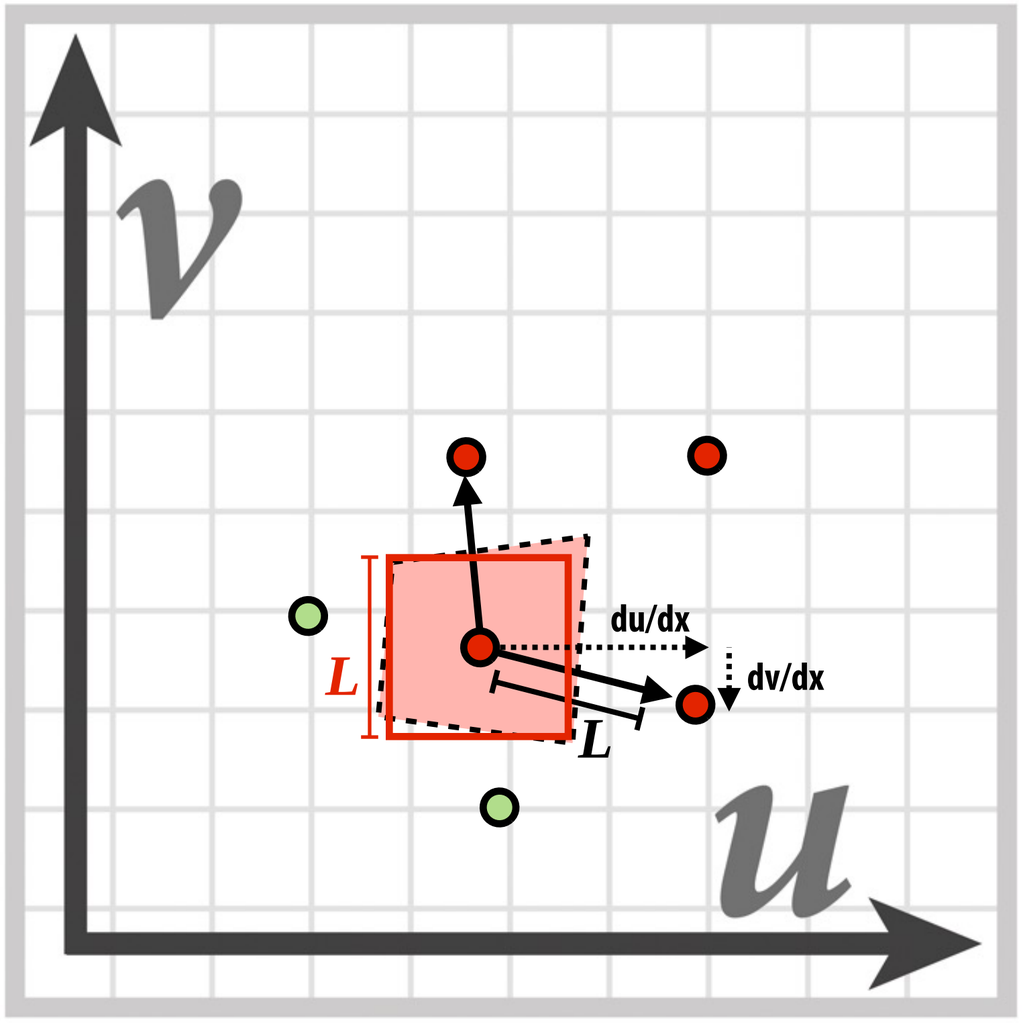


Texture space

Computing *d*

Compute differences between texture coordinate values of neighboring fragments





$du/dx = u_{10} - u_{00}$ $\quad dv/dx = v_{10} - v_{00}$

$du/dy = u_{01} - u_{00}$ $\quad dv/dy = v_{01} - v_{00}$

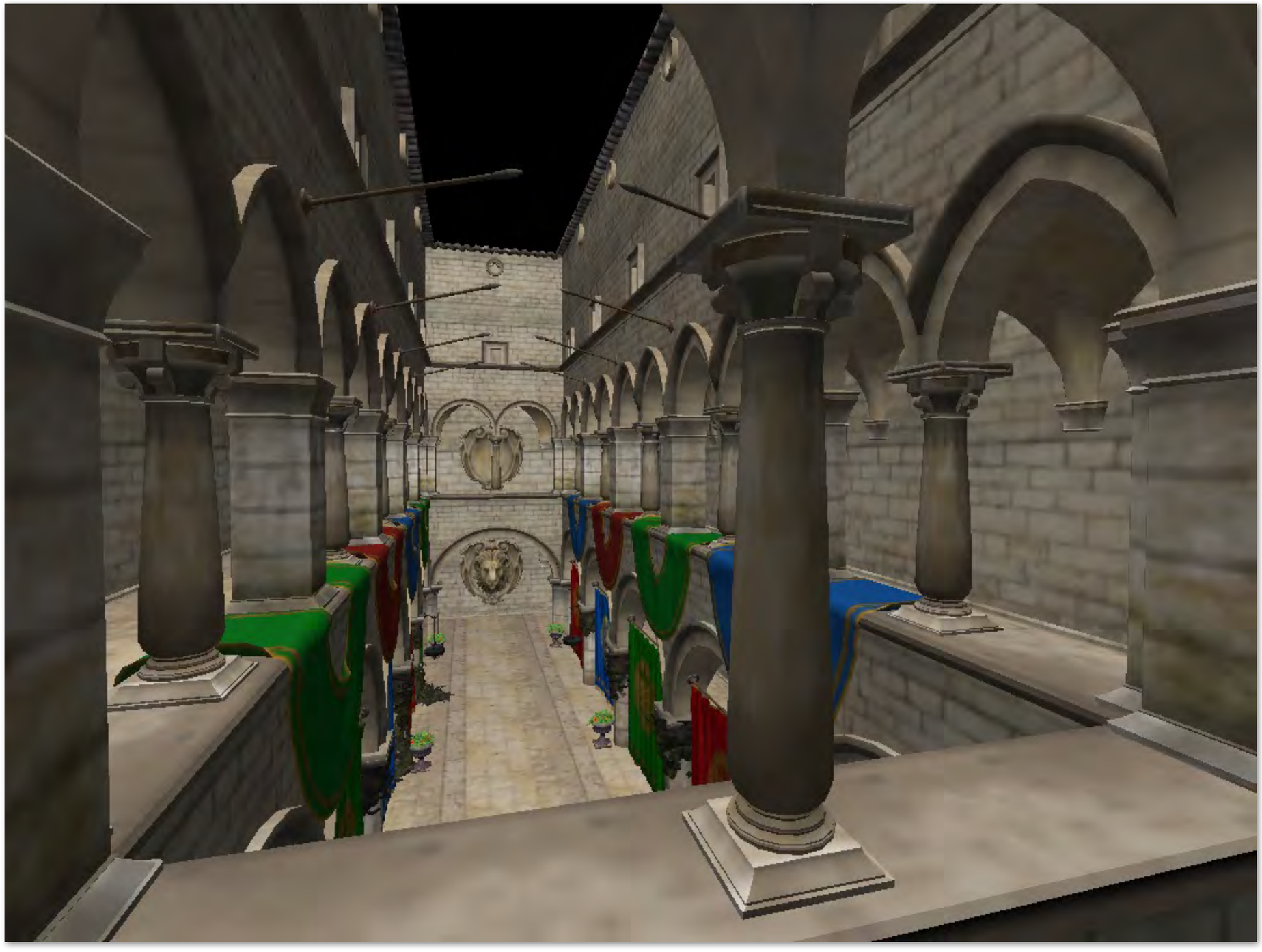
$$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

mip-map d = $\log_2 L$

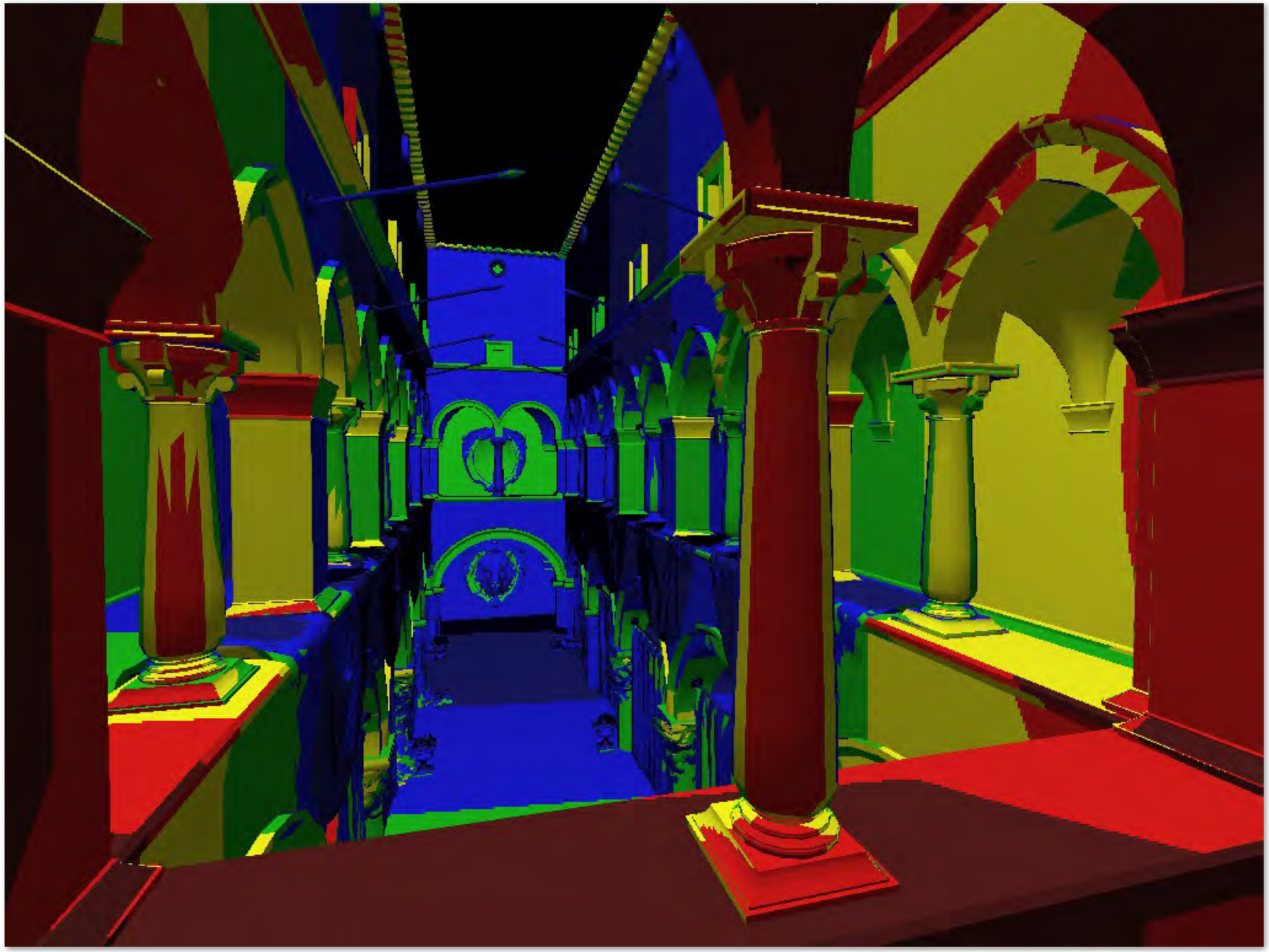
Sponza (bilinear resampling at level 0)

Sponza (bilinear resampling at level 2)

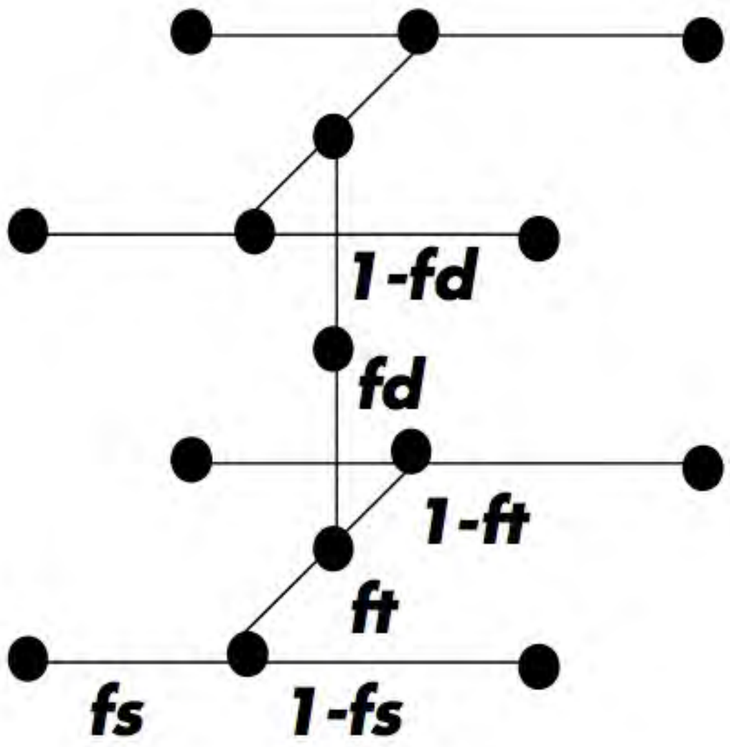
Sponza (bilinear resampling at level 4)



Visualization of mip-map level
(bilinear filtering only: *d* clamped to nearest level)



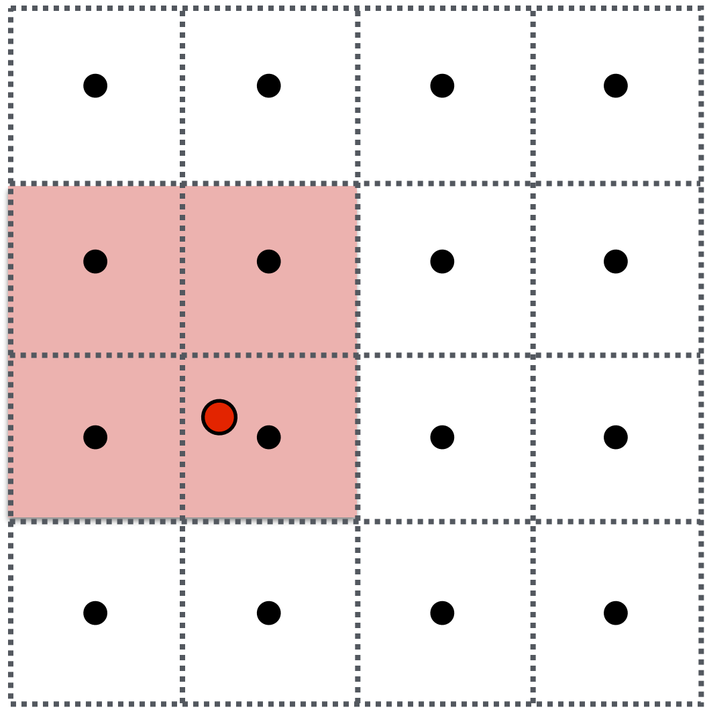
“Tri-linear” filtering



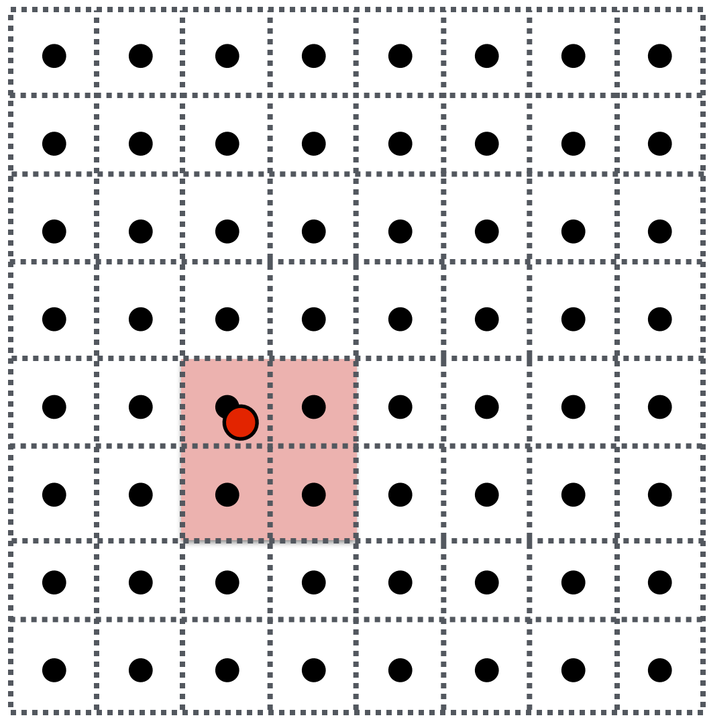
$$lerp(t, v_1, v_2) = v_1 + t(v_2 - v_1)$$

Bilinear resampling:
- four texel reads
- 3 lerps (3 mul + 6 add)

Trilinear resampling:
- eight telex reads
- 7 lerps (7 mul + 14 add)



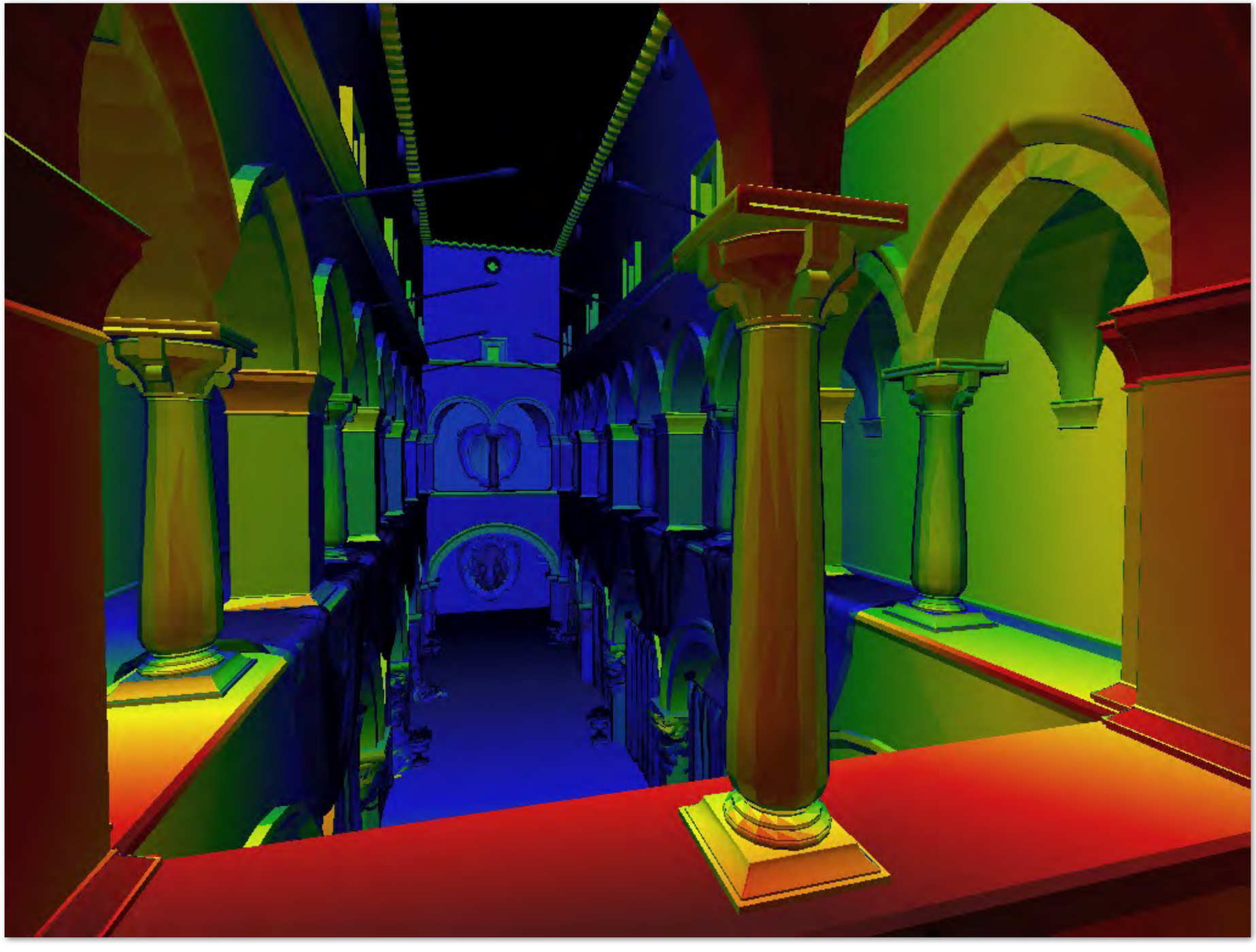
mip-map texels: level d+1



mip-map texels: level d

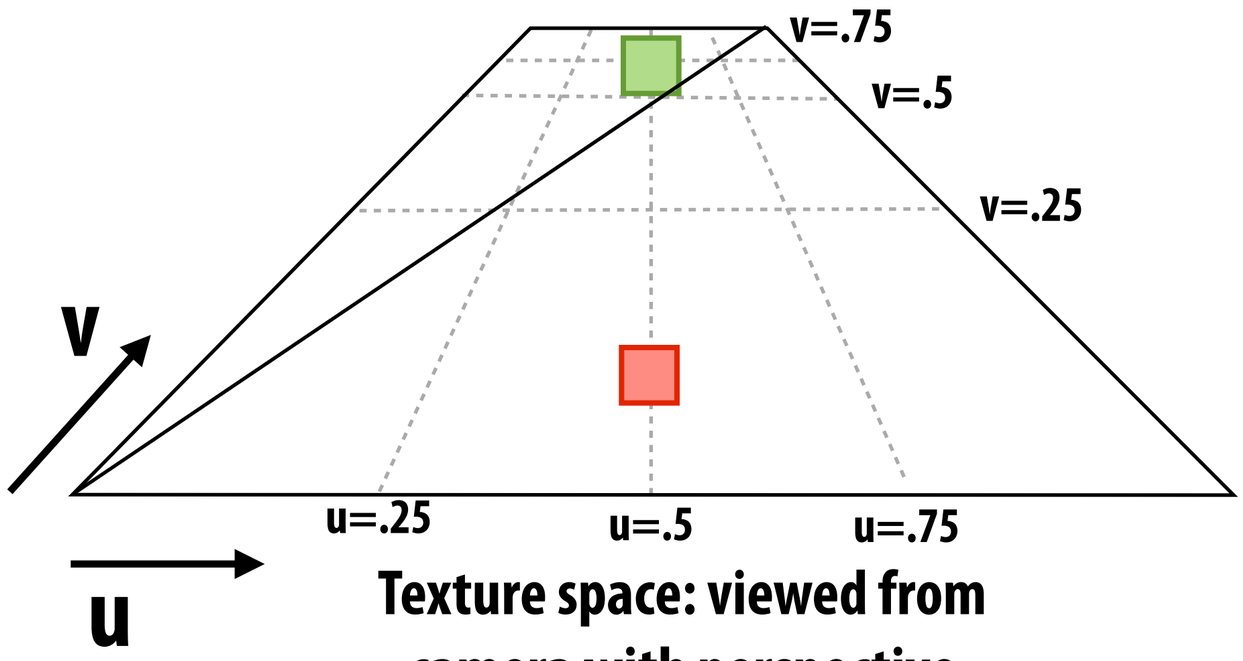
Figure credit: Akeley and Hanrahan

Visualization of mip-map level
(trilinear filtering: visualization of continuous *d*)

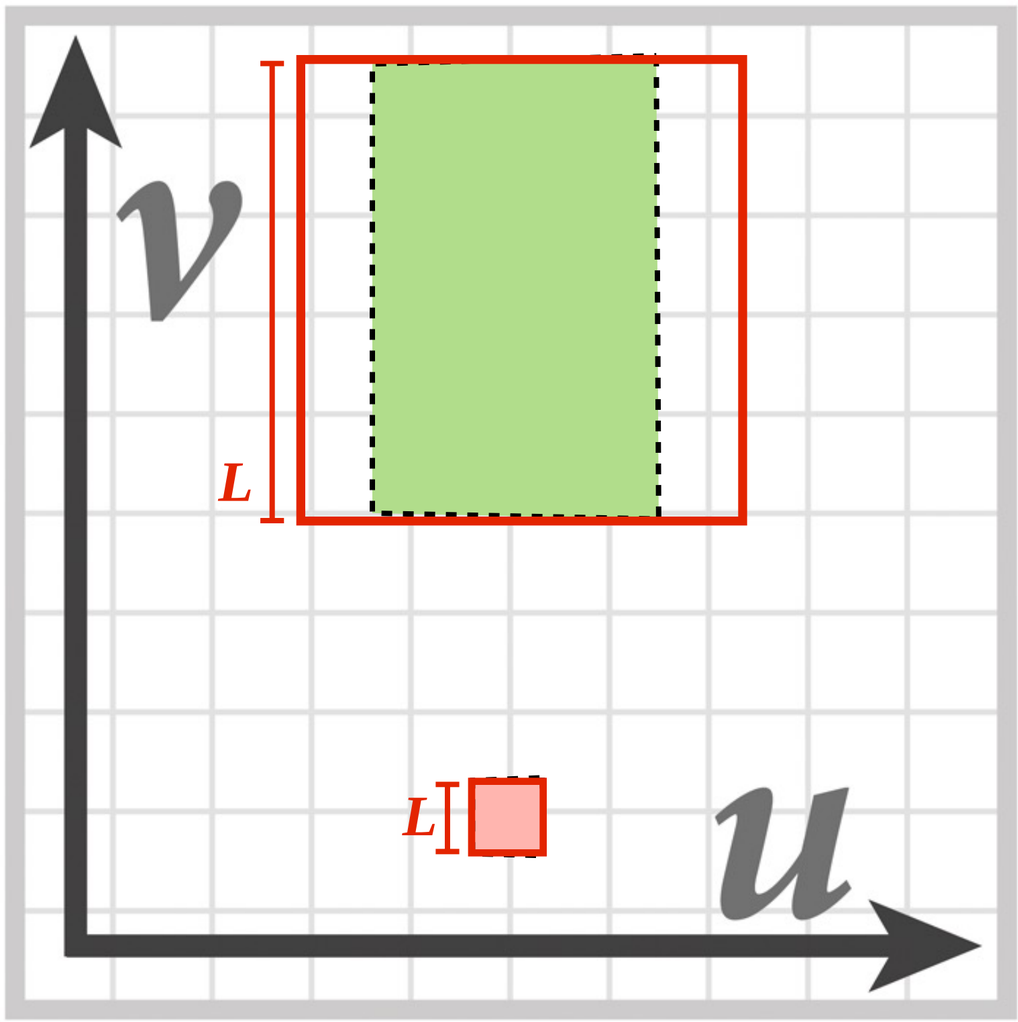


Pixel area may not map to isotropic region in texture

Proper filtering requires anisotropic filter footprint



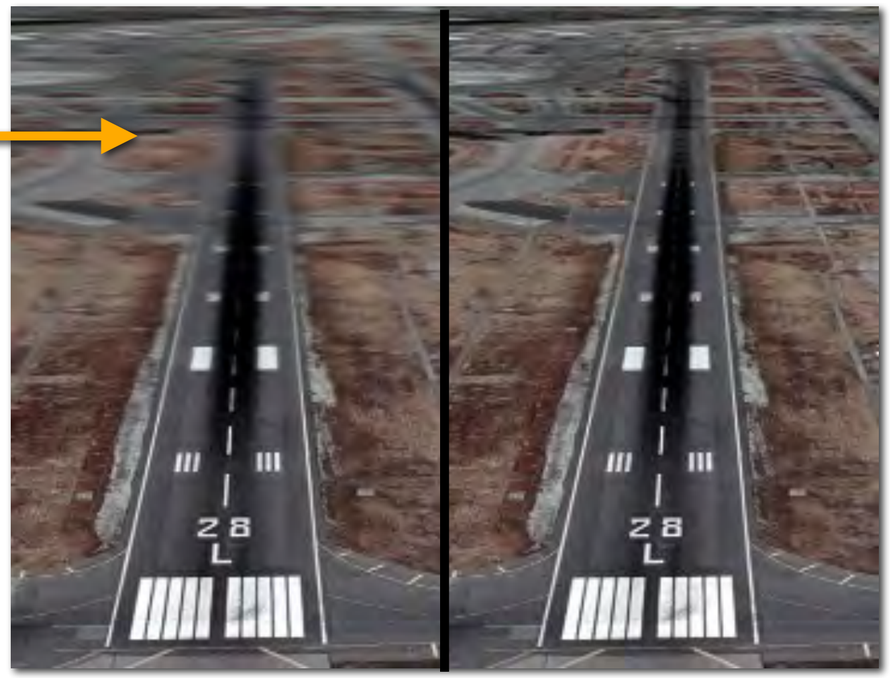
Texture space: viewed from camera with perspective



$$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

$$\textit{mip-map } d = \log_2(L)$$

Overblurring in *u* direction



Trilinear (Isotropic) Filtering

Anisotropic Filtering

Summary: texture filtering using the mip map

- **Small storage overhead (33%)**
  - Mipmap is 4/3 the size of original texture image

- **For each isotropically-filtered sampling operation**
  - Constant filtering cost (independent of *d*)
  - Constant number of texels accessed (independent of *d*)

- **Combat aliasing with prefiltering, rather than supersampling**
  - Recall: we used supersampling to address aliasing problem when sampling coverage

- **Bilinear/trilinear filtering is isotropic and thus will "overblur" to avoid aliasing**
  - Anisotropic texture filtering provides higher image quality at higher compute and memory bandwidth cost

Summary: a texture sampling operation

1. Compute u and v from screen sample x,y (via evaluation of attribute equations)
2. Compute du/dx, du/dy, dv/dx, dv/dy differentials from screen-adjacent samples.
3. Compute *d*
4. Convert normalized texture coordinate (u,v) to texture coordinates texel_u, texel_v
5. Compute required texels in window of filter
6. Load required texels (need eight texels for trilinear)
7. Perform tri-linear interpolation according to (texel_u, texel_v, d)

Takeaway: a texture sampling operation is not just an image pixel lookup! It involves a significant amount of math.

All modern GPUs have dedicated fixed-function hardware support for performing texture sampling operations.

Texturing summary

- Texture coordinates: define mapping between points on triangle's surface (object coordinate space) to points in texture coordinate space

- Texture mapping is a sampling operation and is prone to aliasing
  - Solution: prefilter texture map to eliminate high frequencies in texture signal
  - Mip-map: precompute and store multiple multiple resampled versions of the texture image (each with different amounts of low-pass filtering)
  - During rendering: dynamically select how much low-pass filtering is required based on distance between neighboring screen samples in texture space
    - Goal is to retain as much high-frequency content (detail) in the texture as possible, while avoiding aliasing

Outline

- Introduction
- Texture mapping in OpenGL
- Filtering and Mipmaps
- Example
- Non-color texture maps

Complete example

```
void initTexture()
{
  load image into memory; // can use libjpeg, libtiff, or other image library
  // image should be stored as a sequence of bytes, usually 3 bytes per
     pixel (RGB), or 4 bytes (RGBA); image size is 4 * 256 * 256 bytes in
     this example
  // we assume that the image data location is stored in pointer “pointerToImage”

  // create placeholder for texture
  glGenTextures(1, &texName); // must declare a global variable in
program header: GLUint texName
  glBindTexture(GL_TEXTURE_2D, texName); // make texture
“texName” the currently active texture

  (continues on next page)
```

Complete example (part 2)

```
 // specify texture parameters (they affect whatever texture is active)
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
// repeat pattern in s
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// repeat pattern in t


  // use linear filter both for magnification and minification
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
  GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
  GL_LINEAR);

  // load image data stored at pointer “pointerToImage” into the currently
     active texture (“texName”)
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0,
     GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage);

} // end init()
```

Complete example (part 3)

```
void display()
{
  …
  // no modulation of texture color with lighting; use texture color directly
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
   GL_REPLACE);

  // turn on texture mapping (this disables standard OpenGL lighting, unless in
GL_MODULATE mode)
  glEnable(GL_TEXTURE_2D);

  (continues on next page)
```

Complete example (part 4)

```
glBegin(GL_QUADS); // draw a textured quad
  glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
  glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
  glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
  glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
  glEnd();

  // turn off texture mapping
  glDisable(GL_TEXTURE_2D);

  // draw some non-texture mapped objects
(standard OpenGL lighting will be used if it is enabled)
  …
  // switch back to texture mode, etc.
  …
} // end display()
```

Outline

- Introduction
- Texture mapping in OpenGL
- Filtering and Mipmaps
- Example
- Non-color texture maps

Textures do not have to represent color

- Specularity (patches of shininess)
- Transparency (patches of clearness)
- Normal vector changes (bump maps)
- Reflected light (environment maps)
- Shadows
- Changes in surface height (displacement maps)

Represent precomputed lighting and shadows

Original model

With ambient occlusion

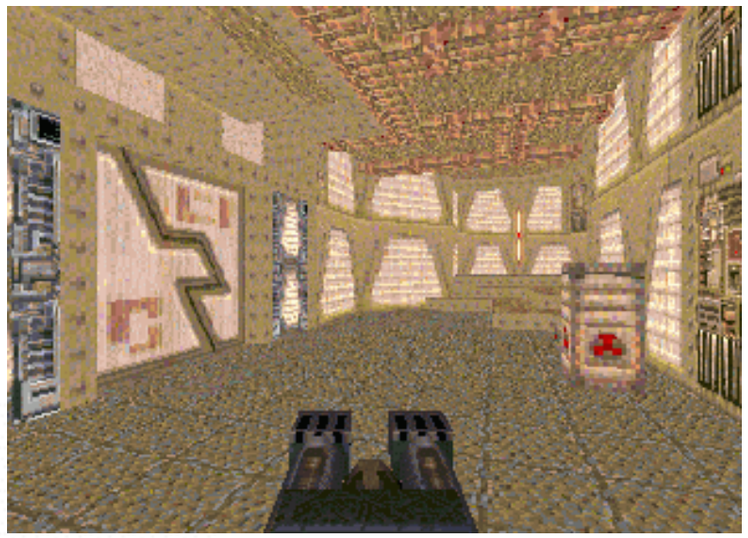
Extracted ambient occlusion map

Grace Cathedral environment map

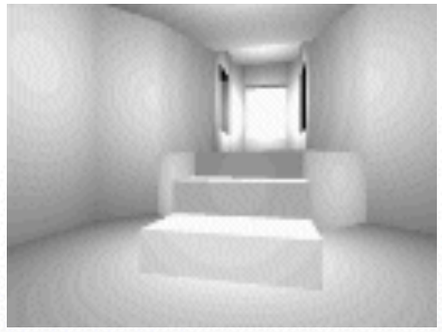
Environment map used in rendering

Light Mapping

- *Quake* uses *light maps* in addition to texture maps. Texture maps are used to add detail to surfaces, and light maps are used to store pre-computed illumination. The two are multiplied together at run-time, and cached for efficiency.



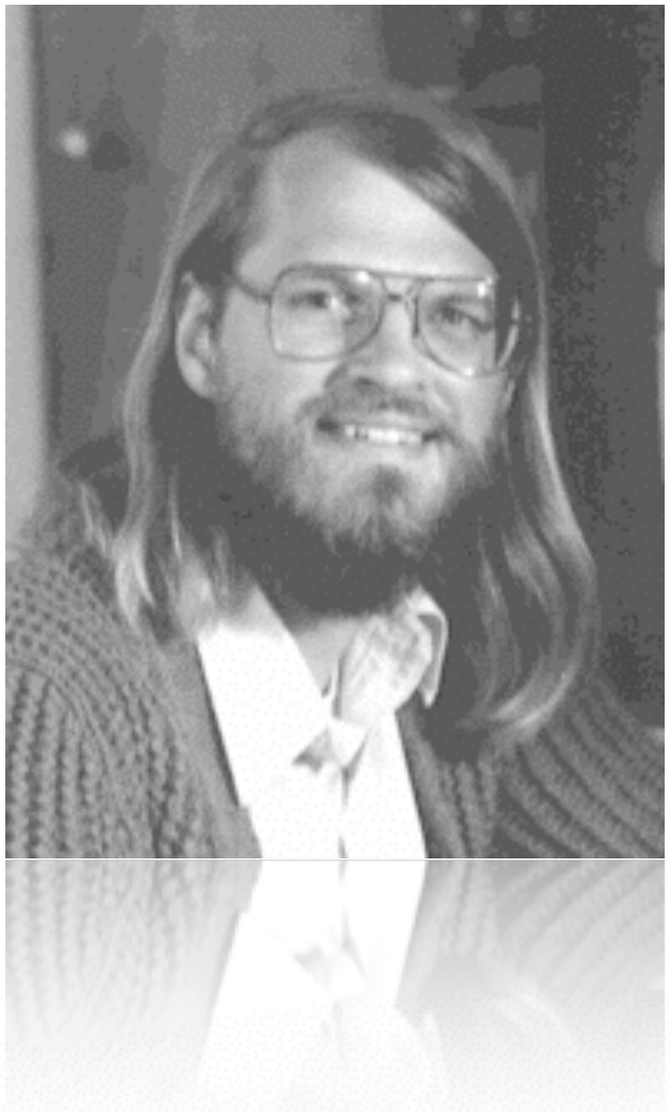
Texture Map Only



Light Map

Texture + Light Map

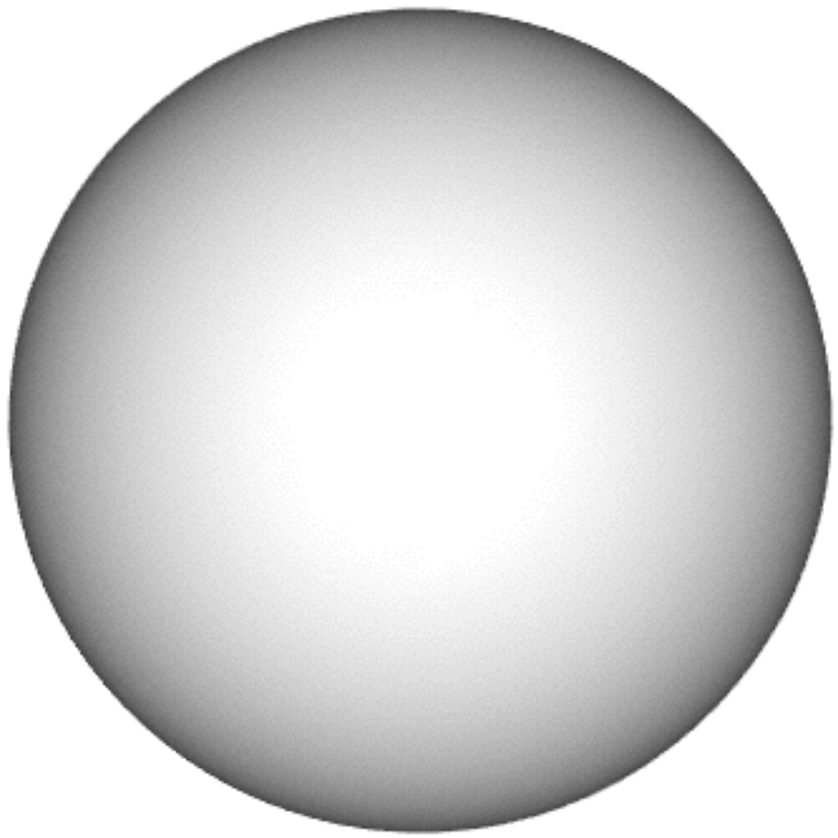
A long time ago, in 1978



… bump mapping was born

courtesy by ZBrush

For Meshes



vertex normal interpolation

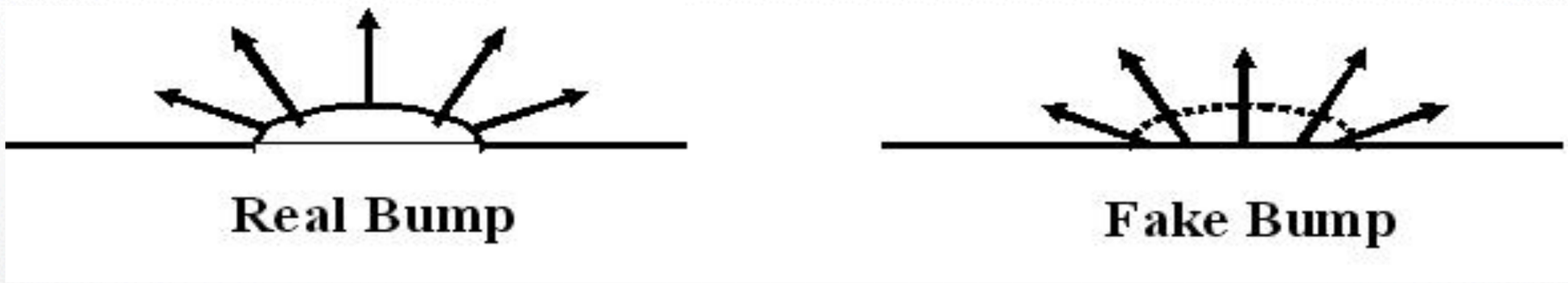
↓

smooth shading

What about
accessing **textures** to modify **surface normals**...

Bump mapping

- How do you make a surface look *rough*?
  - Option 1: model the surface with many small polygons
  - Option 2: perturb the normal vectors before the shading calculation
    - Fakes small displacements above or below the true surface
    - The surface doesn't actually change,
      but shading makes it look like there are irregularities!
    - A texture stores information about the "fake" height of the surface



Normal mapping

Use texture value to perturb surface normal to give appearance of a bumpy surface
Observe: smooth silhouette and smooth shadow boundary indicates surface geometry is not bumpy

Rendering using high-resolution surface geometry (note bumpy silhouette and shadow boundary)

We now have an **inexpensive way** to add **geometric details**

Other bump mapping techniques exist

Further Readings

- “Simulation of Wrinkled Surfaces” [Blinn 1978]
- “Real-Time Rendering” [Akenine-Möller and Haines 2002] p.166 – 177