

CSE160 – Oct 29 - Textures

- What are textures for?
- UV Coordinates
- Where do UV come from?
- Texture sampling NOT 1:1
- Magnification
- Minification
- Mipmapping
- Anisotropic Texturing
- OpenGL Texture
- Beyond 2D Colored Surfaces
- Administrative
- Q&A

Blocky Animal Contest

What are textures for?

Texture mapping



Many uses of texture mapping

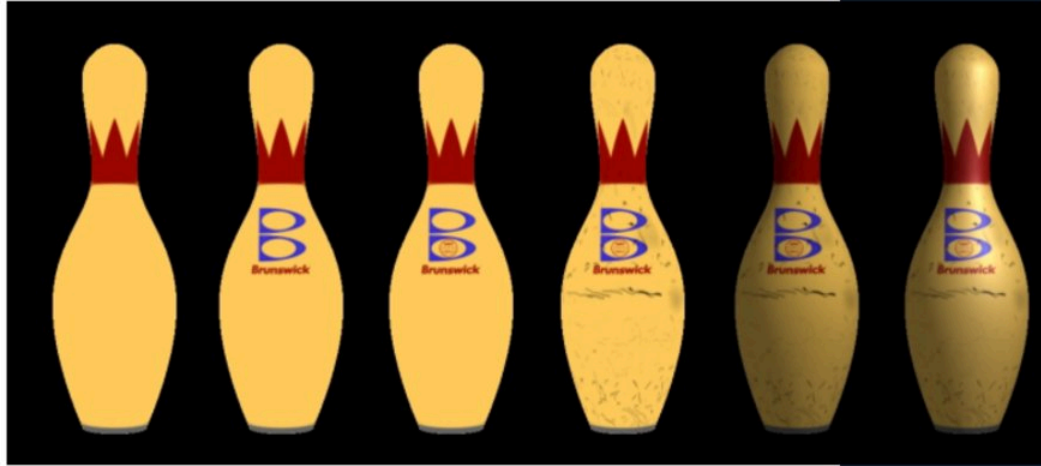
Define variation in surface reflectance



Pattern on ball

Wood grain on floor

Describe surface material properties

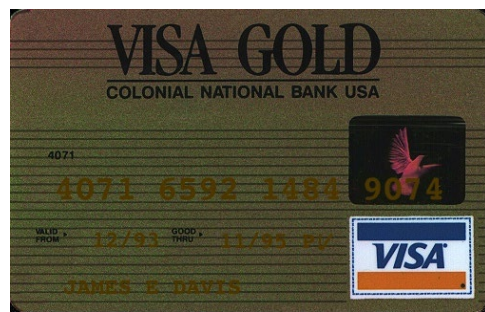


Multiple layers of texture maps for color, logos, scratches, etc.



©2013 CRYTEK GMBH. ALL RIGHTS RESERVED. RYSE IS A REGISTERED TRADEMARK OF CRYTEK GMBH

RYSE
SON OF ROME



4071 6592 1484 9074

12/93 11/95 P/V

JAMES E DAVIS

4071 6592 1484

12/93 11/95 P/V

JAMES E DAVIS



CS348B -1993
Student: James Davis

Represent precomputed lighting and shadows



Original model



With ambient occlusion



Extracted ambient occlusion map

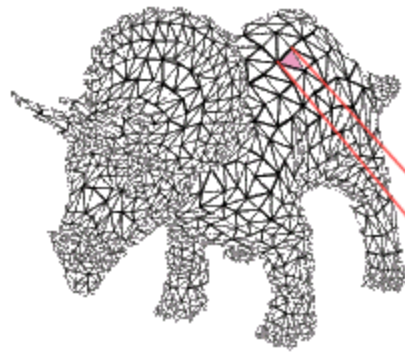


Grace Cathedral environment map

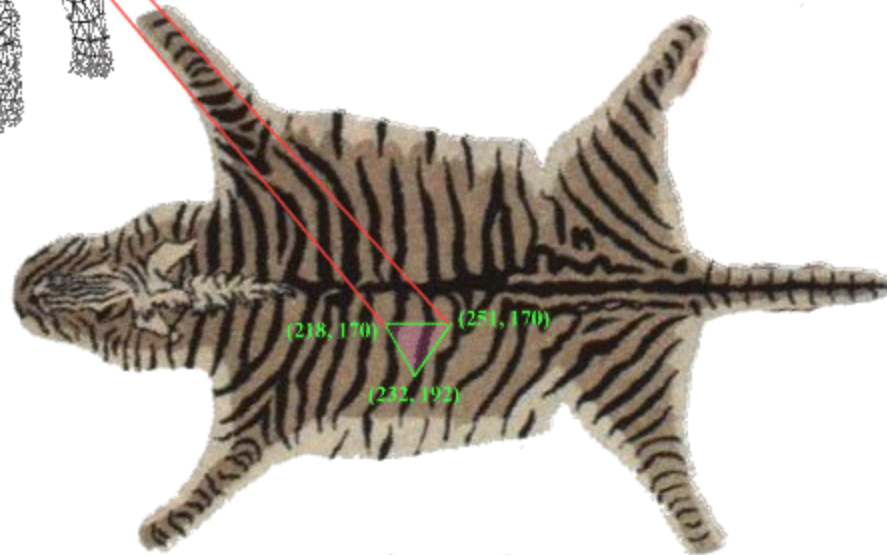


Environment map used in rendering

UV Coordinates

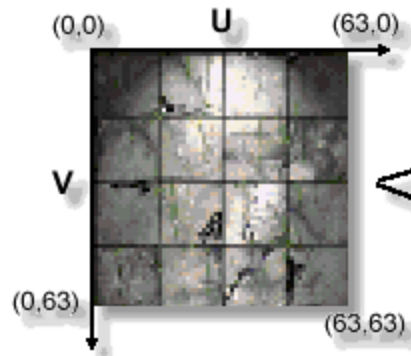


*For each triangle in the model
establish a corresponding region
in the phototexture*

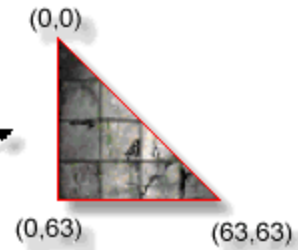


*During rasterization interpolate the
coordinate indices into the texture map*

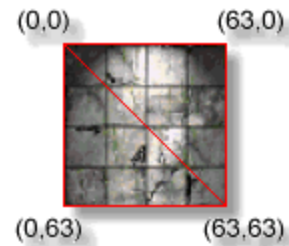
Source Texture Map 64x64 Pixels



Example 1: Single Triangle

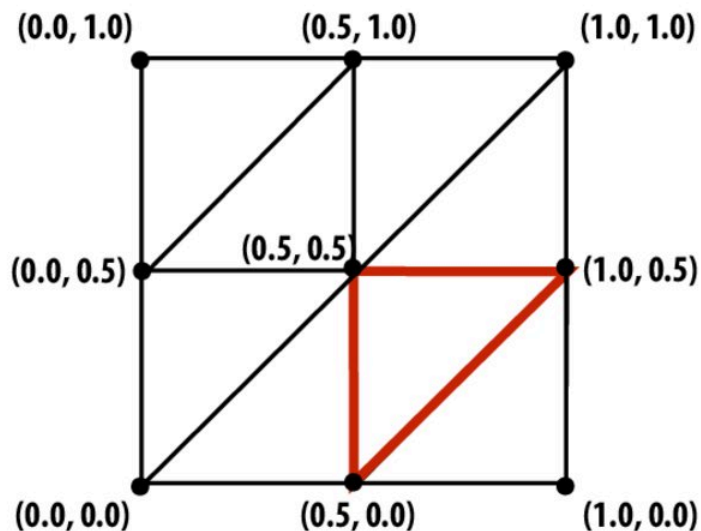


Example 2: Two Triangles Making a Quadrilateral

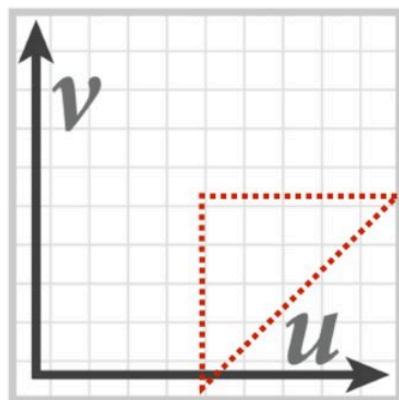


Texture coordinates

“Texture coordinates” define a mapping from surface coordinates (points on triangle) to points in texture domain.

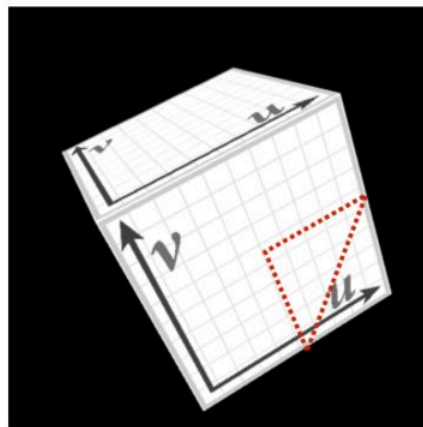


Eight triangles (one face of cube) with surface parameterization provided as per-vertex texture coordinates.



$\text{myTex}(u, v)$ is a function defined on the $[0, 1]^2$ domain (represented by 2048x2048 image)

Location of highlighted triangle in texture space shown in red.



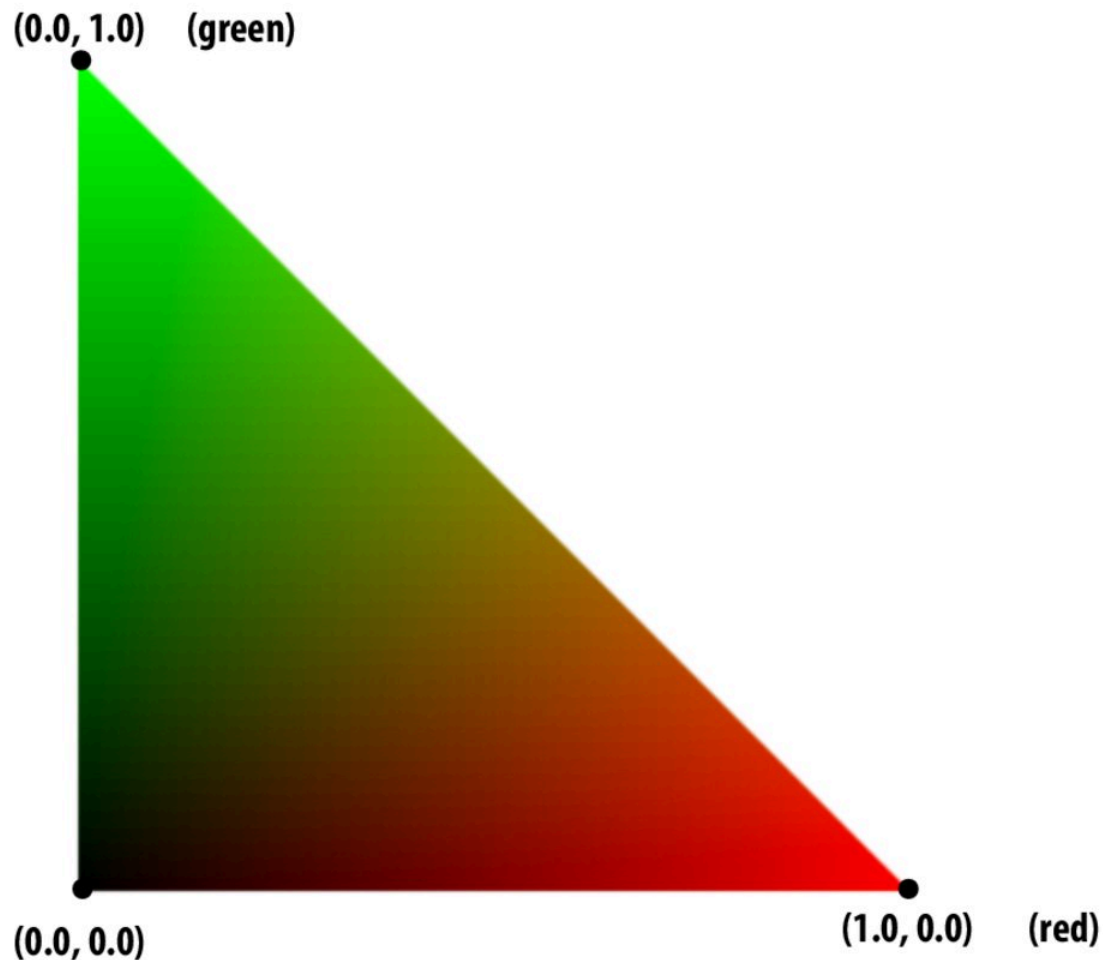
Final rendered result (entire cube shown).

Location of triangle after projection onto screen shown in red.

Today we'll assume surface-to-texture space mapping is provided as per vertex attribute (Not discussing methods for generating surface texture parameterizations)

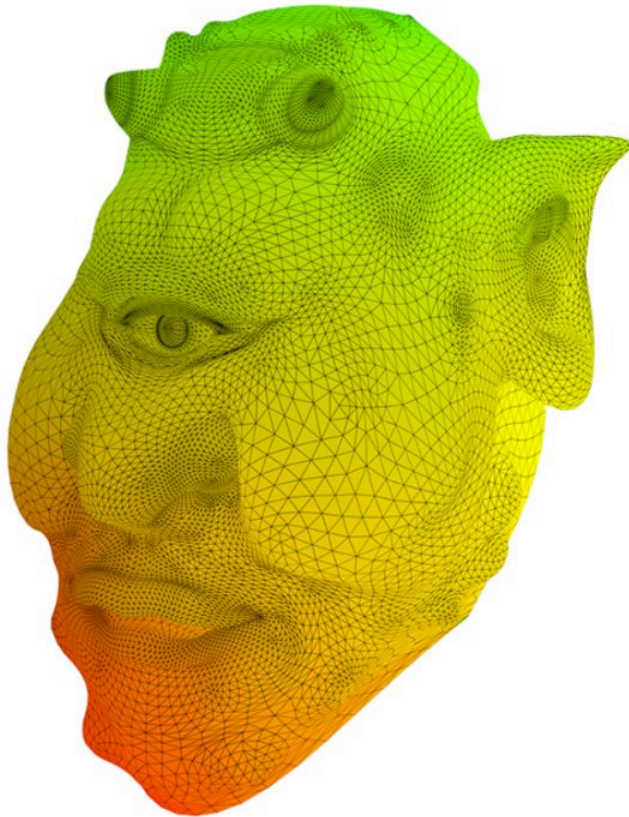
Visualization of texture coordinates

Texture coordinates linearly interpolated over triangle

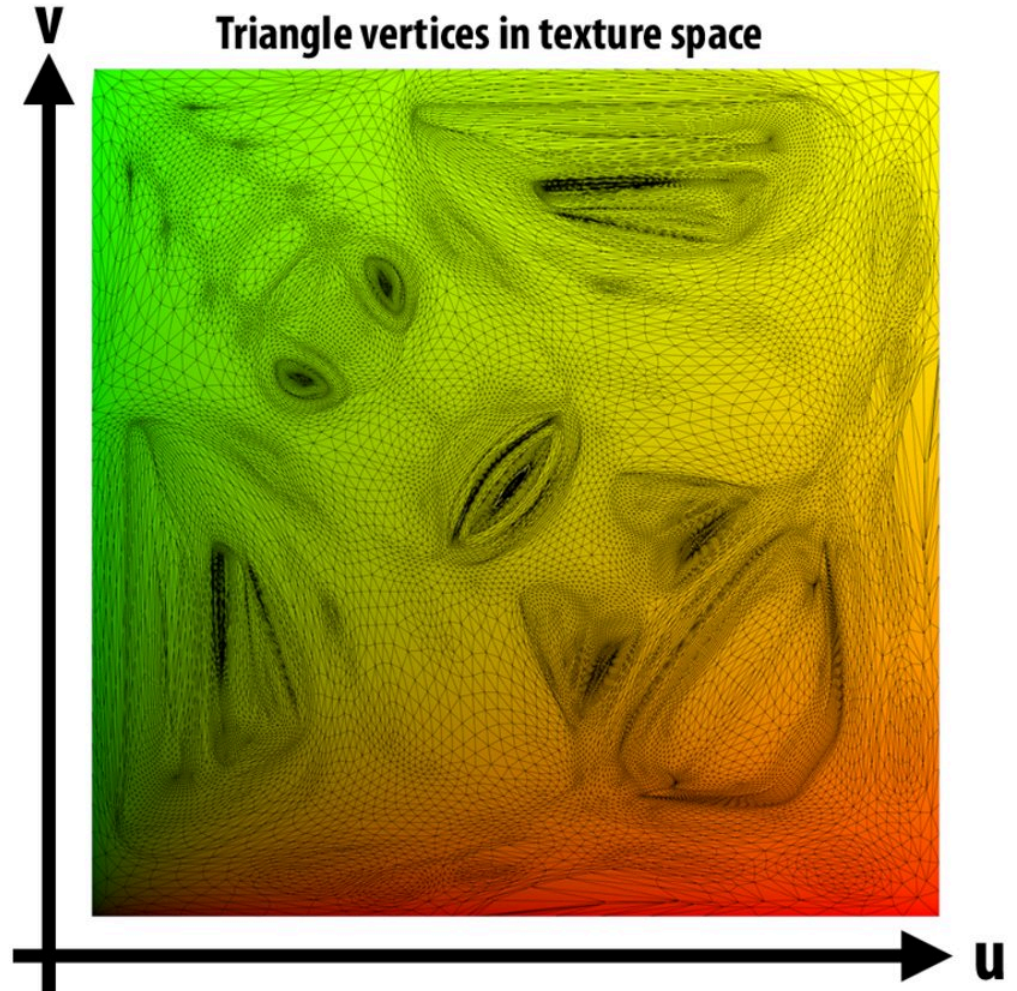


More complex mapping

Visualization of texture coordinates



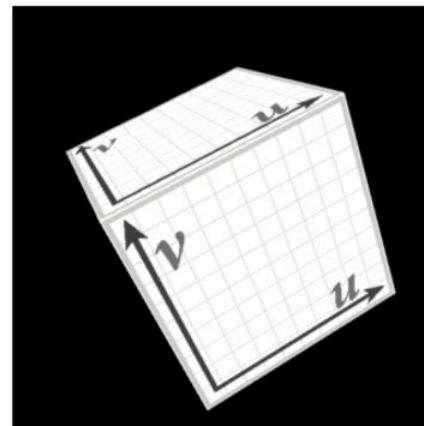
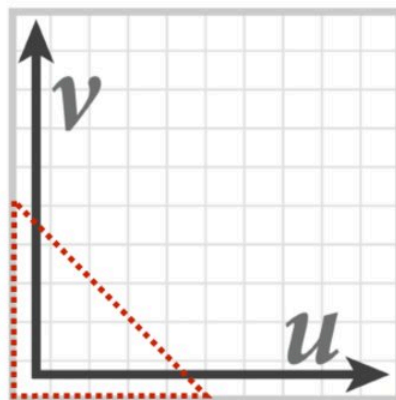
Triangle vertices in texture space



Each vertex has a coordinate (u,v) in texture space.
(Actually coming up with these coordinates is another story!)

Texture sampling 101

- Basic algorithm for mapping texture to surface:
 - For each color sample location (X,Y)
 - Interpolate U and V coordinates across triangle to get value at (X,Y)
 - Sample (evaluate) texture at (U,V)
 - Set color of fragment to sampled texture value

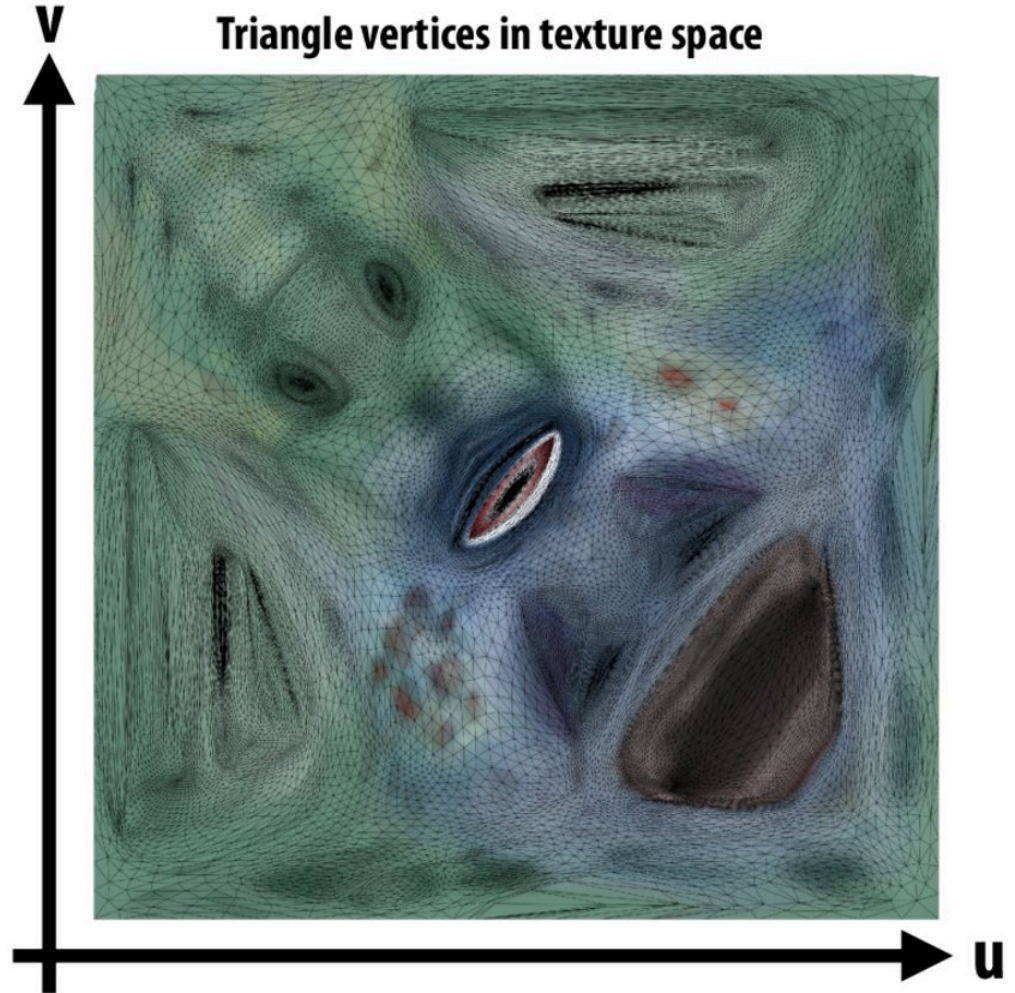


Texture mapping adds detail

Rendered result



Triangle vertices in texture space



Texture mapping adds detail

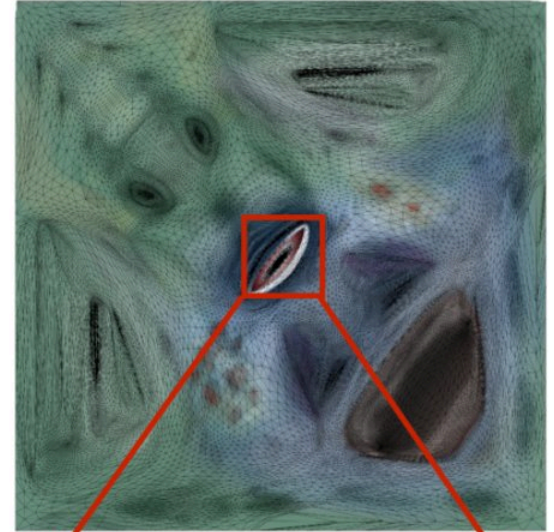
rendering without texture



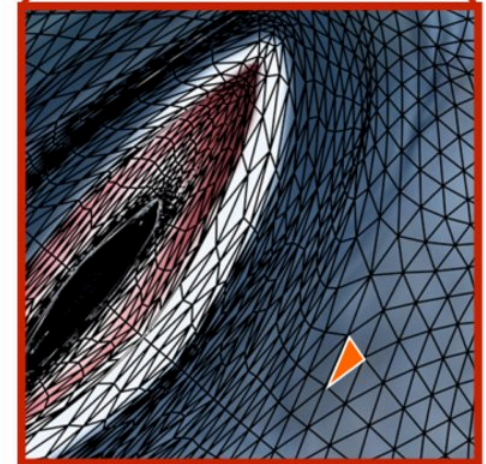
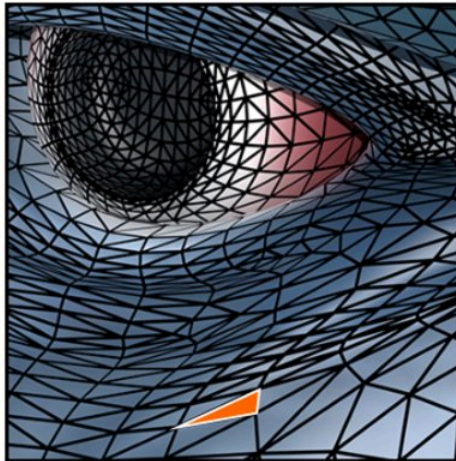
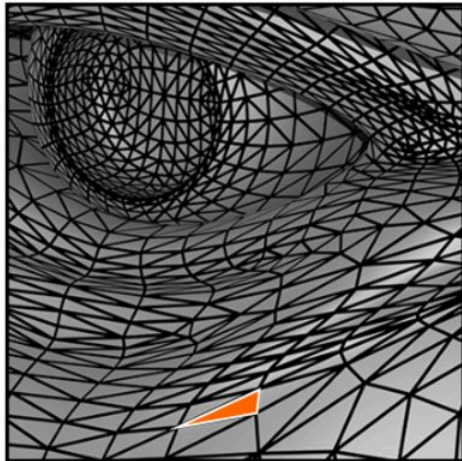
rendering with texture



texture image

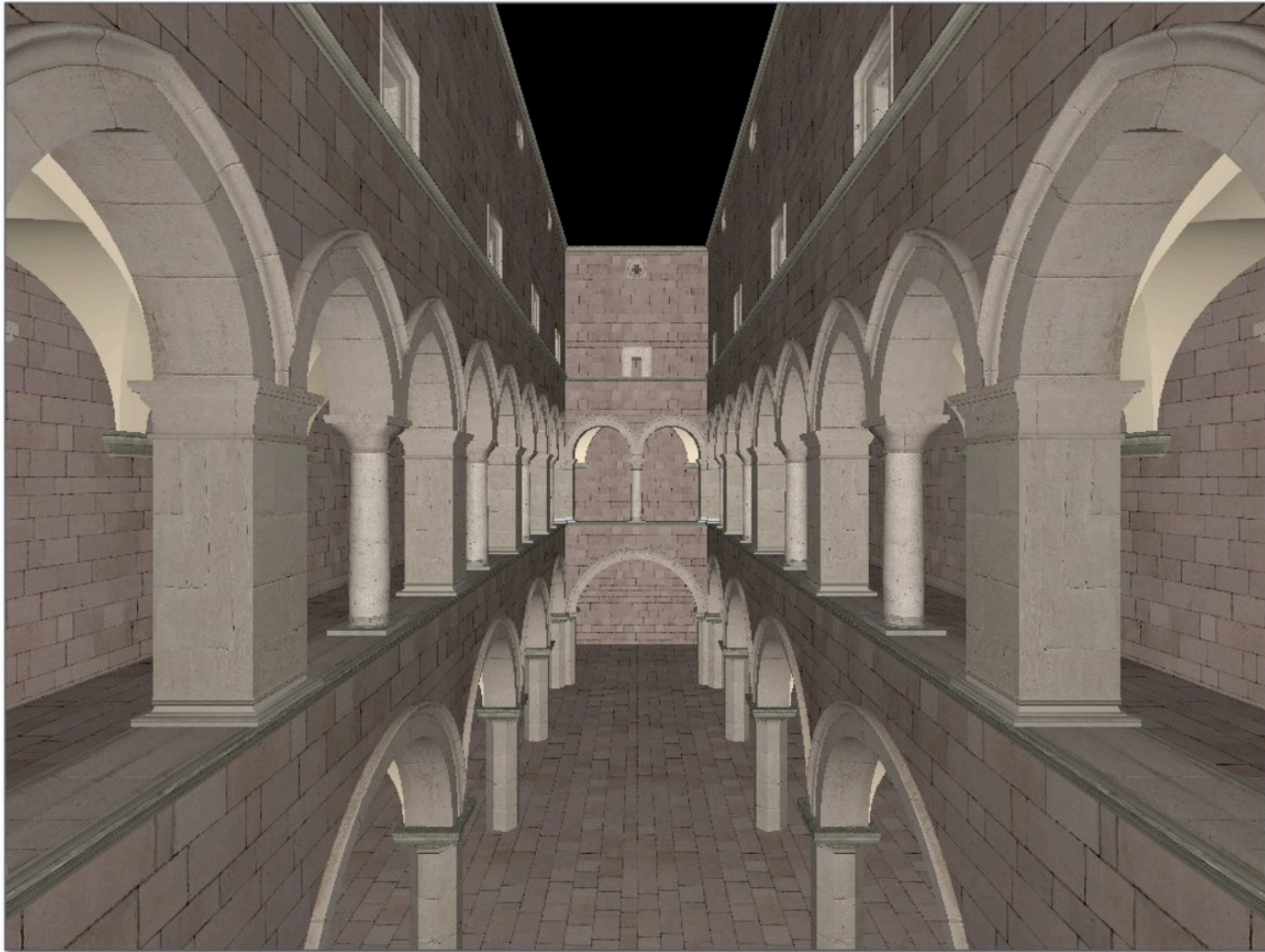


zoom

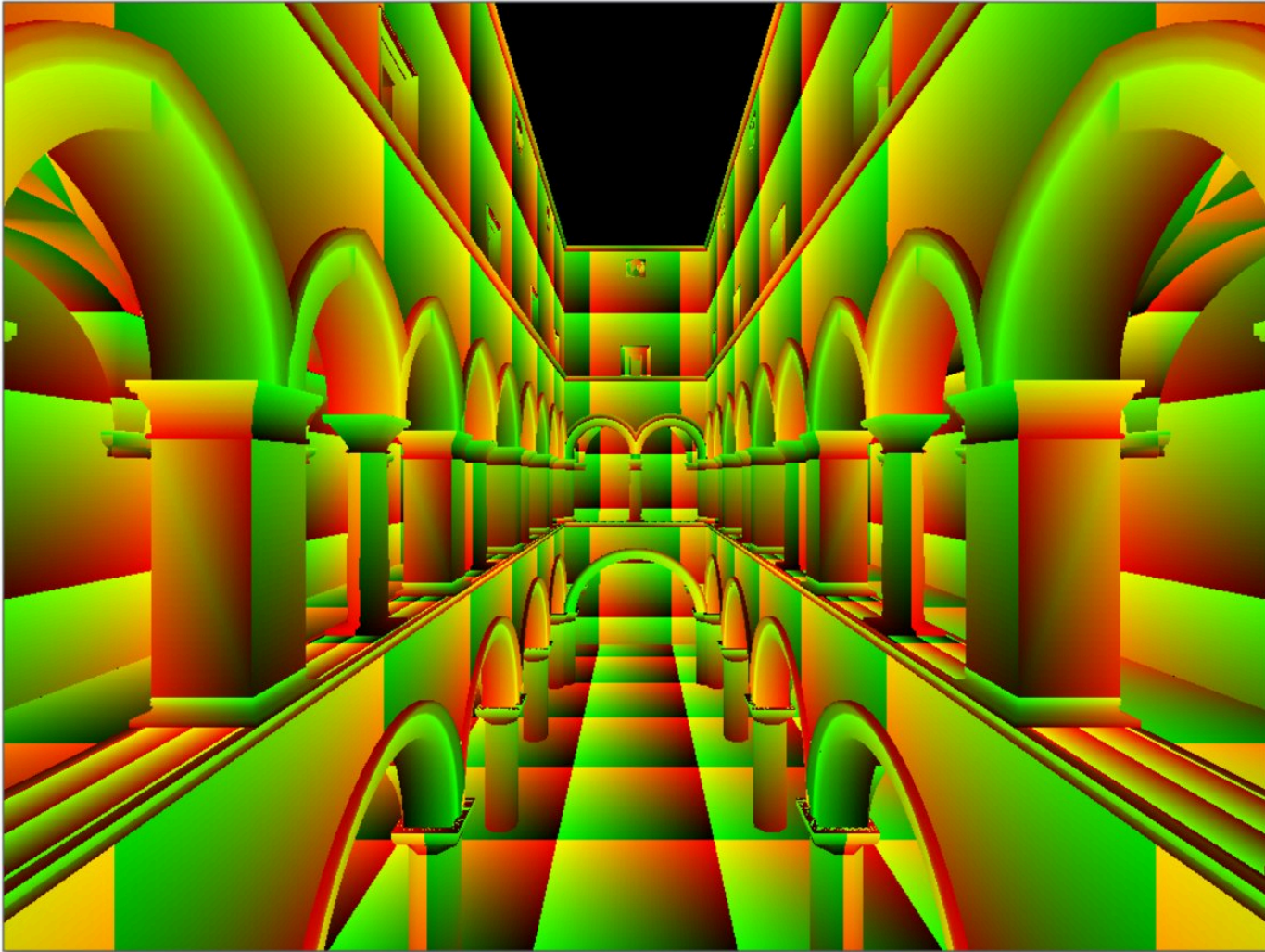


Each triangle “copies” a piece of the image back to the surface.

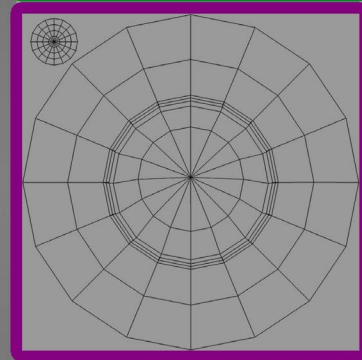
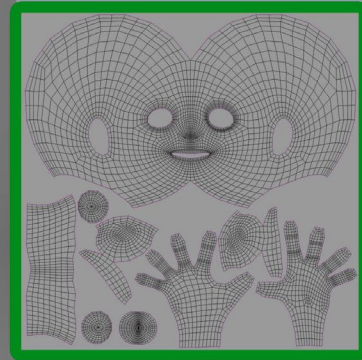
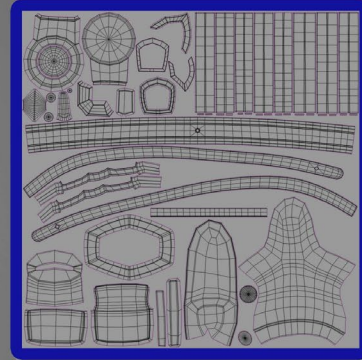
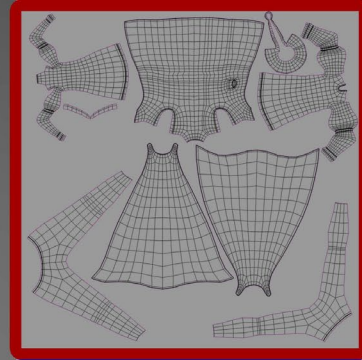
Textured Sponza



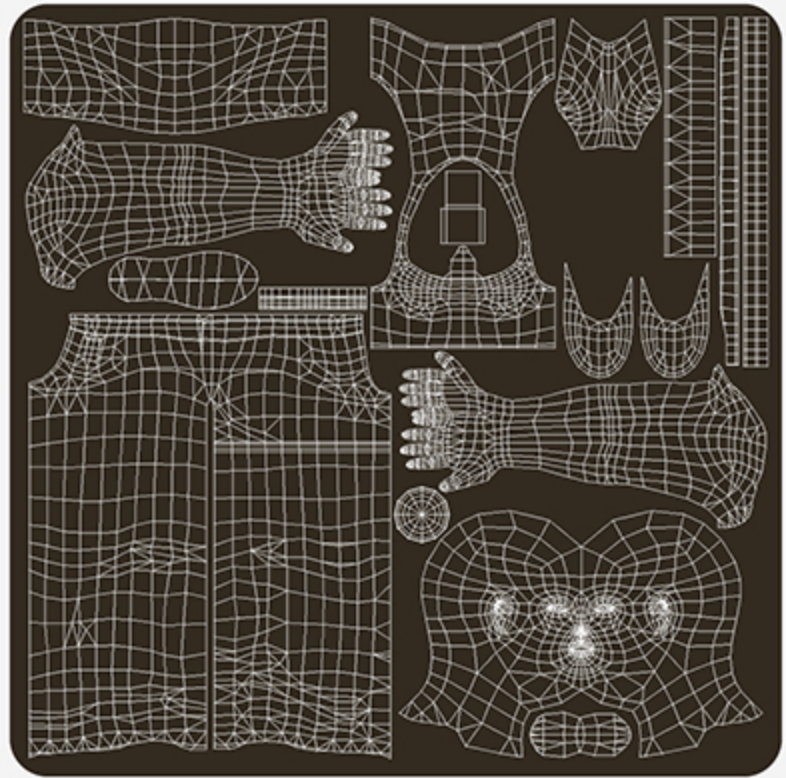
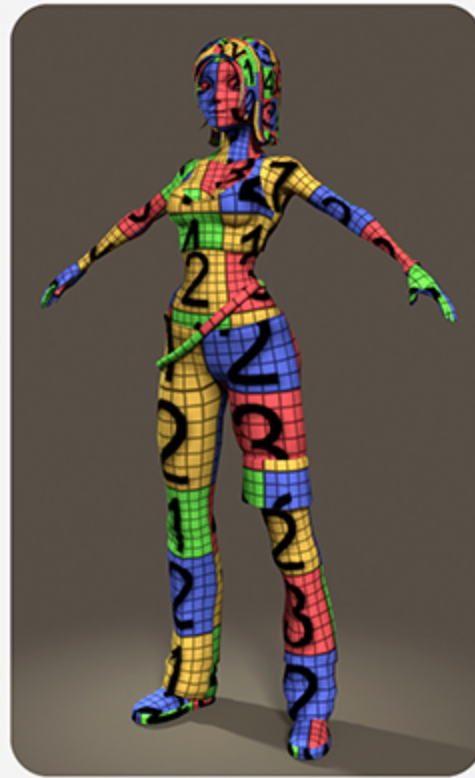
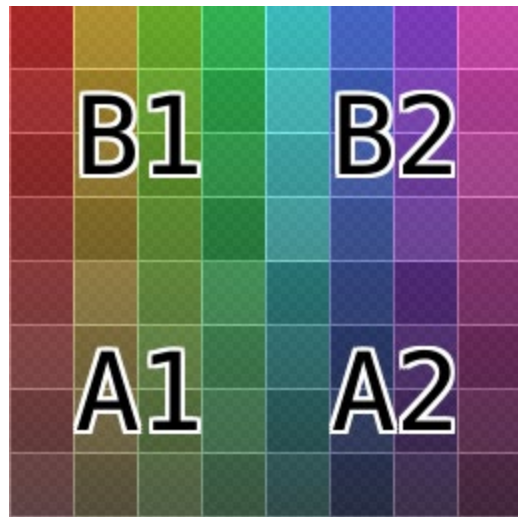
Another example: Sponza



Notice texture coordinates repeat over surface.

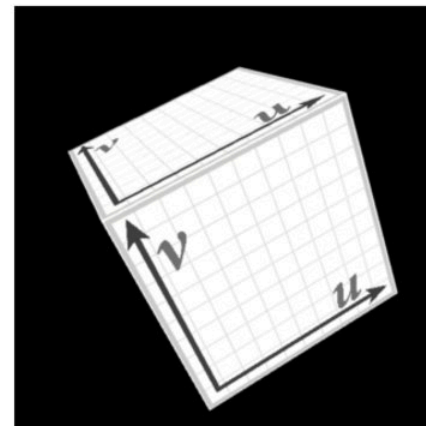
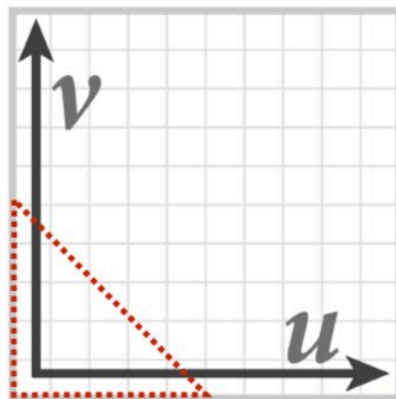


Debugging – Get an easy to understand texture



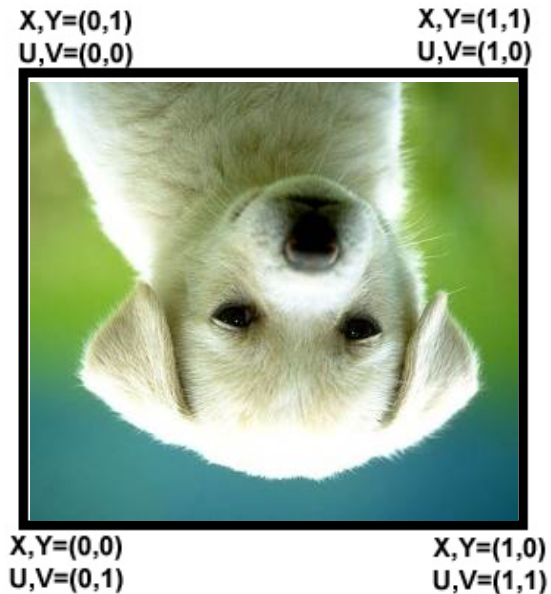
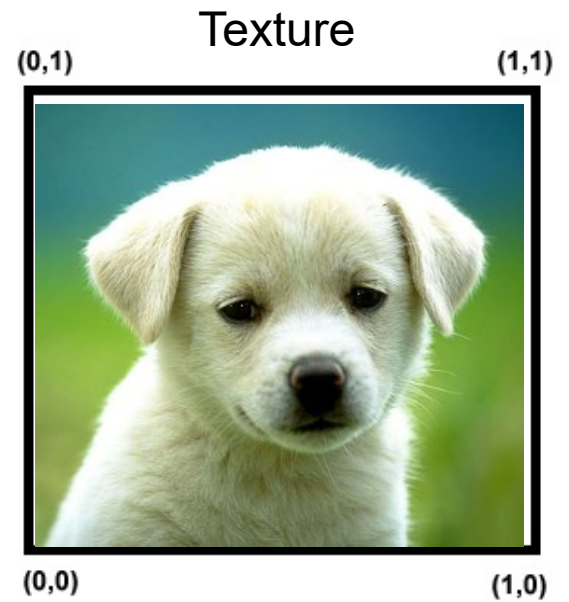
Summary

- Basic algorithm for mapping texture to surface:
 - For each color sample location (X,Y)
 - Interpolate U and V coordinates across triangle to get value at (X,Y)
 - Sample (evaluate) texture at (U,V)
 - Set color of fragment to sampled texture value

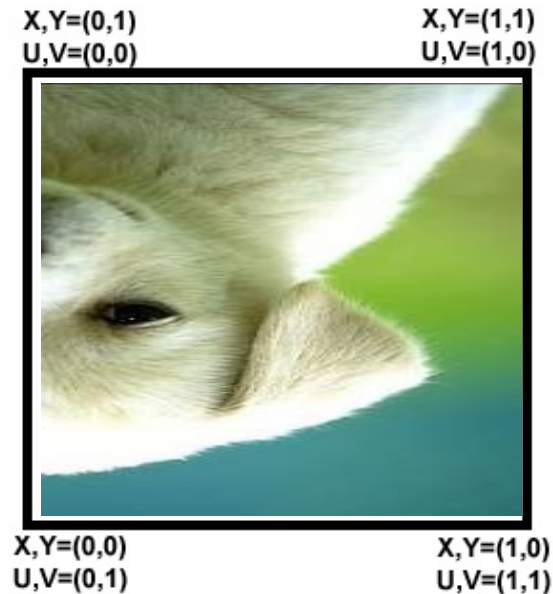


...sadly not this easy in general!

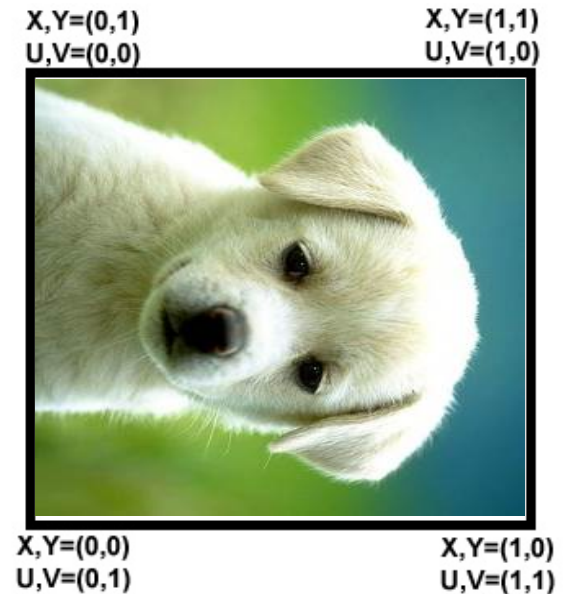
Which polygon below is textured correctly?



(A)



(B)



(C)

Where do UV come from?

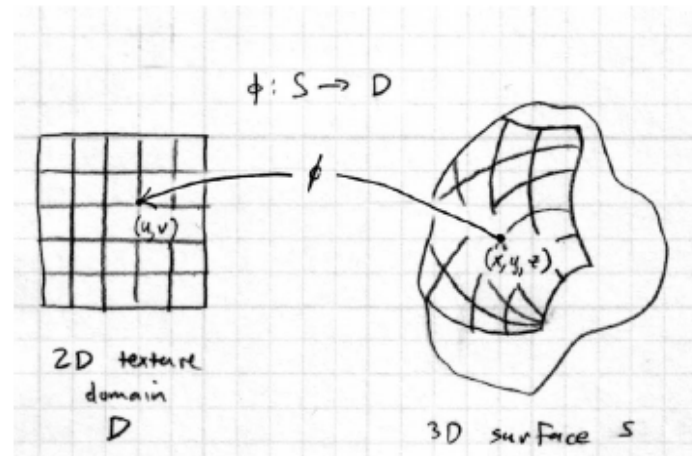
Texture coordinate functions

- Non-parametrically defined surfaces: more to do
 - can't assign texture coordinates as we generate the surface
 - need to have the *inverse* of the function f

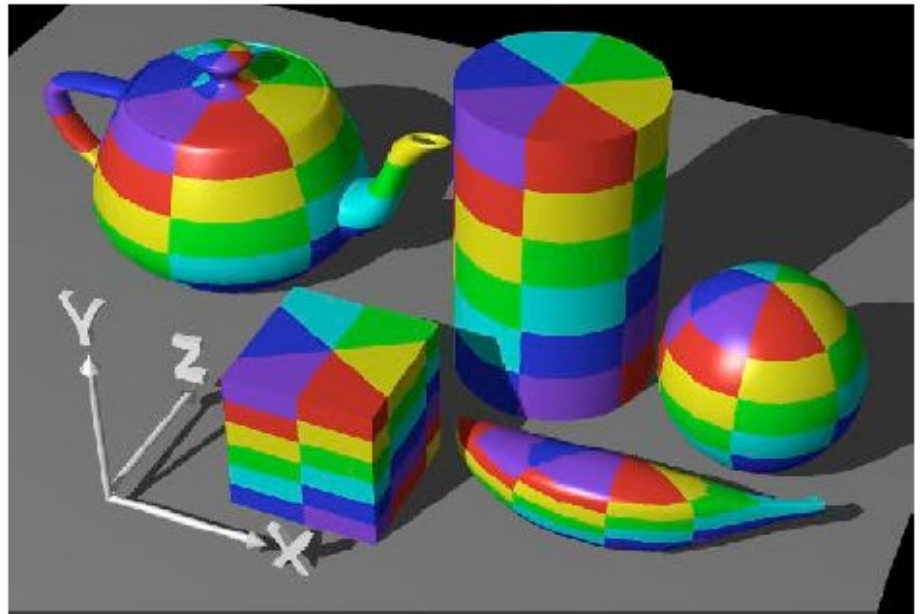
- Texture coordinate fn.

$$\phi: S \rightarrow \mathbb{R}^2$$

- for a vtx. at $\mathbf{p}$
get texture at
 $f(\mathbf{p})$



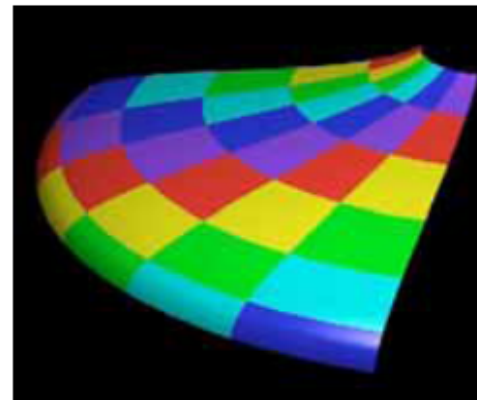
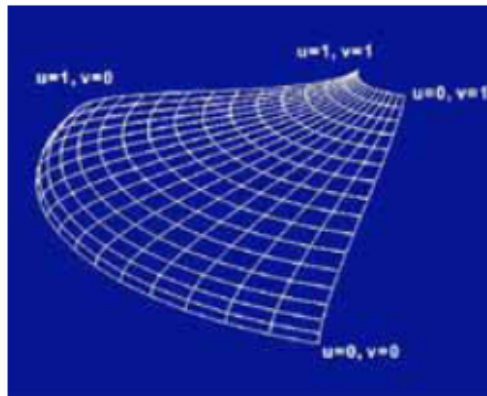
Cylindrical Parameterization



$$f : (x, y, z) \rightarrow (r, \theta, h) \rightarrow (u_\theta, v_h)$$

Examples of coordinate functions

- A parametric surface (e.g. spline patch)
 - surface parameterization gives mapping function directly
(well, the inverse of the parameterization)

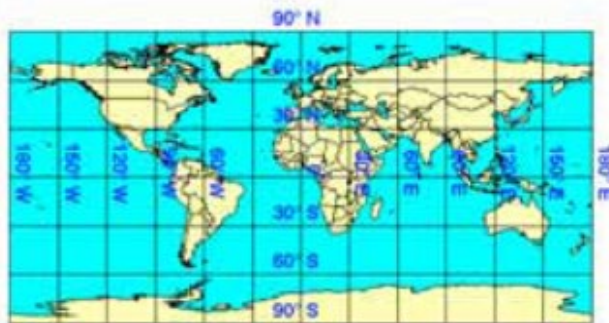


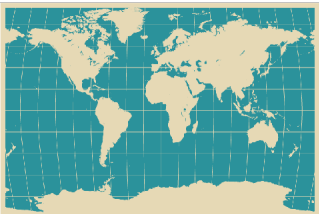
[Wolfe / SG97 Slide set]

Examples of coordinate functions

- For a sphere: latitude-longitude coordinates
 - f maps point to its latitude and longitude

[map: Peter H. Dana]





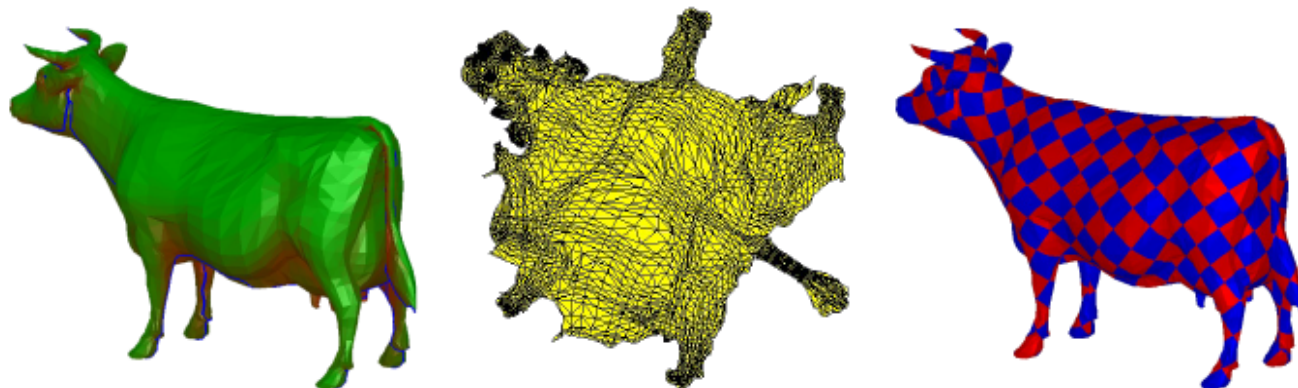
Parameterization

Parameterizing Polygonal Models

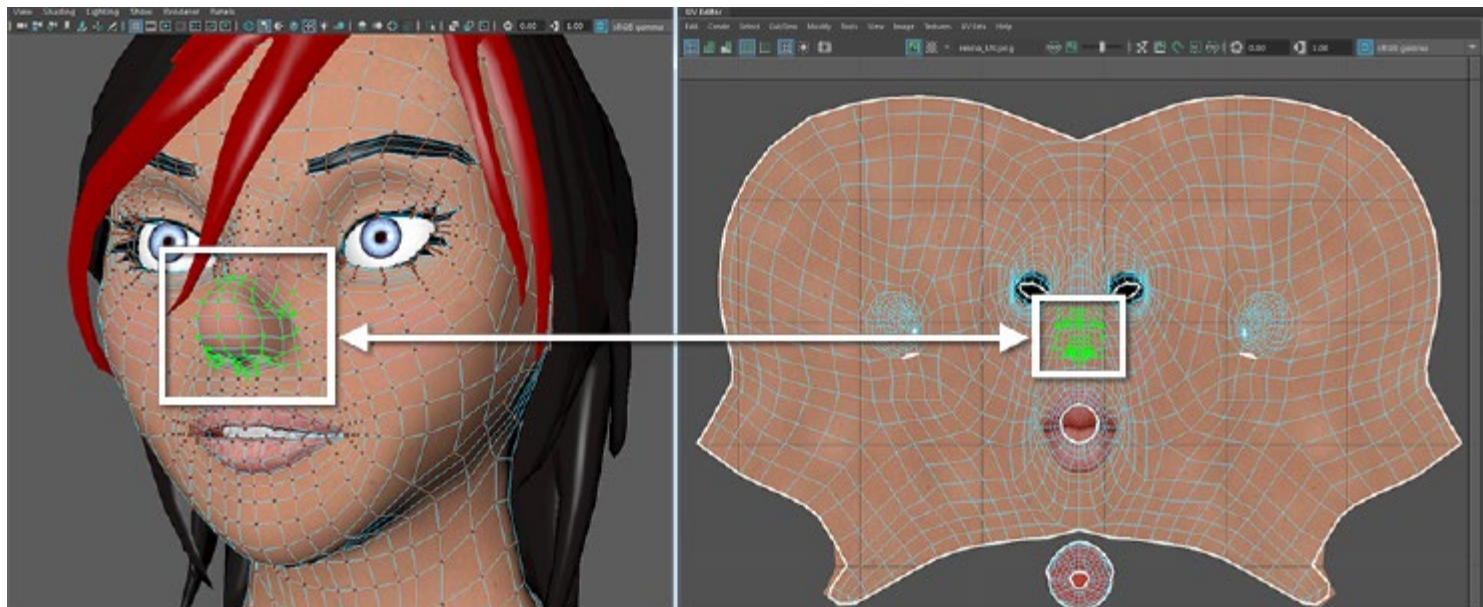
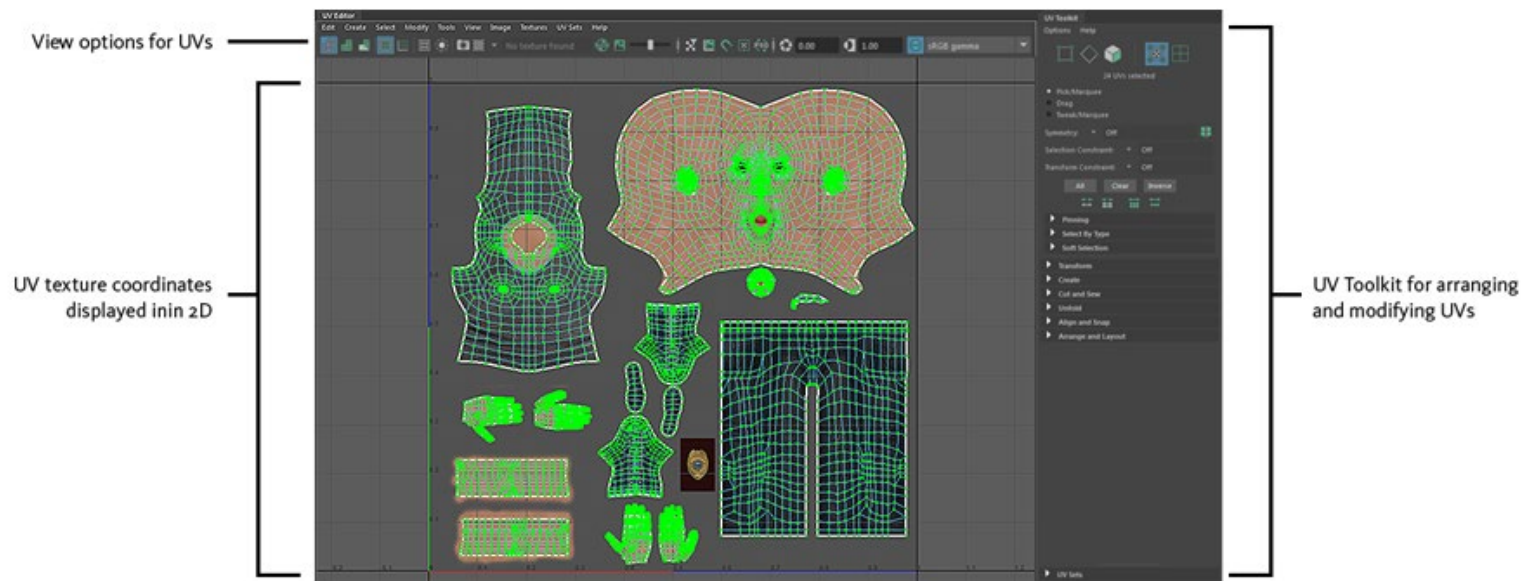
- Need mapping from vertices to **texels** (texture elements):

$$f : (x, y, z) \rightarrow (u, v)$$

- Most 3D geometric models don't have natural 2D parameterizations
- Creating these parameterizations can be difficult:
 - Must avoid parameter-space distortion
 - Have to handle seams and cracks
 - Should minimize wasted texture memory
- Use natural projections (planar, spherical, cylindrical, etc.) when possible



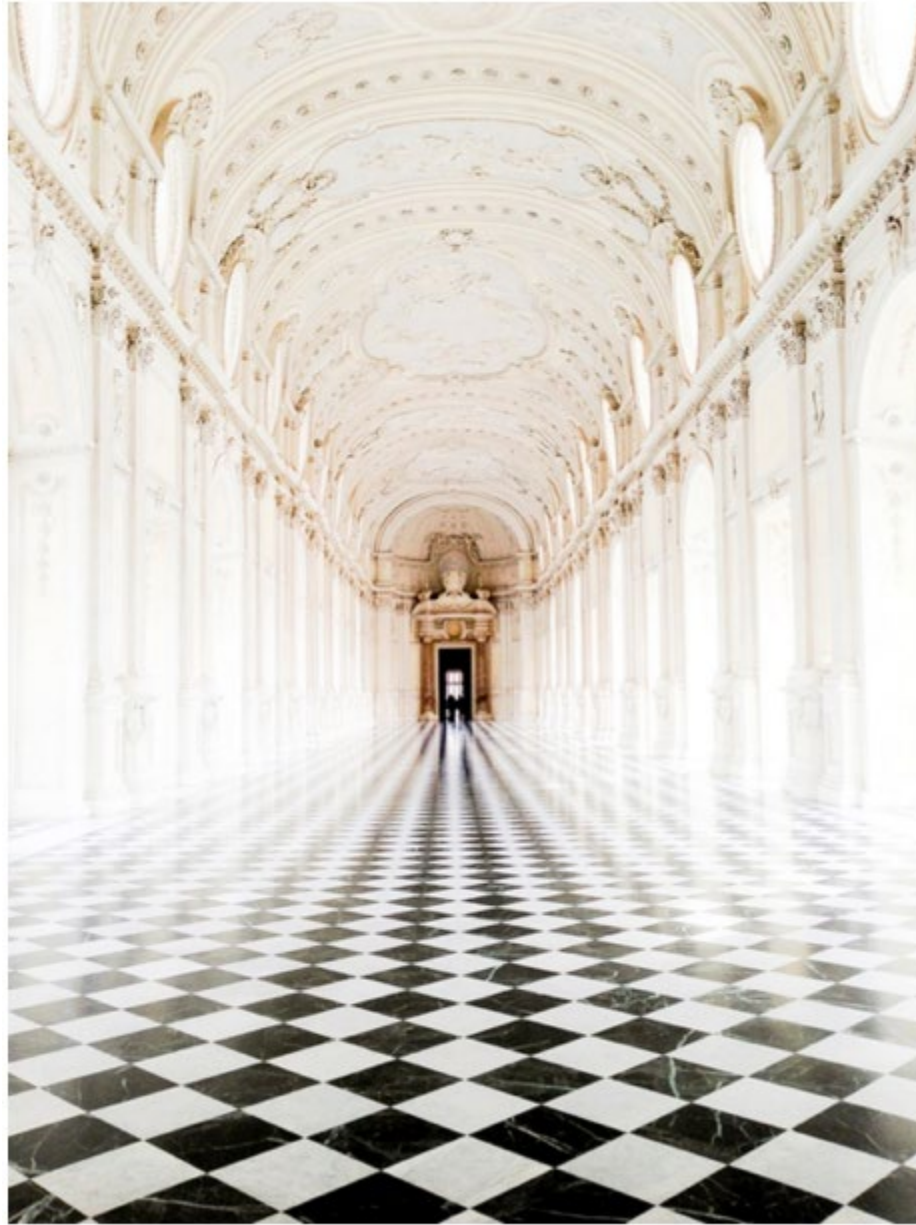
UV Editor in artist tool (Maya)



Perspective View

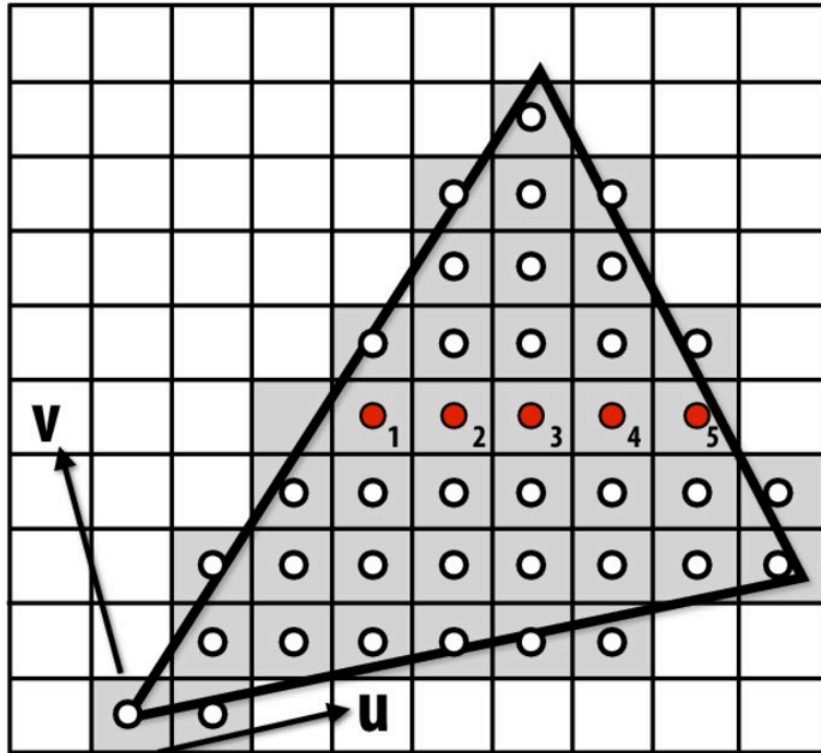
UV Editor View

Texture sampling NOT 1:1



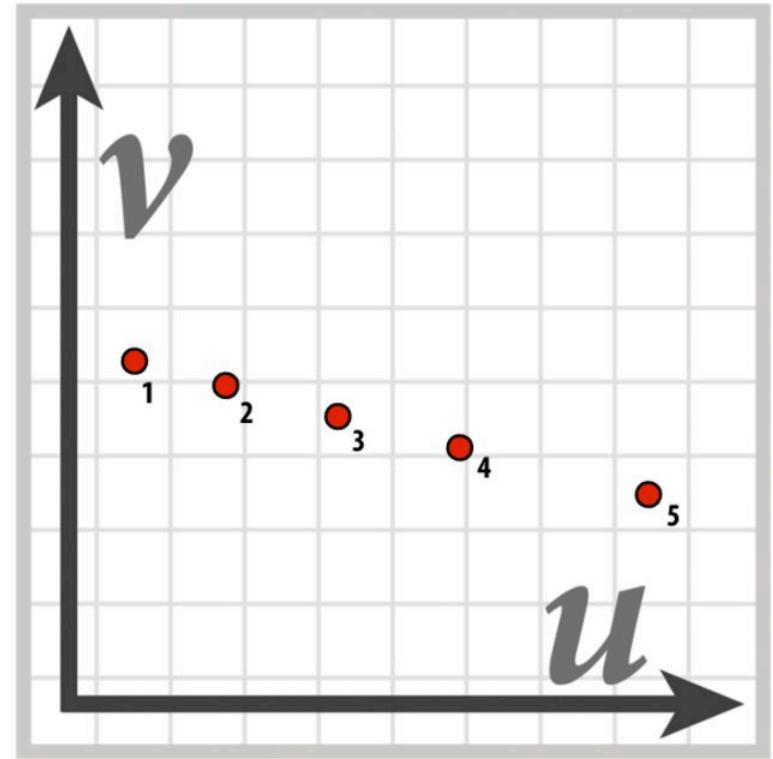
Texture space samples

Sample positions in XY screen space

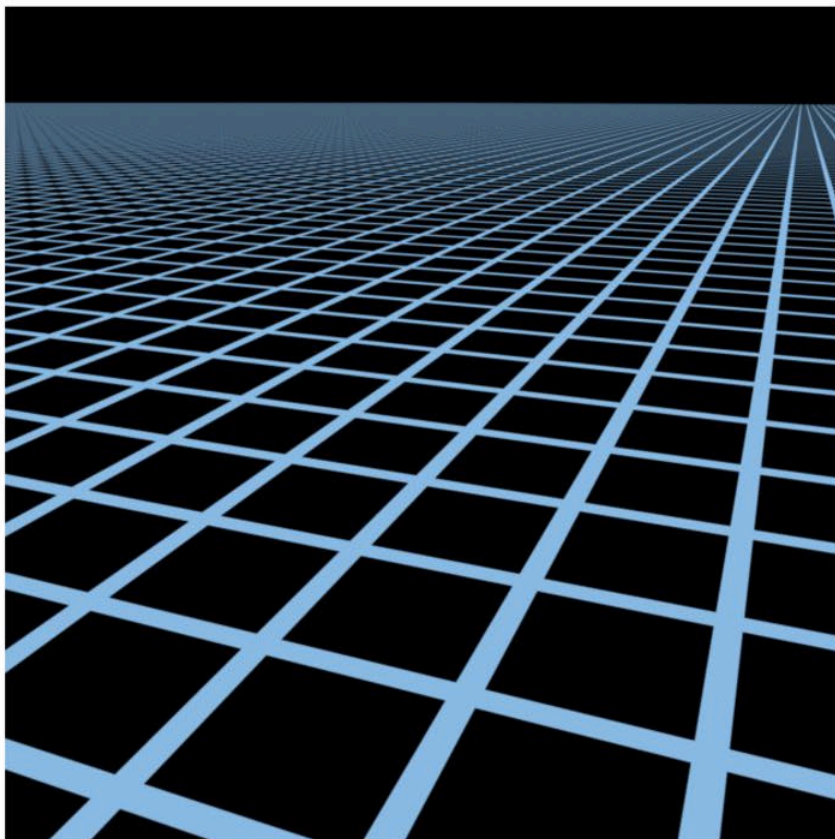


Sample positions are uniformly distributed in screen space (rasterizer samples triangle's appearance at these locations)

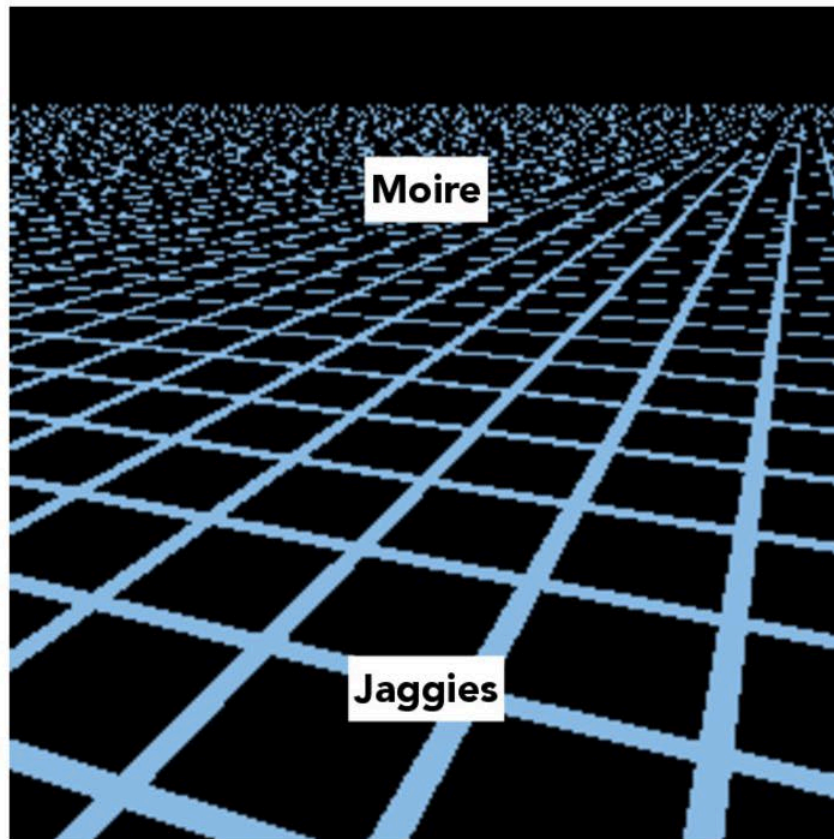
Sample positions in texture space



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



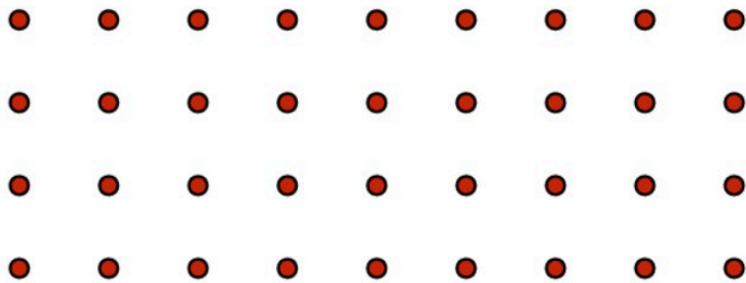
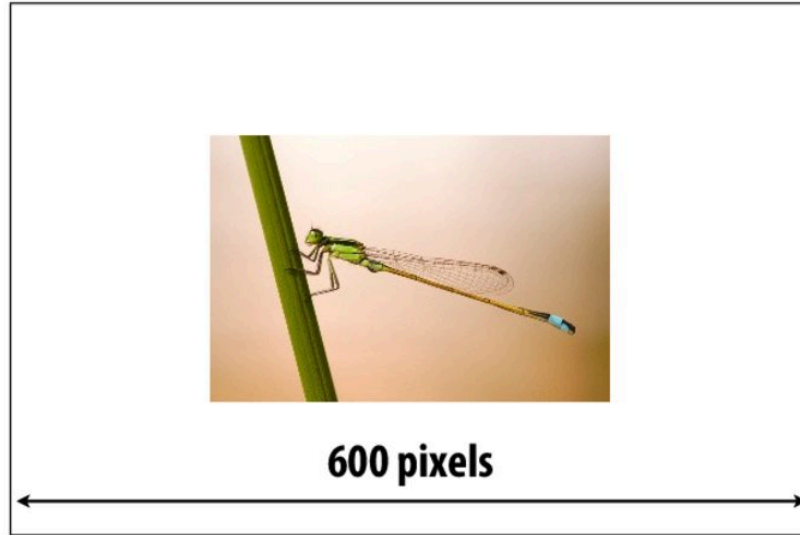
Source image: 1280x1280 pixels



Rendered image: 256x256 pixels

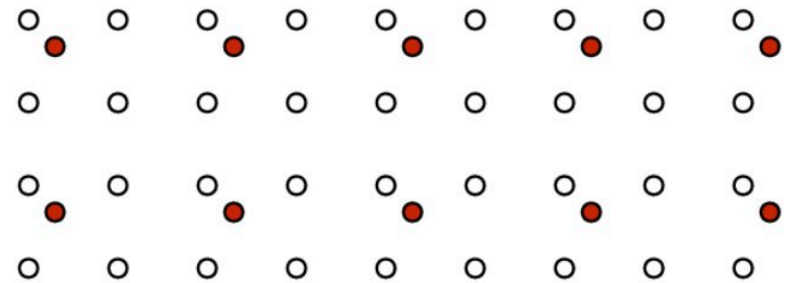
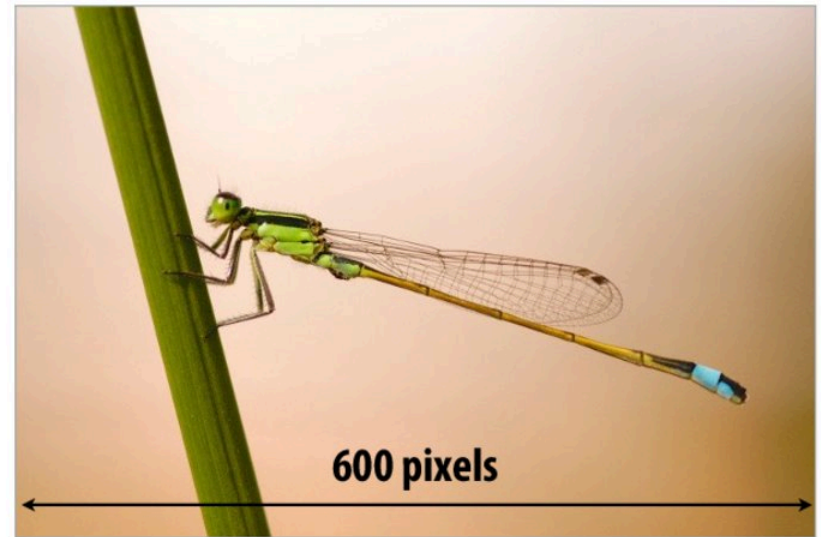
Sampling rate on screen vs texture

Rendered image (object zoomed out)



Screen space (x,y)

Texture Image



Texture space (u,v)

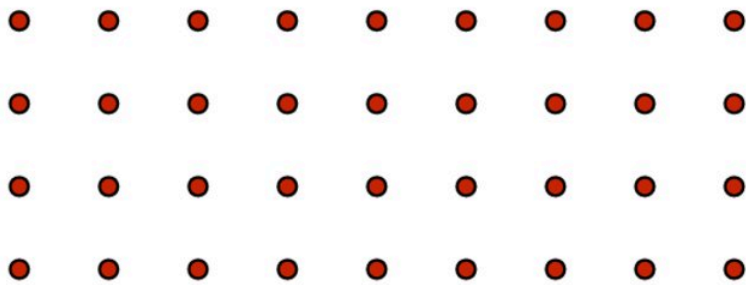
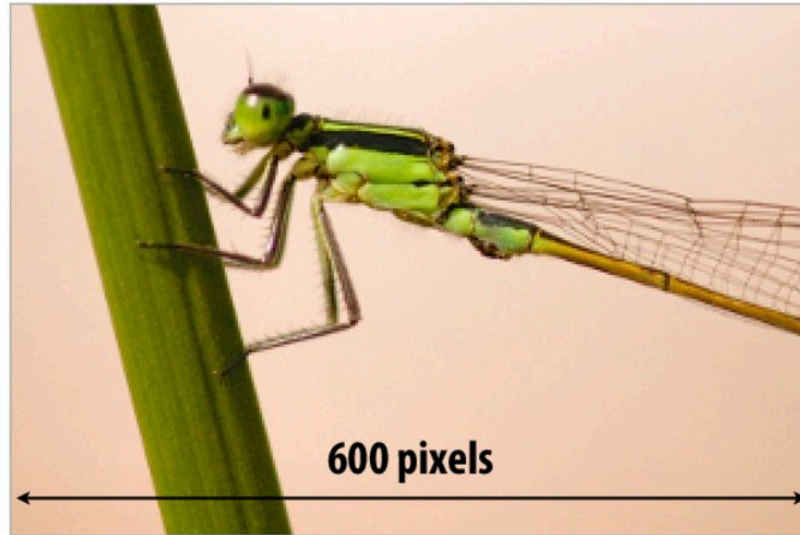
Red dots = samples needed to render

White dots = samples existing in texture map

Texture is "minified"

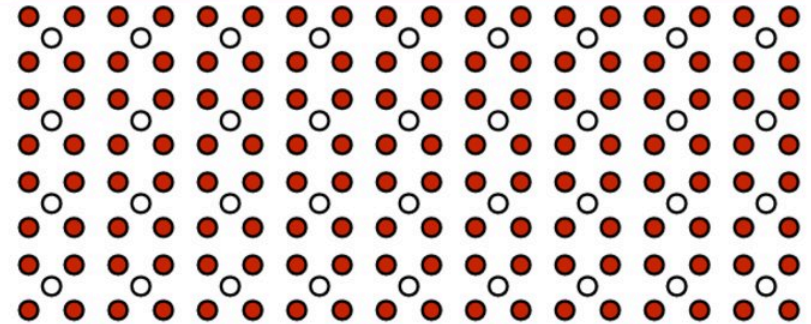
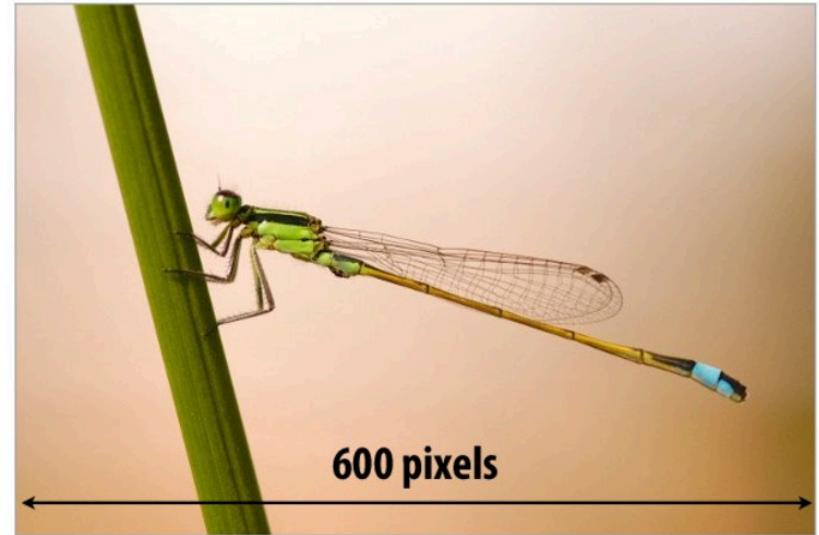
Sampling rate on screen vs texture

Rendered image (zoomed in)



Screen space (x,y)

Texture Image



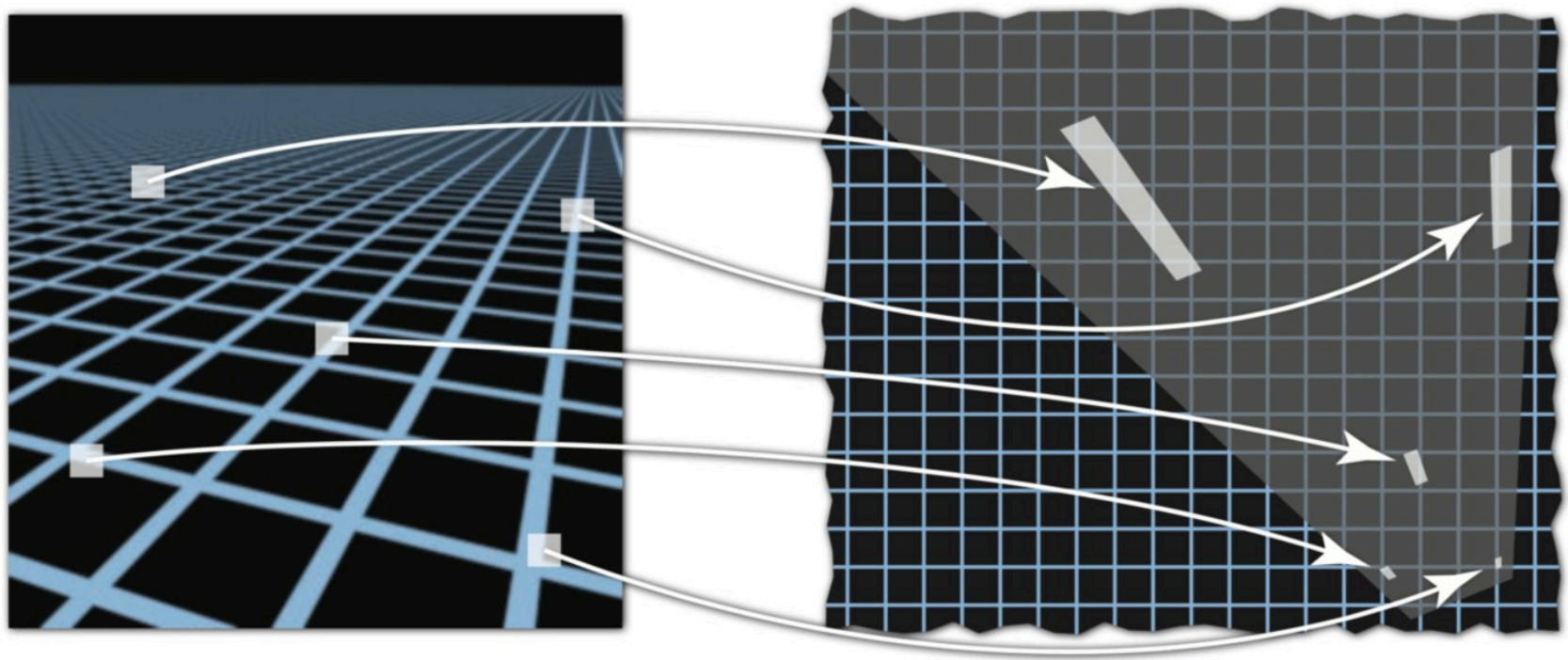
Texture space (u,v)

Texture is "magnified" on screen

Red dots = samples needed to render

White dots = samples existing in texture map

Screen pixel footprint in texture space

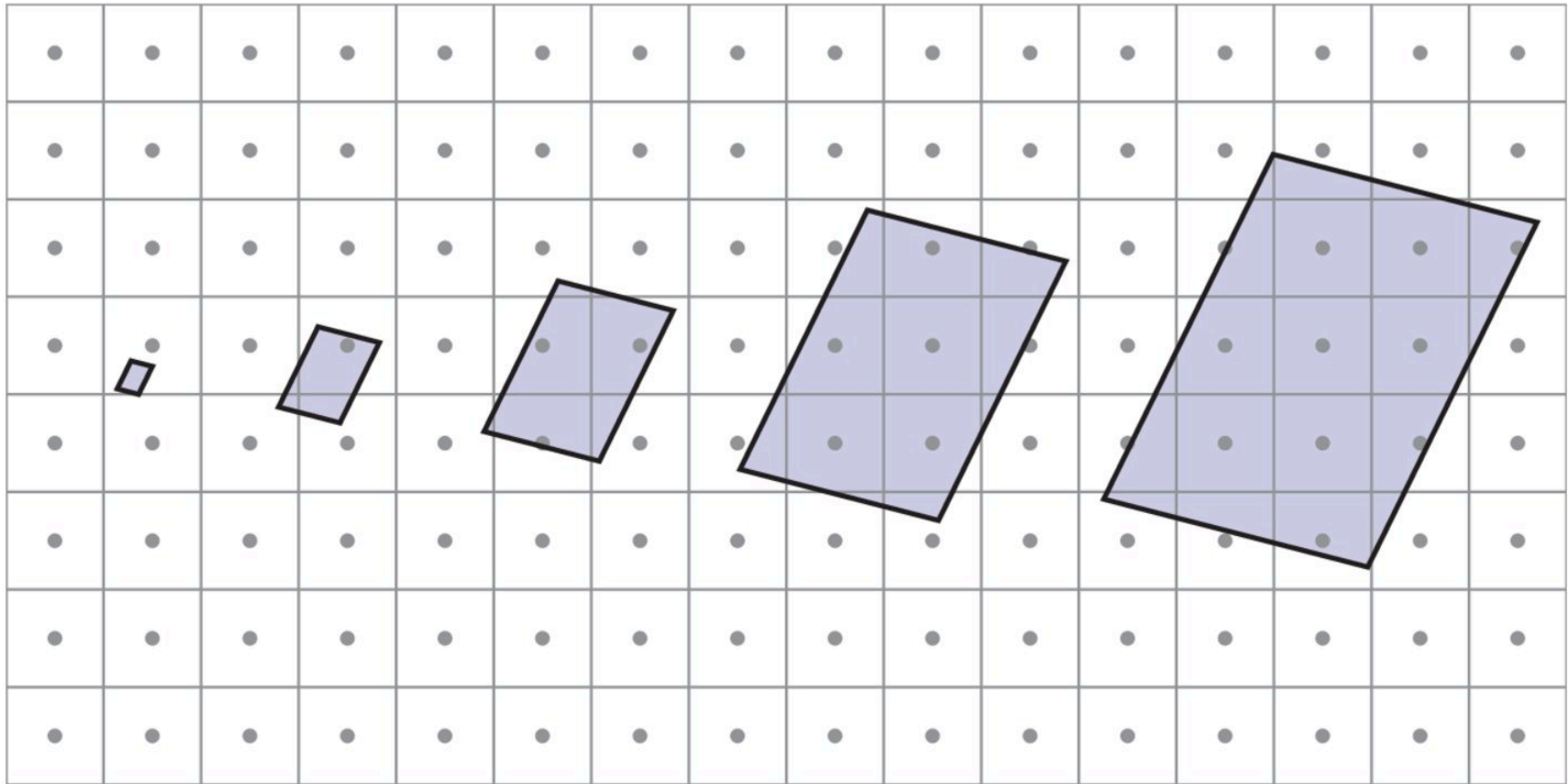


Screen space

Texture space

Texture sampling pattern not rectilinear or isotropic

Screen pixel footprint in texture space



**Upsampling
(Magnification)**

*Camera zoomed in
close to object*

**Downsampling
(Minification)**

*Camera far away
from object*

http://tiny.cc/160-1029

Participation Apr 28

Form description

This form is automatically collecting email addresses for UC Santa Cruz users. [Change settings](#)

I was in class Apr 28

- ☐ Yes
- ☐ No

Roughly how long did you spend on [A2](#) ([Blocky](#) Animal)

- ☐ 0-5 hours
- ☐ 5-10 hours
- ☐ 10-15 hours
- ☐ 15+ hours

There were YouTube videos to help with coding for [A2](#):

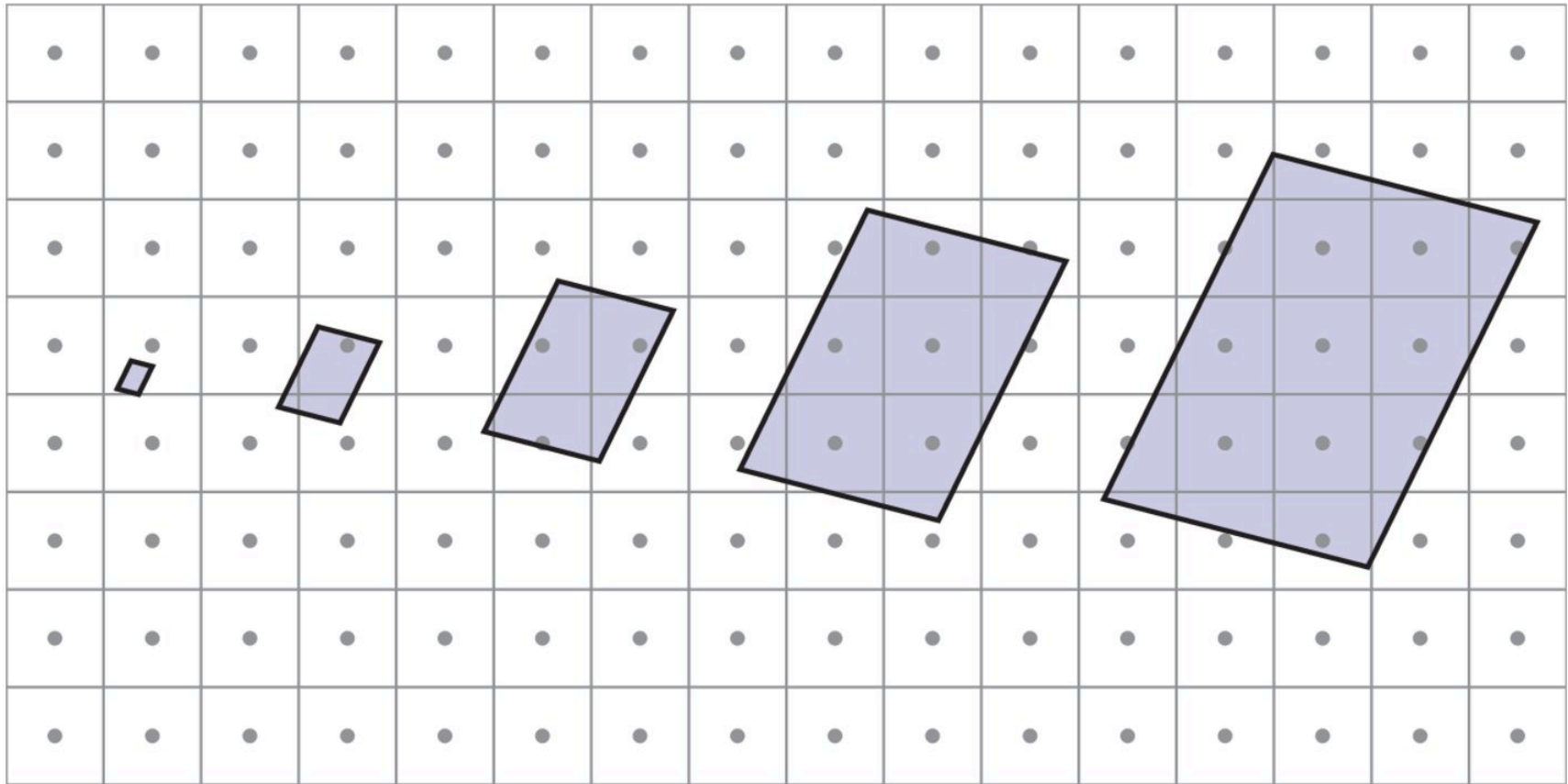
Suggestions: [Add all](#) | [Yes](#) [No](#) [Maybe](#)

- ☐ I didn't watch them, I just did the assignment
- ☐ I watched them, they were NOT helpful
- ☐ I watched them, they WERE helpful
- ☐ Other...

The textbook introduces the concepts needed for [Asg2](#). ([Matsuda WebGL](#)) Is this book good, or it should be replaced?

Magnification

Screen pixel footprint in texture space



**Upsampling
(Magnification)**

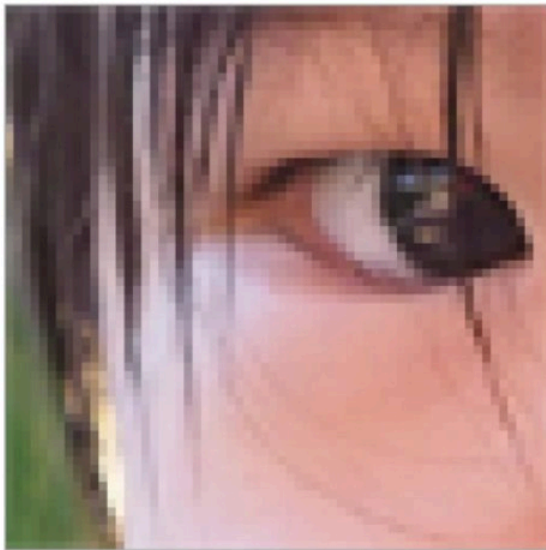
*Camera zoomed in
close to object*

**Downsampling
(Minification)**

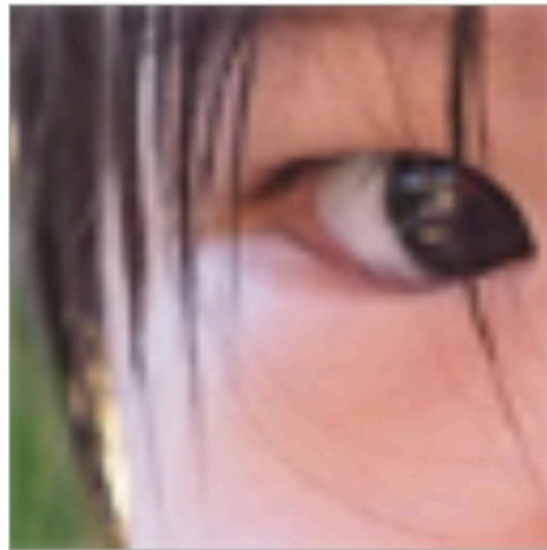
*Camera far away
from object*

Texture magnification - easy case

- Generally don't want this situation — it means we have insufficient texture resolution
- This is image interpolation (below: three different kernel functions)



Nearest



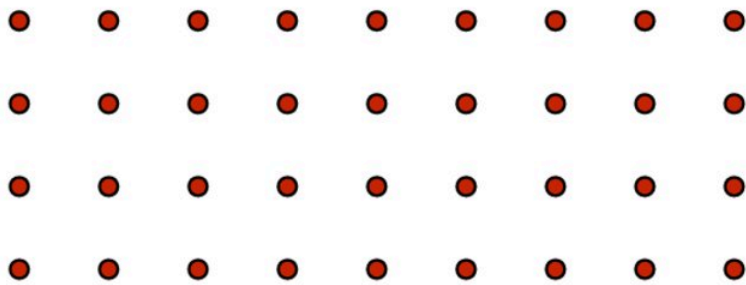
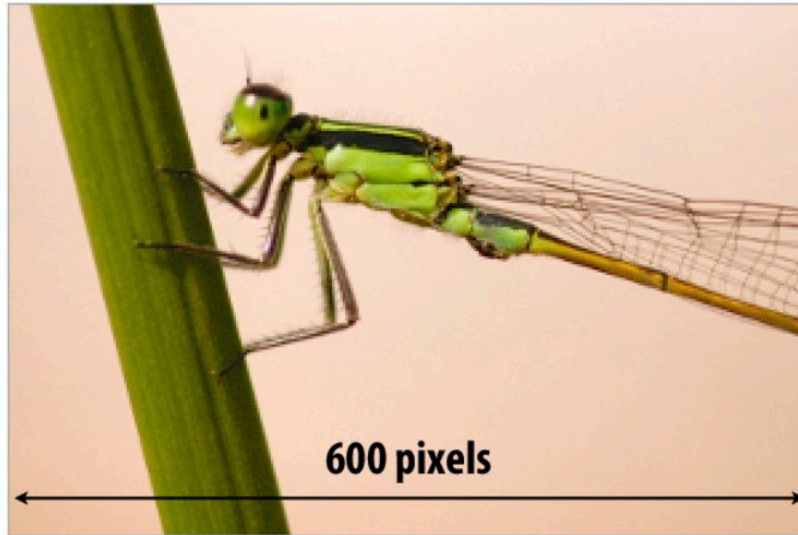
Bilinear



Bicubic

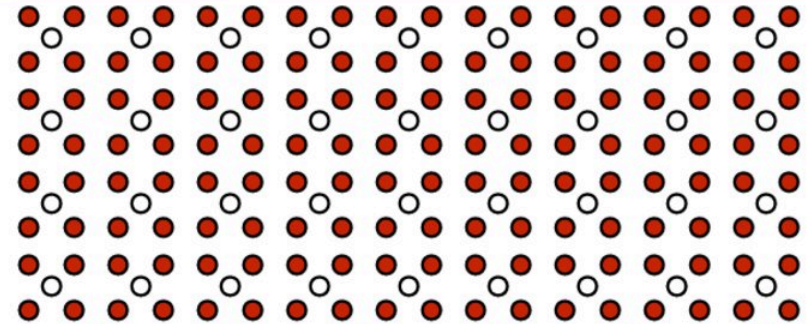
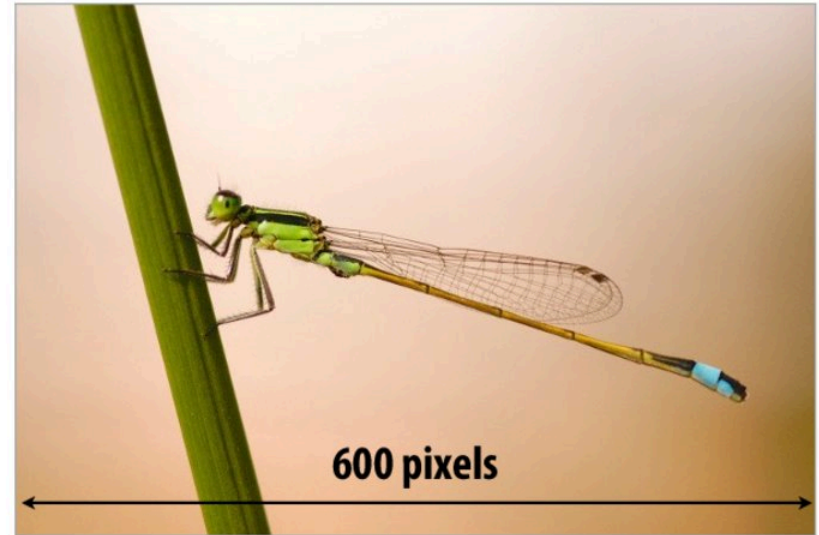
Sampling rate on screen vs texture

Rendered image (zoomed in)



Screen space (x,y)

Texture Image



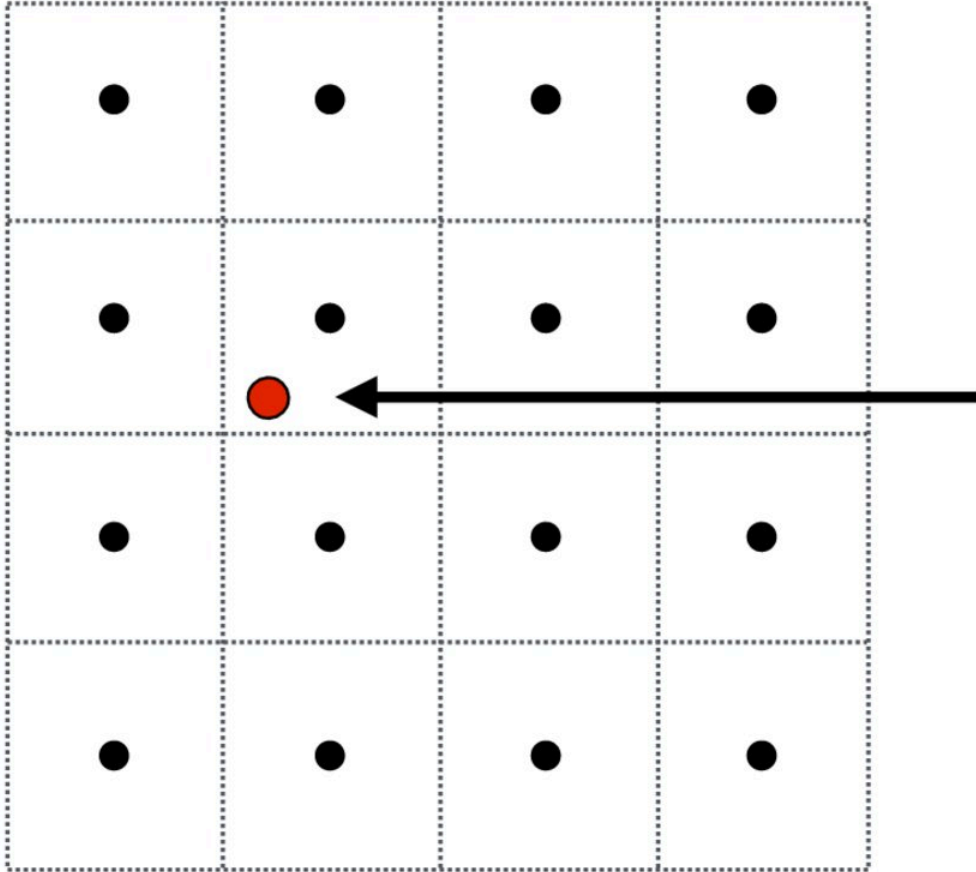
Texture space (u,v)

Texture is "magnified" on screen

Red dots = samples needed to render

White dots = samples existing in texture map

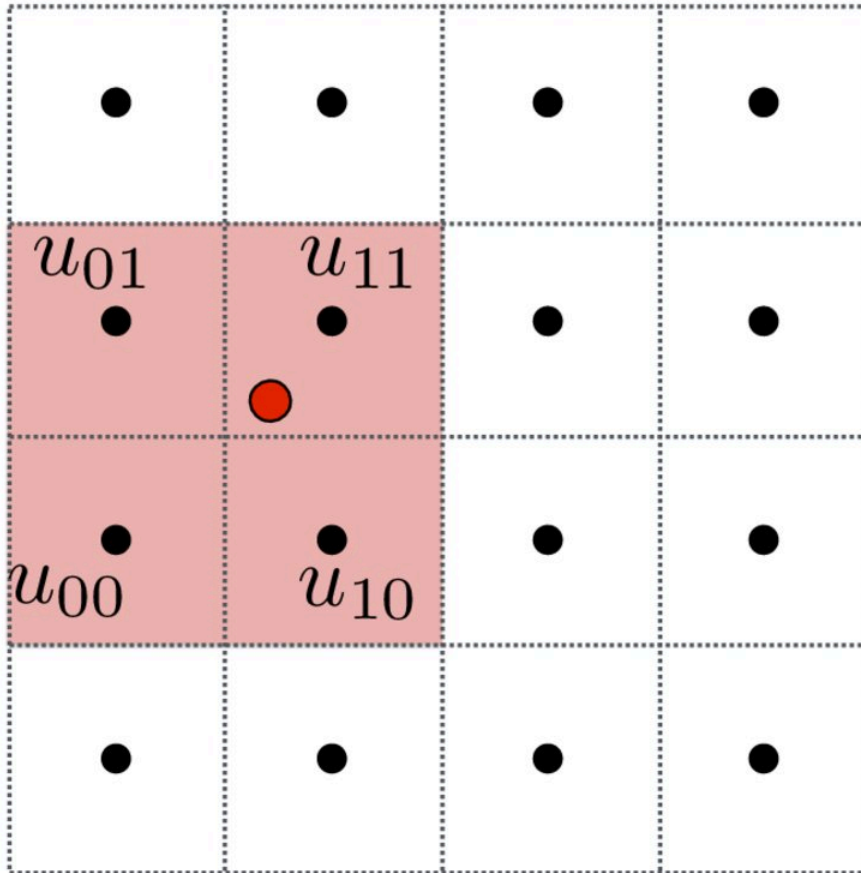
Bilinear filtering



**Want to sample texture
value $f(x,y)$ at red point**

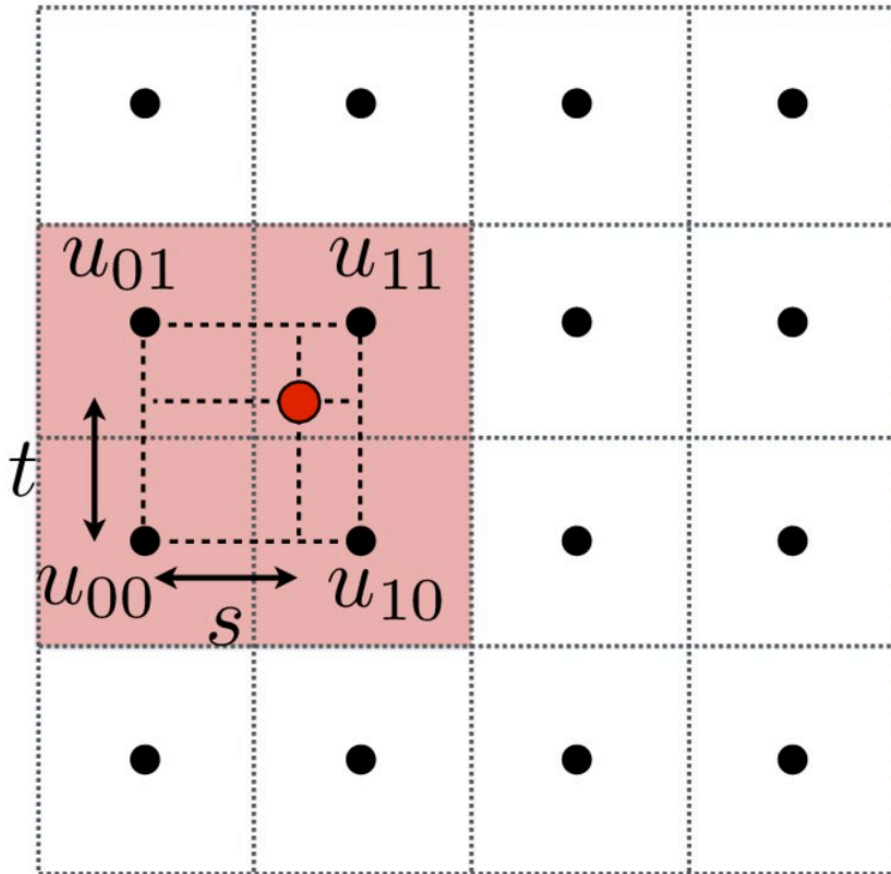
**Black points indicate texture
sample locations**

Bilinear filtering



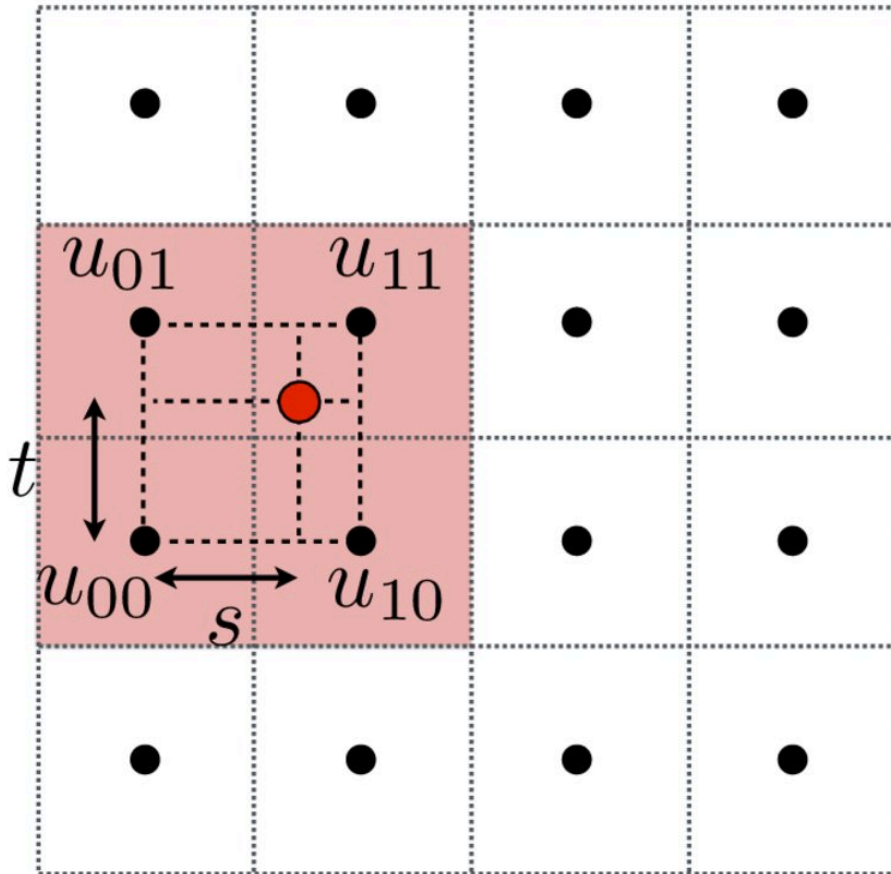
Take 4 nearest sample locations, with texture values as labeled.

Bilinear filtering



**And fractional offsets,
(s,t) as shown**

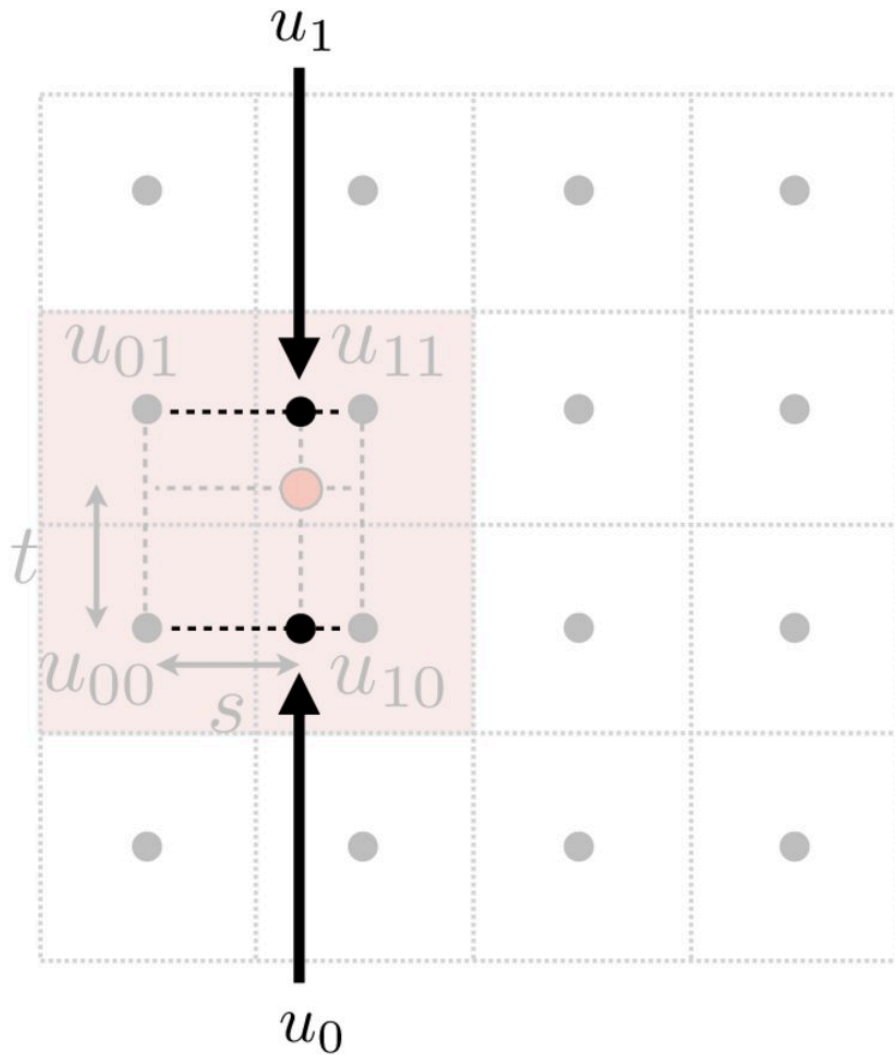
Bilinear filtering



Linear interpolation (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Bilinear filtering



Linear interpolation (1D)

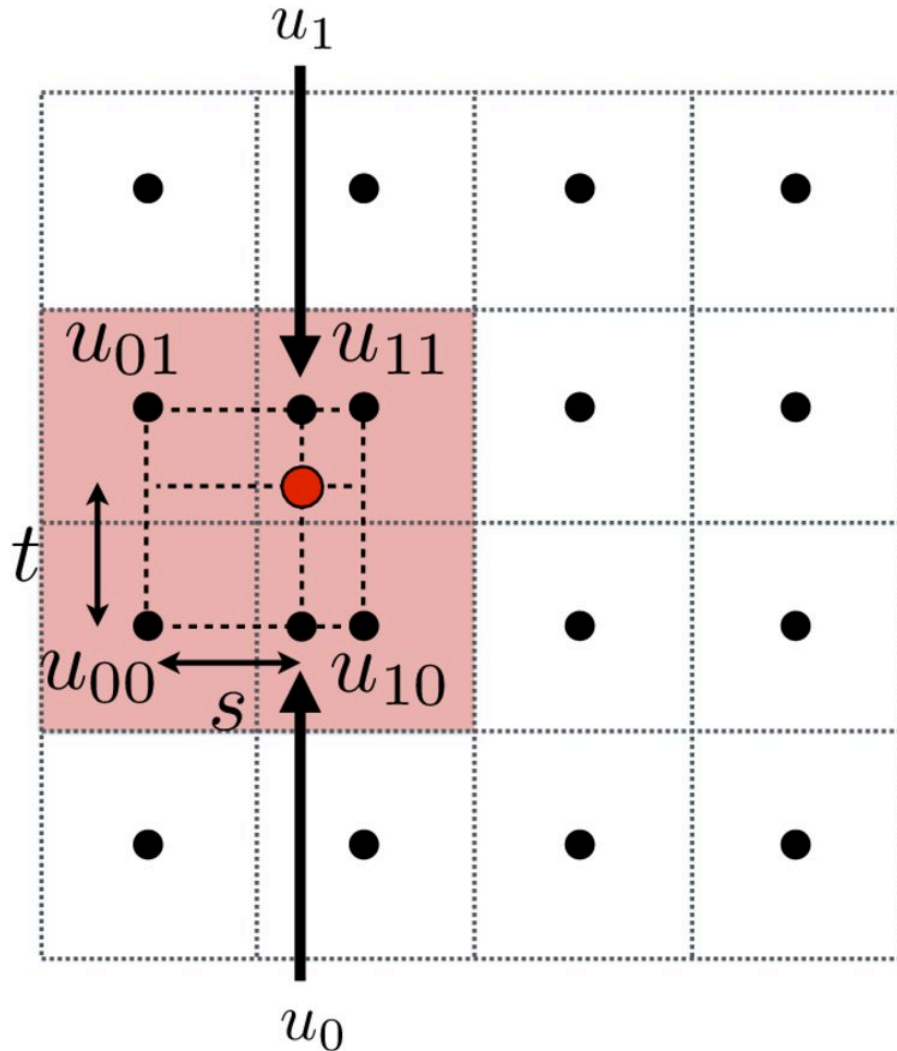
$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps (horizontal)

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

Bilinear filtering



Linear interpolation (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

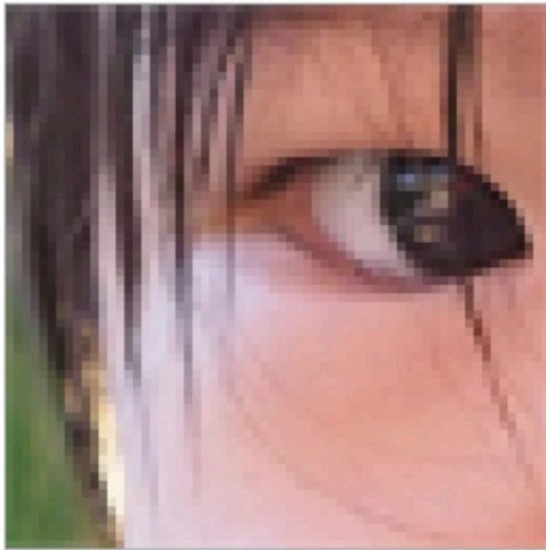
$$u_1 = \text{lerp}(t, u_{01}, u_{11})$$

Final vertical lerp, to get result:

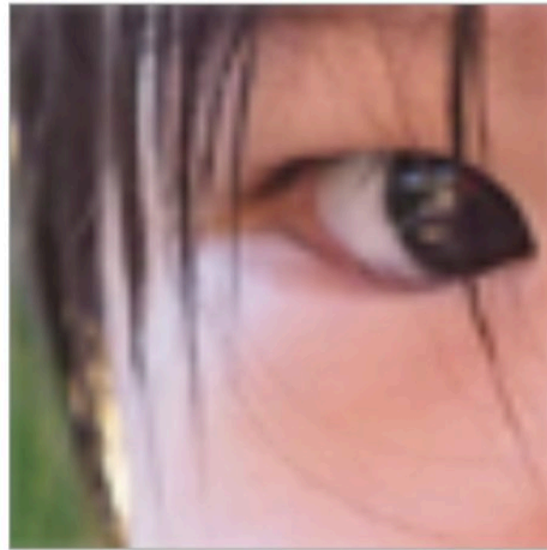
$$f(x, y) = \text{lerp}(t, u_0, u_1)$$

Texture magnification - easy case

- Generally don't want this situation — it means we have insufficient texture resolution
- This is image interpolation (below: three different kernel functions)



Nearest



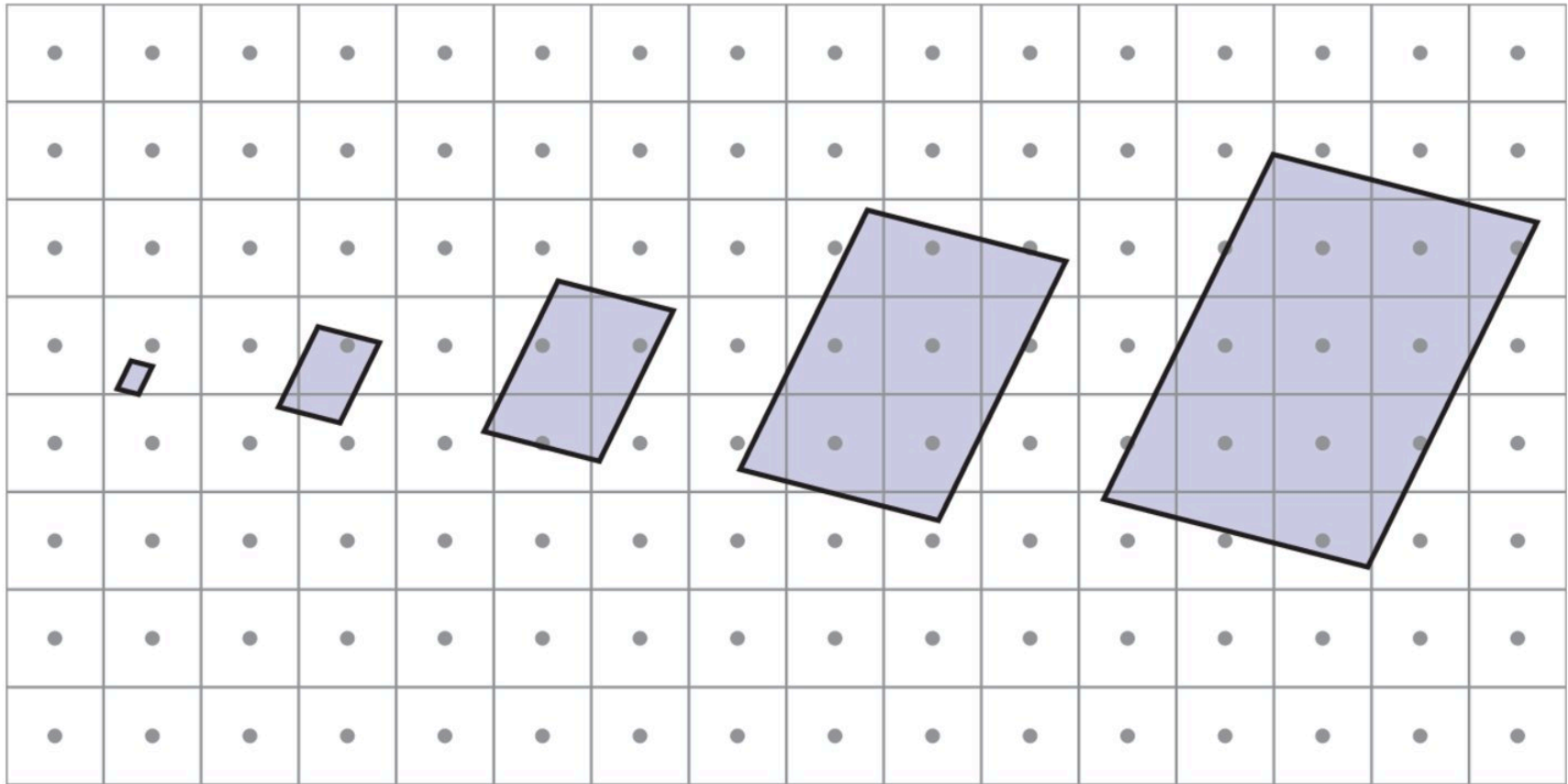
Bilinear



Bicubic

Minification

Screen pixel footprint in texture space



**Upsampling
(Magnification)**

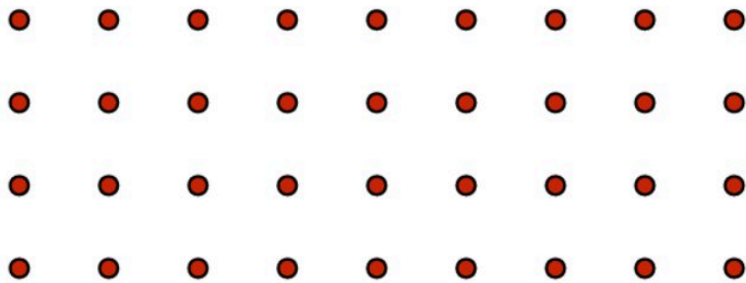
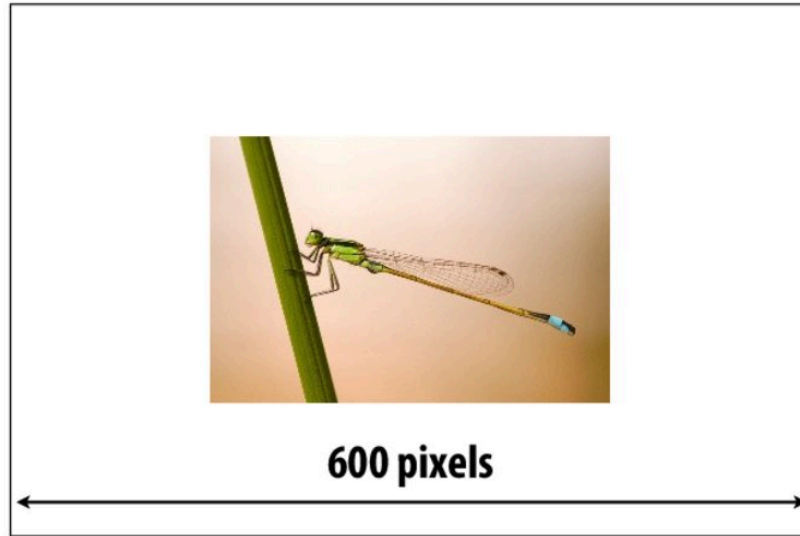
*Camera zoomed in
close to object*

**Downsampling
(Minification)**

*Camera far away
from object*

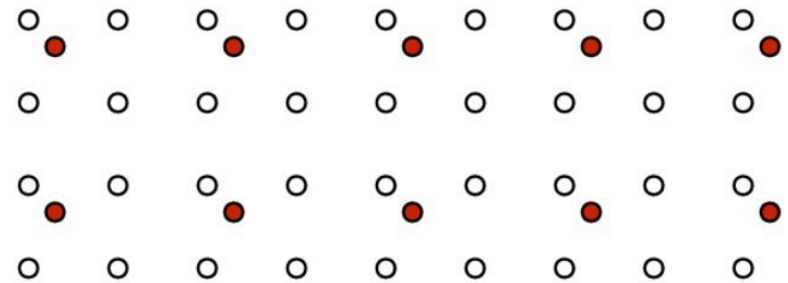
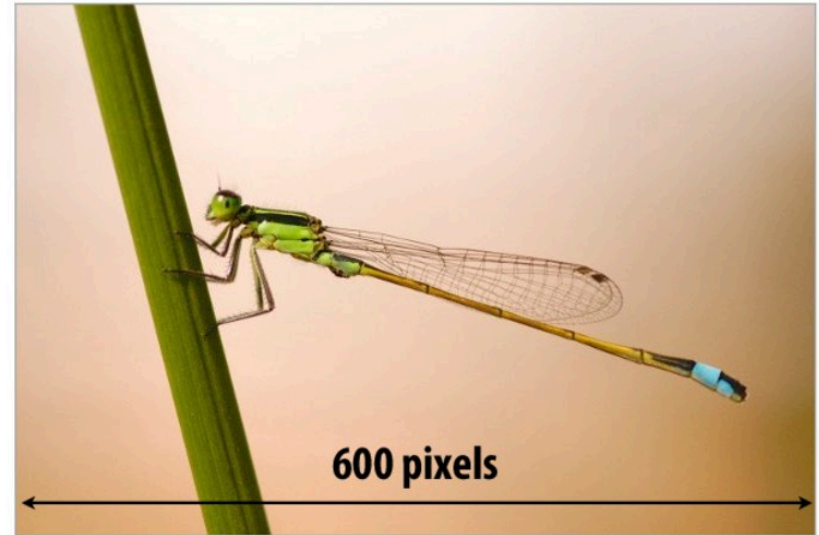
Sampling rate on screen vs texture

Rendered image (object zoomed out)



Screen space (x,y)

Texture Image



Texture space (u,v)

Red dots = samples needed to render

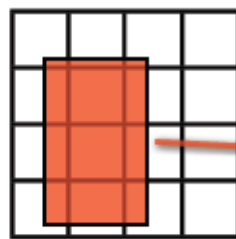
White dots = samples existing in texture map

Texture is "minified"

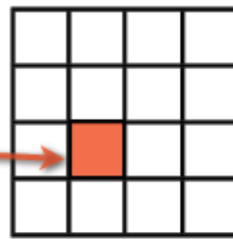
Minification

Texel Shrinking

- One pixel is covered by multiple texels:



Texel Space

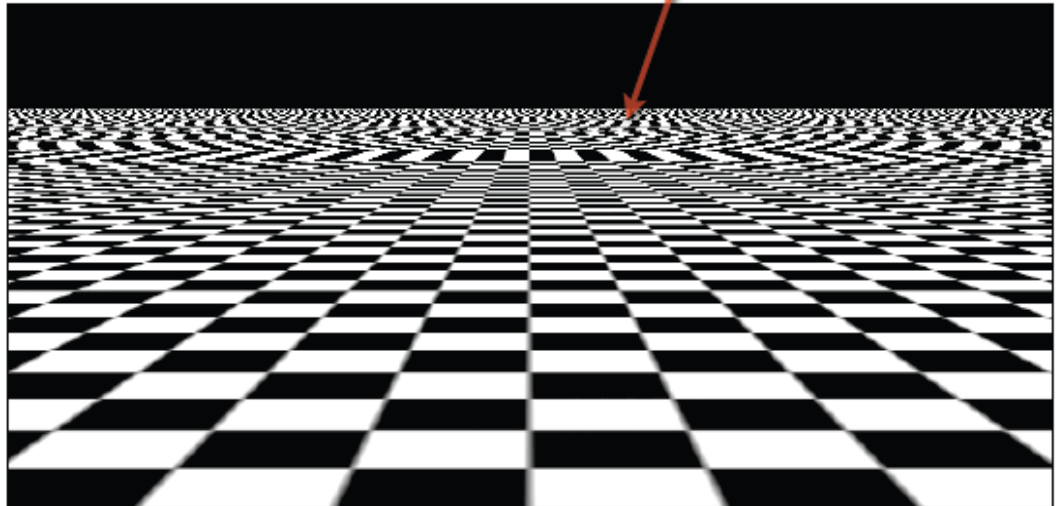


Pixel Space

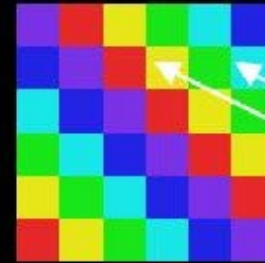
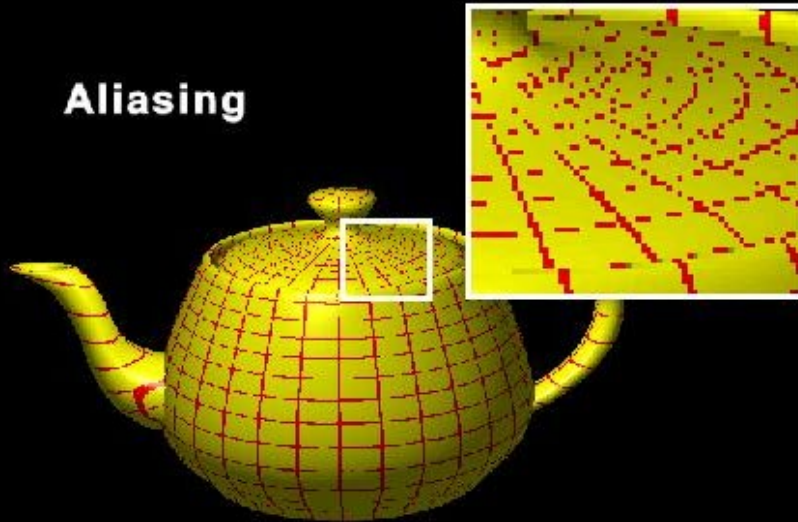
Can cause severe **aliasing**

Filtering Techniques

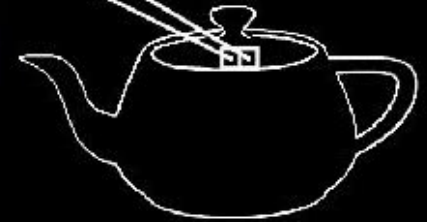
- Nearest neighbor
- Bilinear interpolation
- Mipmapping



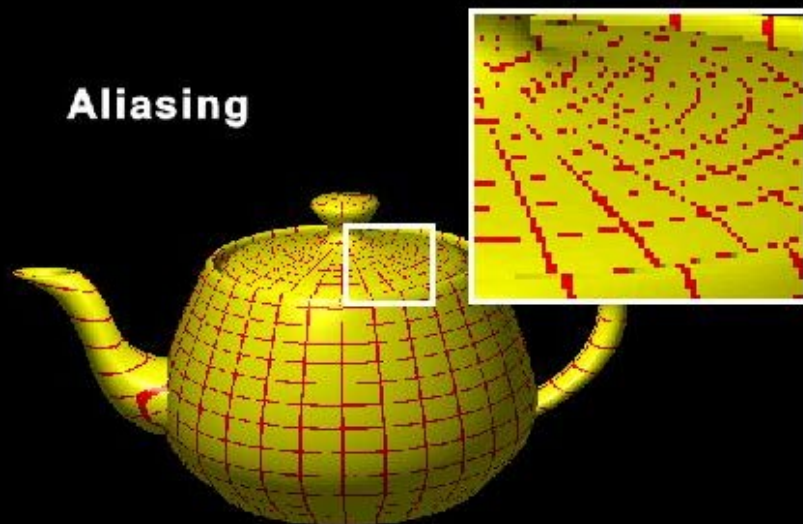
Aliasing



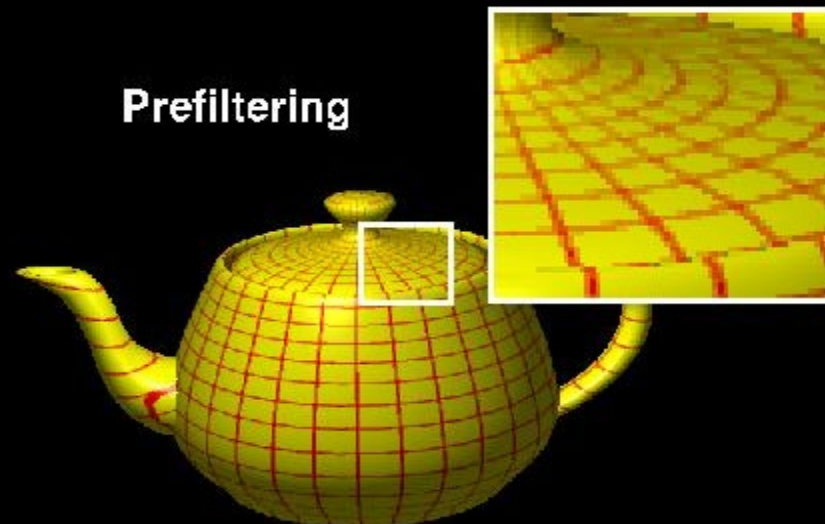
Undersampling

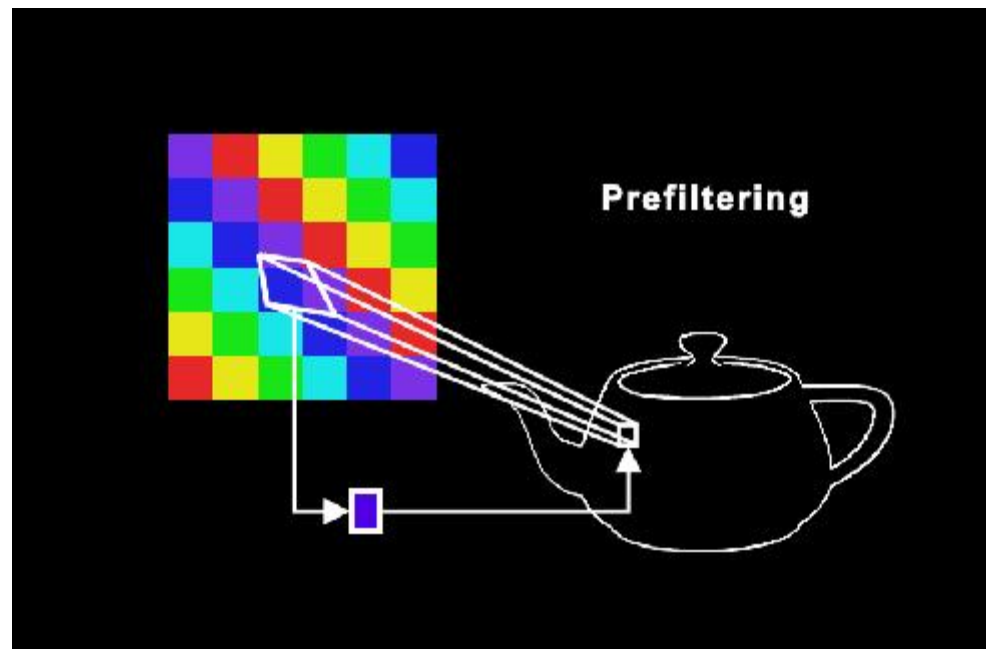


Aliasing



Prefiltering

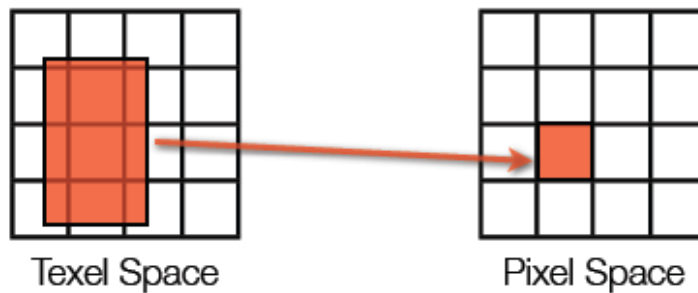




Minification

Texel Shrinking

- One pixel is covered by multiple texels:



Its slow to add up all those values!
How do we make it faster?

Mipmapping

Pyramidal Parametrics

Lance Williams

Computer Graphics Laboratory
 New York Institute of Technology
 Old Westbury, New York

Abstract

The mapping of images onto surfaces may substantially increase the realism and information content of computer-generated imagery. The projection of a flat source

1. Pyramidal Data Structures

Pyramidal data structures may be based on various subdivisions: binary trees, quad trees, oct trees, etc.

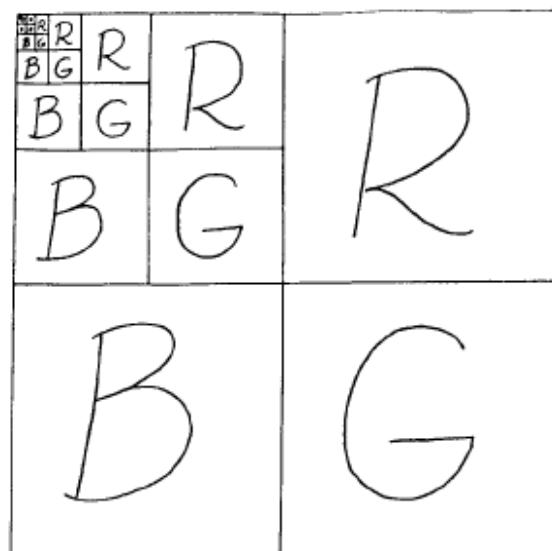
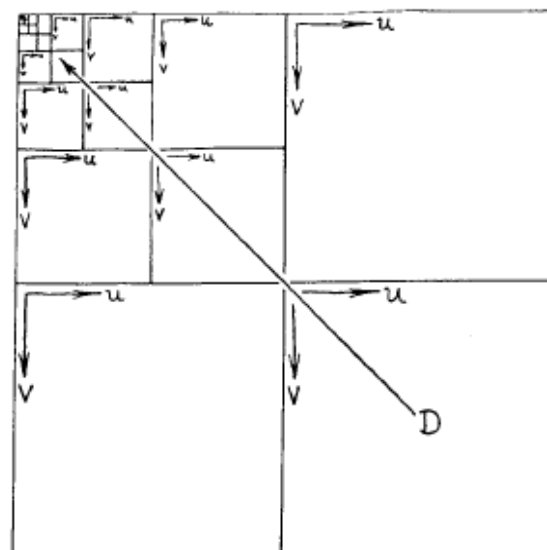


Figure (1)

Structure of a Color Mip Map
 Smaller and smaller images diminish into the upper left corner of the map. Each of the images is averaged down from its larger predecessor.

below:
 Mip maps are indexed by three coordinates: U, V, and D. U and V are spatial coordinates of the map; D is the variable used to index, and interpolate between, the different levels of the pyramid.



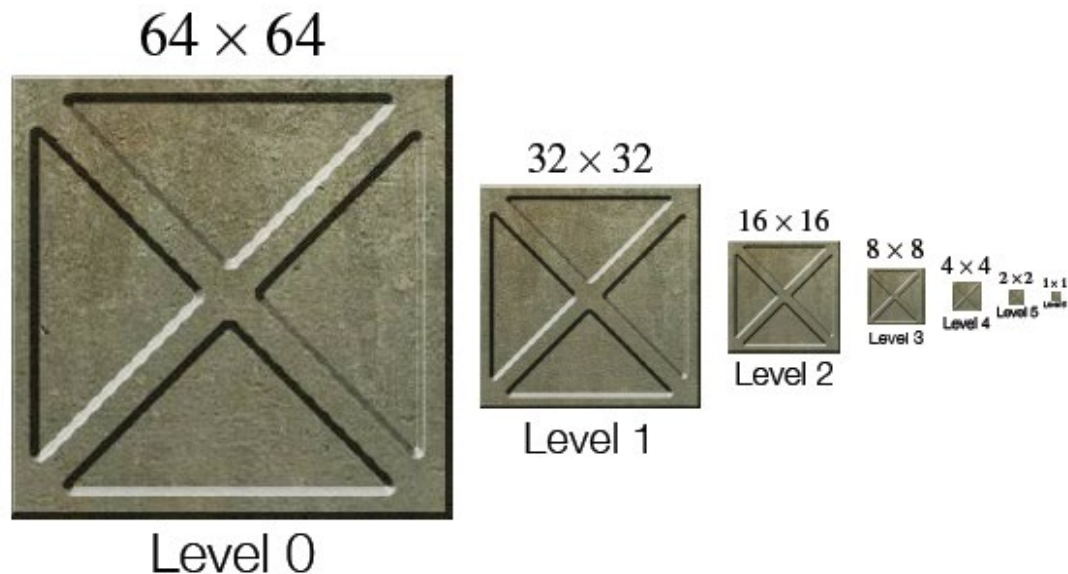
Mipmapping

Mipmaps

- From the Latin *multum in parvo*, many things in a small place
- Create pyramid of successively smaller versions of original texture:

$$Area(k+1) = \left(\frac{1}{4}\right) Area(k)$$

- Quality of pyramid heavily dependent on filter used in downsampling

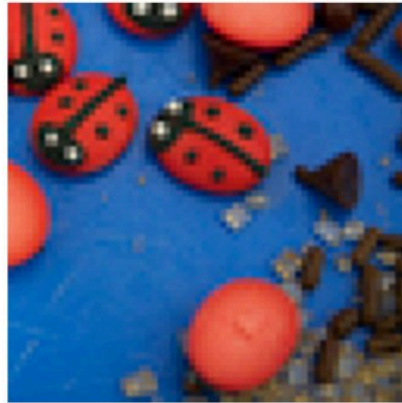


Mipmap (L. Williams 83)

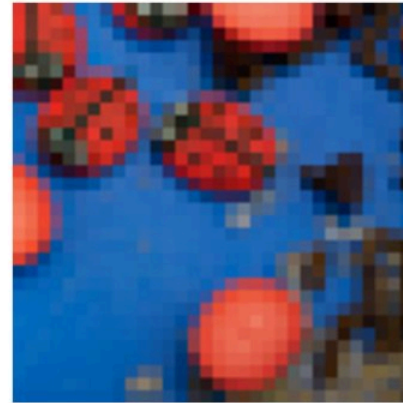
Each mipmap level is downsampled (low-pass filtered) version of the previous



Level 0 = 128x128



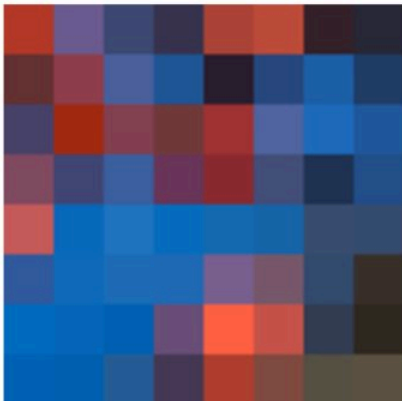
Level 1 = 64x64



Level 2 = 32x32



Level 3 = 16x16



Level 4 = 8x8



Level 5 = 4x4



Level 6 = 2x2



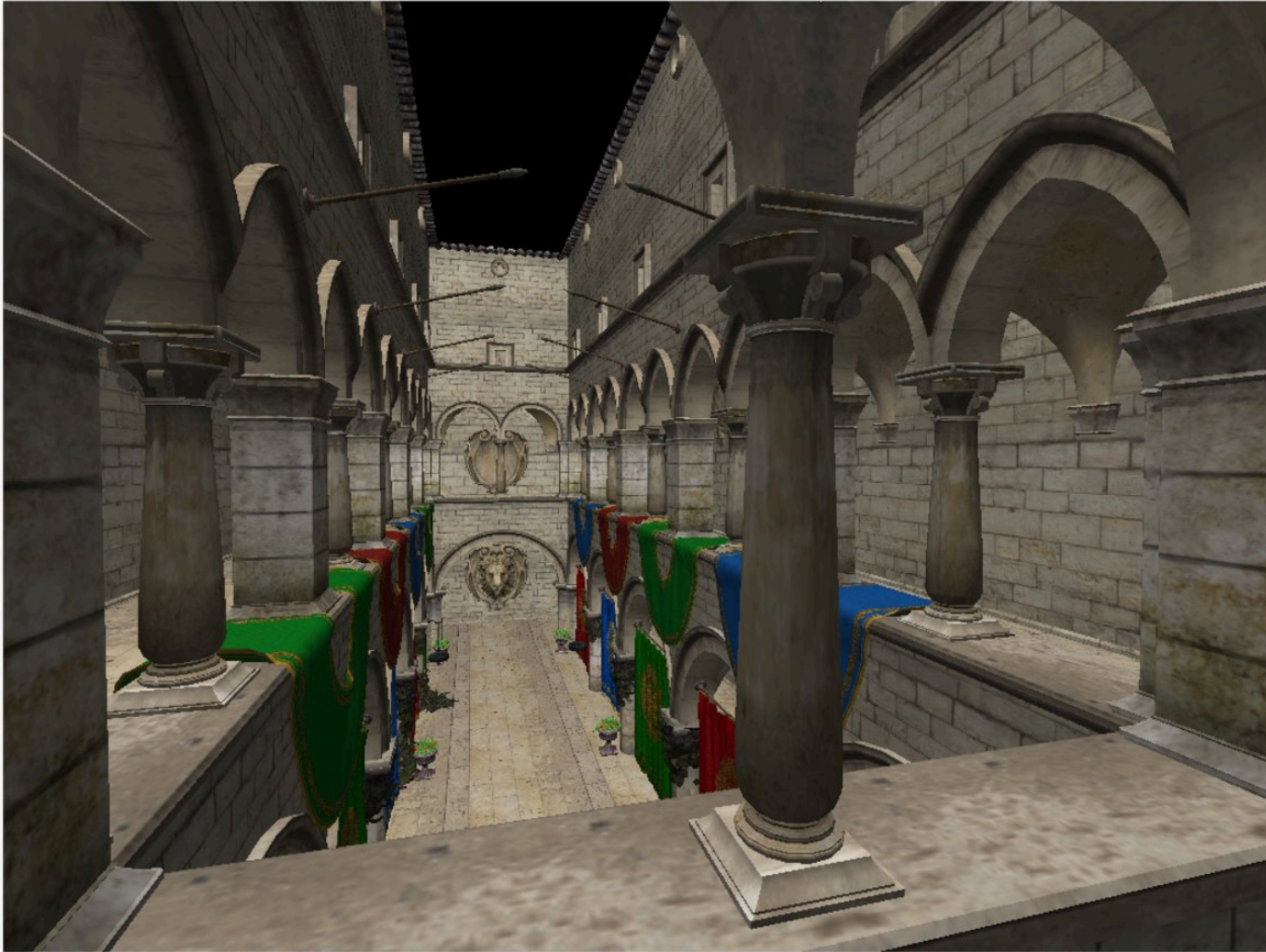
Level 7 = 1x1

“Mip” comes from the Latin “multum in parvo”, meaning a multitude in a small space

Bilinear resampling at level 0



Bilinear resampling at level 2

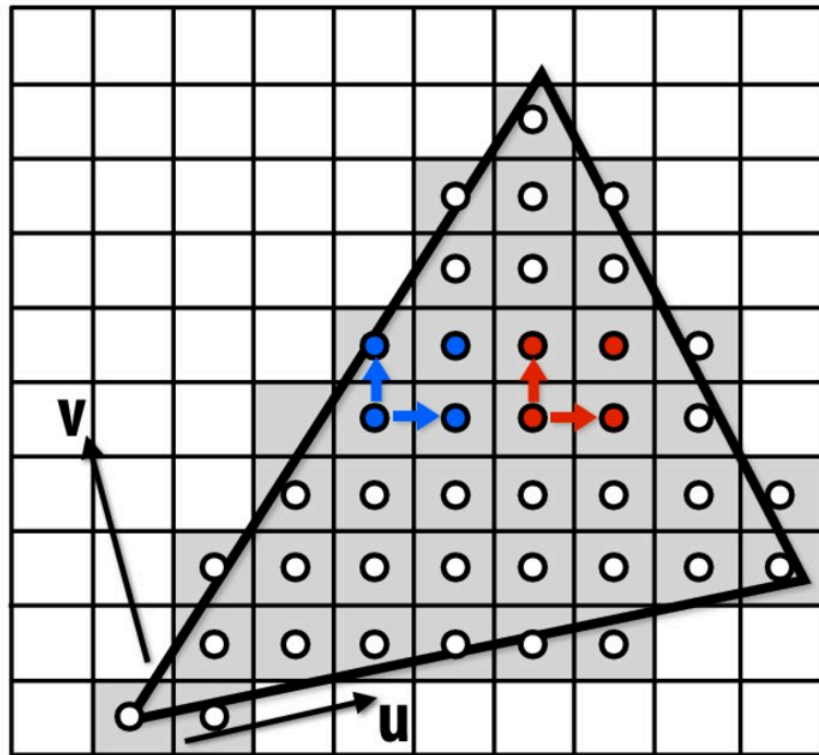


Bilinear resampling at level 4

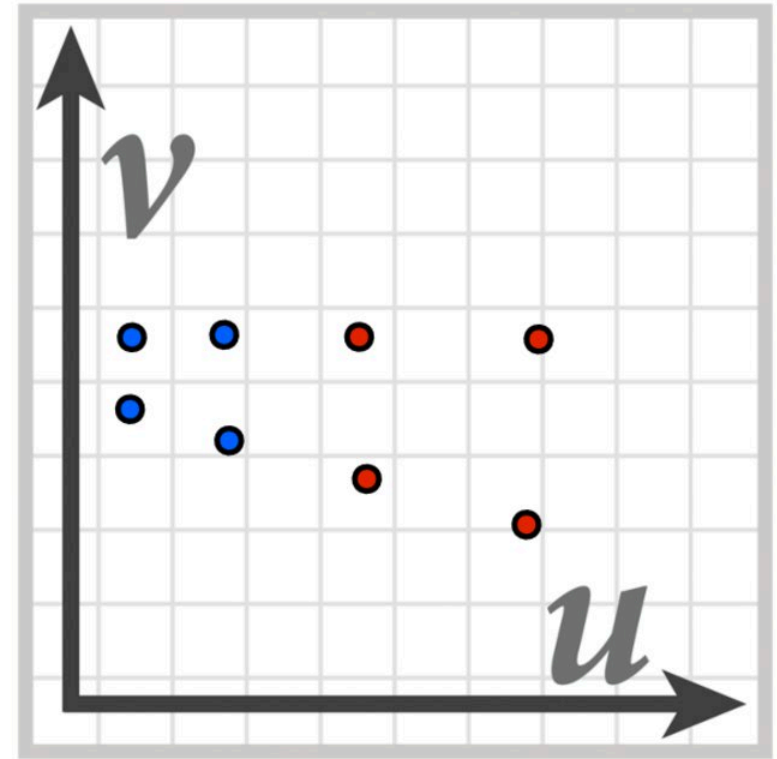


Computing mipmap level

Compute differences between texture coordinate values of neighboring screen samples



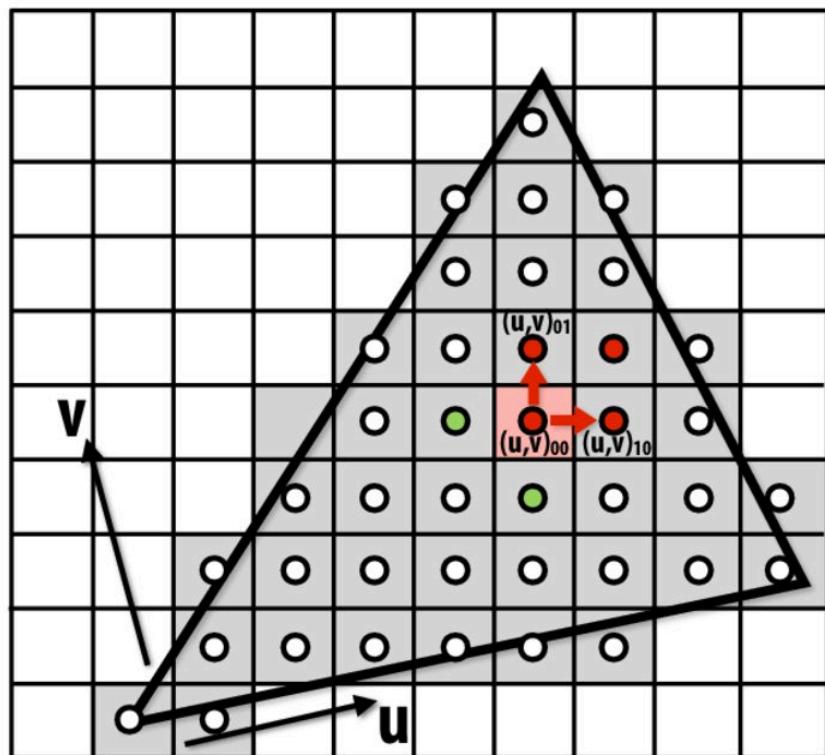
Screen space



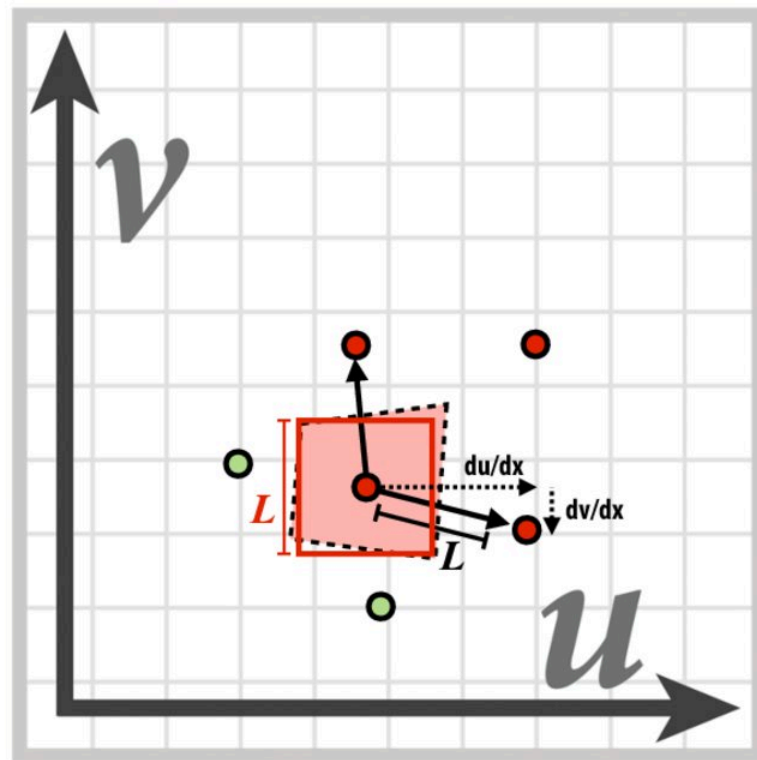
Texture space

Computing mipmap level

Compute differences between texture coordinate values of neighboring screen samples



$$\begin{aligned} du/dx &= u_{10} - u_{00} & dv/dx &= v_{10} - v_{00} \\ du/dy &= u_{01} - u_{00} & dv/dy &= v_{01} - v_{00} \end{aligned}$$

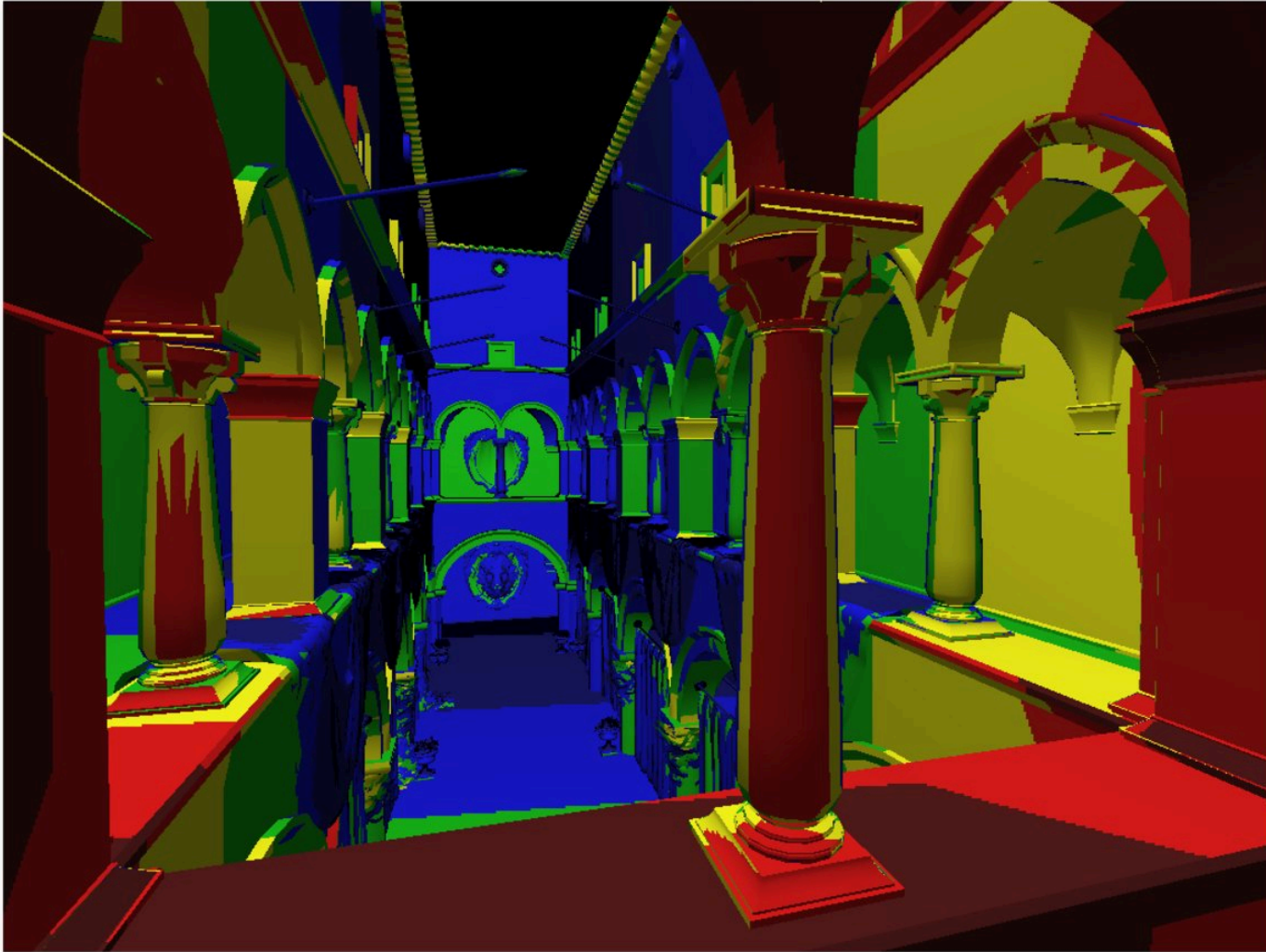


$$L = \max \left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2} \right)$$

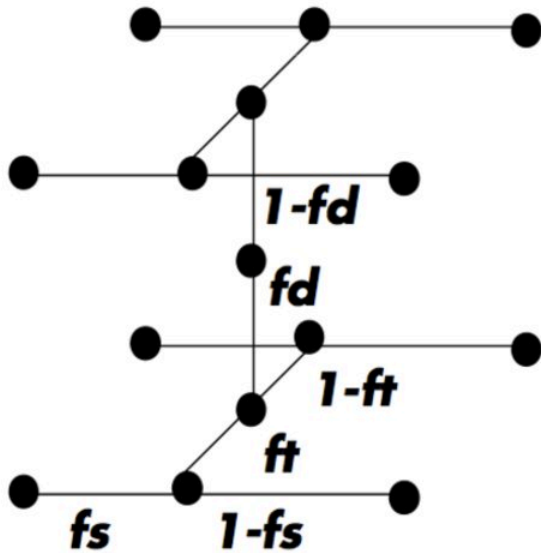
mip-map $d = \log_2 L$

Visualization of mipmap level

(bilinear filtering only: d clamped to nearest level)



“Tri-linear” filtering



$$\text{lerp}(t, v_1, v_2) = v_1 + t(v_2 - v_1)$$

Bilinear resampling:

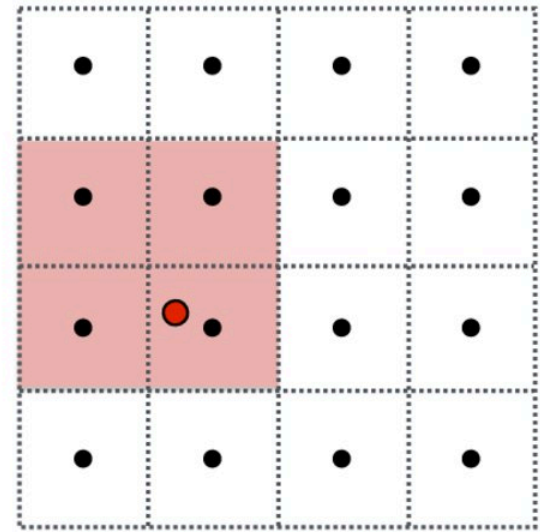
four texel reads

3 lerps (3 mul + 6 add)

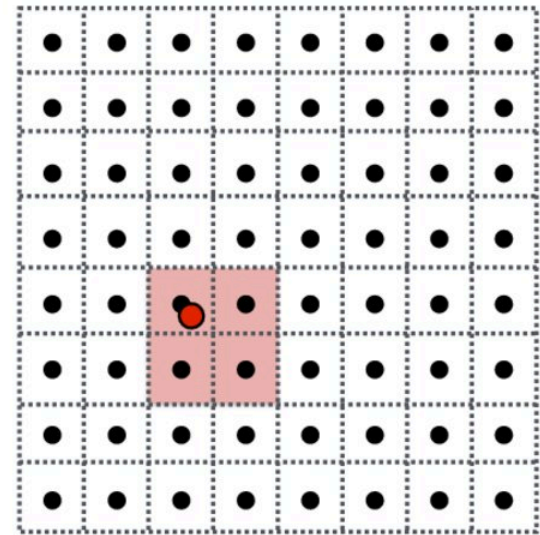
Trilinear resampling:

eight texel reads

7 lerps (7 mul + 14 add)



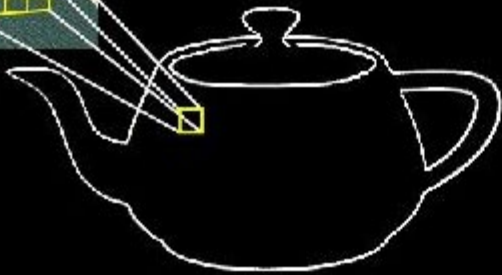
mip-map texels: level $d+1$



mip-map texels: level d



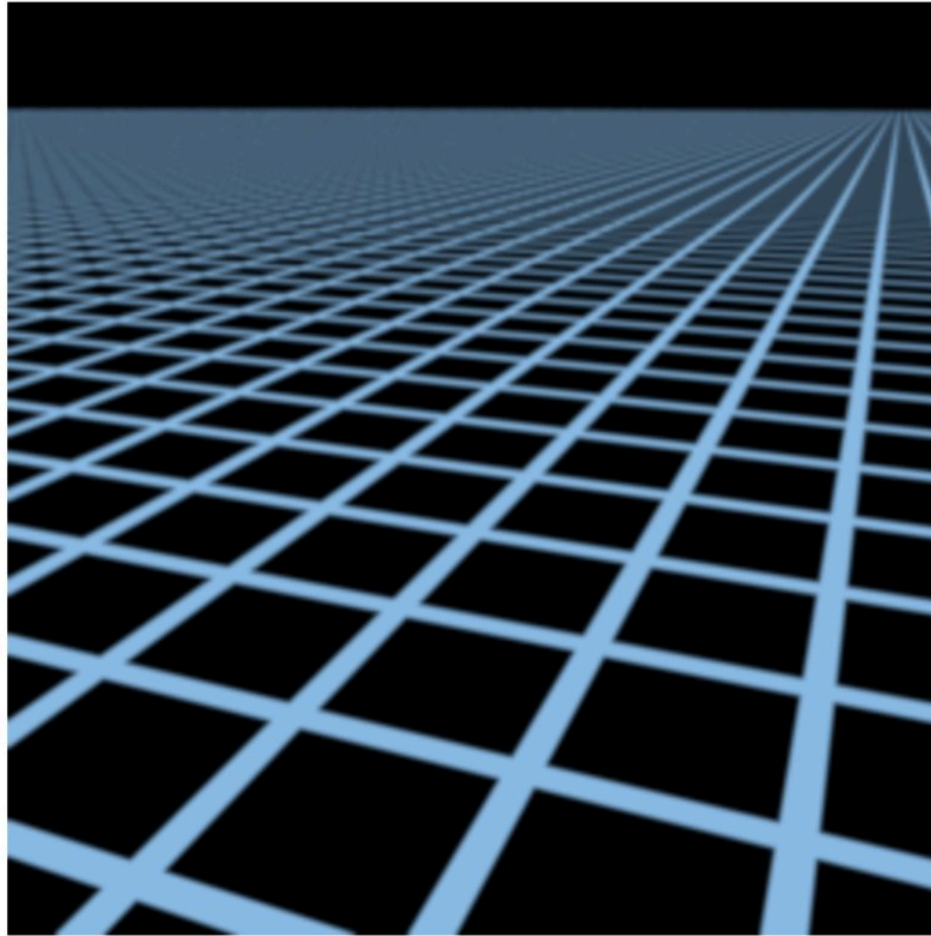
9:1



1:1



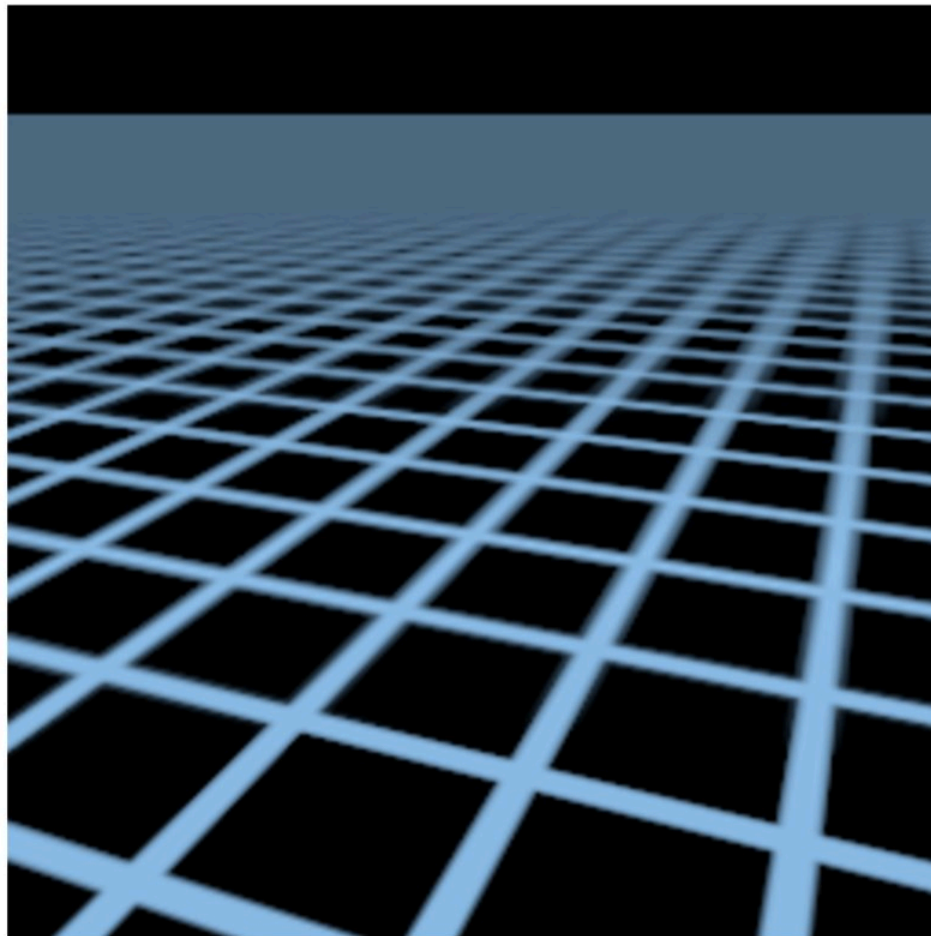
Example: mipmap limitations



**Supersampling 512x
(desired answer)**

Example: mipmap limitations

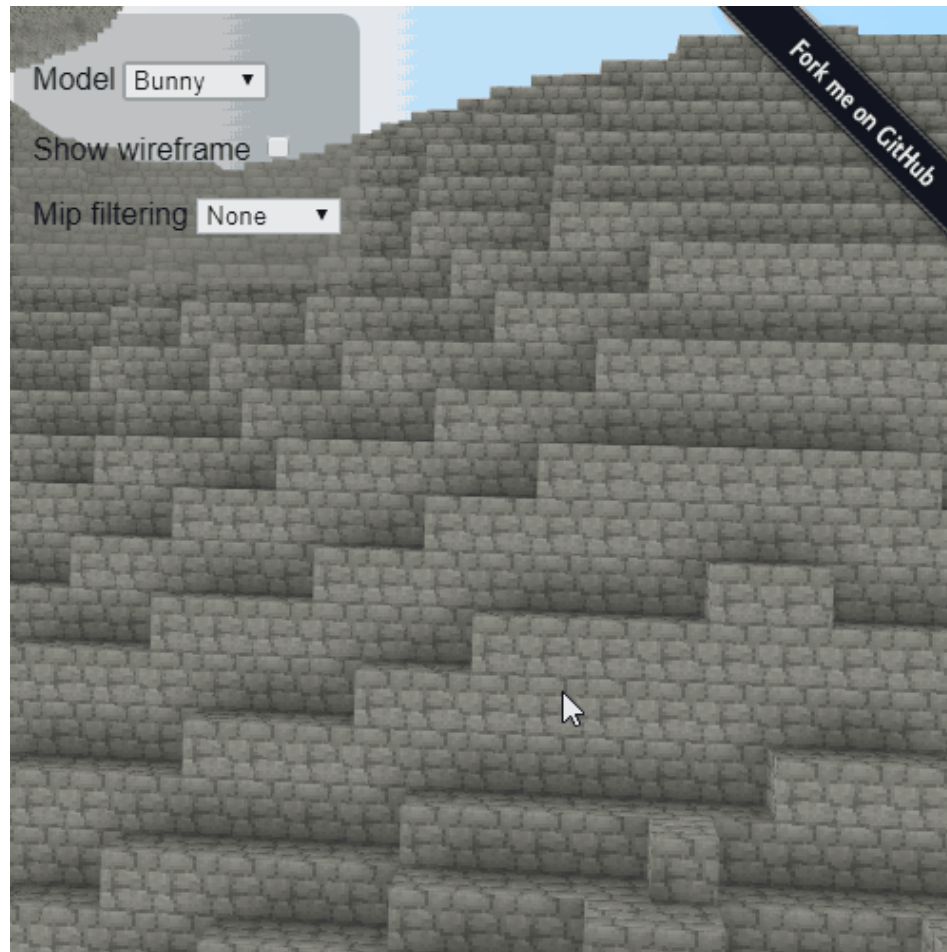
Overblur
Why?



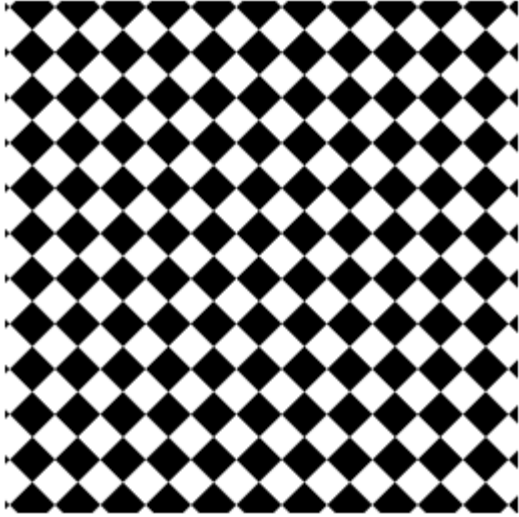
Mipmap trilinear sampling

Video example: mipmapping





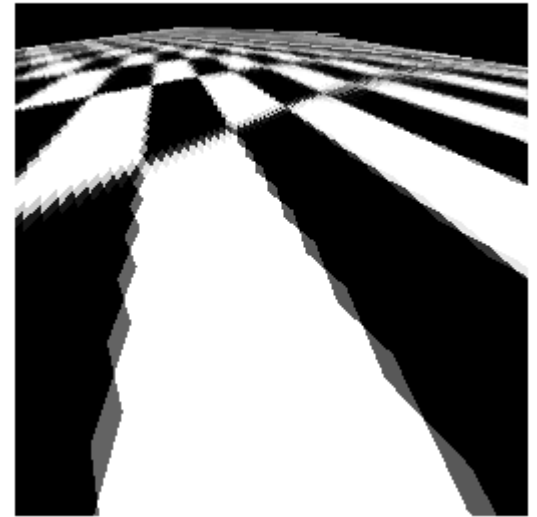
Initial Image



Bilinear interpolation AND mipmap filtering



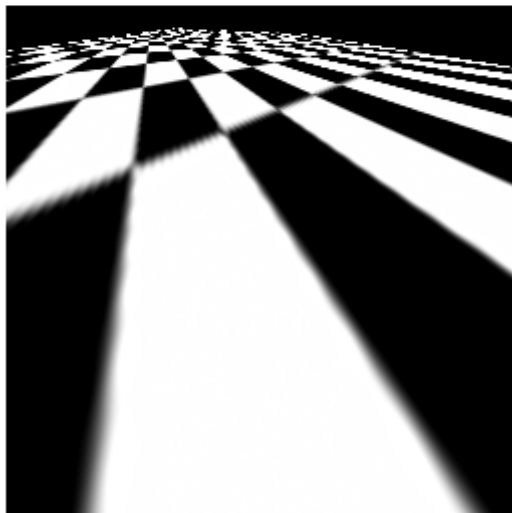
NOT Bilinear interpolation
but yes mipmap filtering



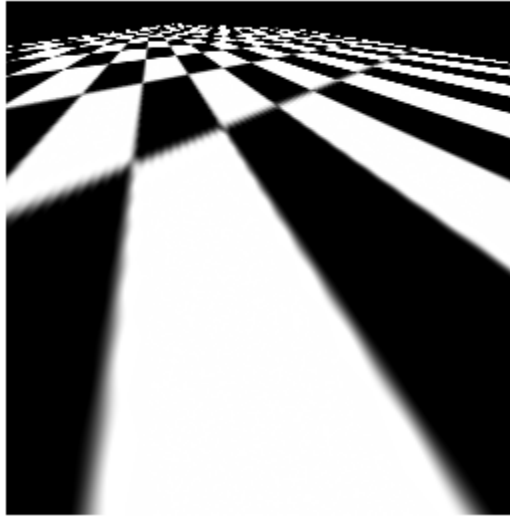
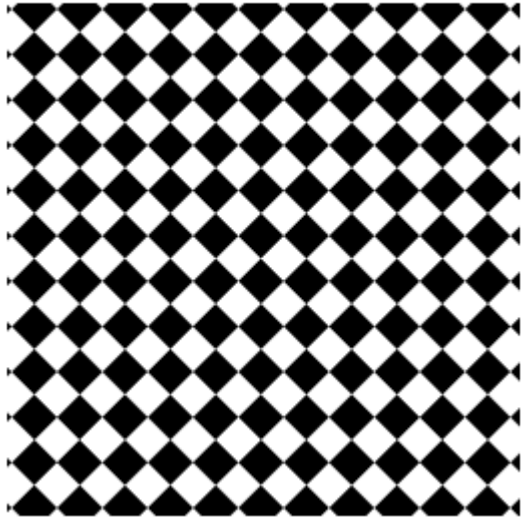
Point sampled



Bilinear interpolation
but not mipmap filtering



Initial Image



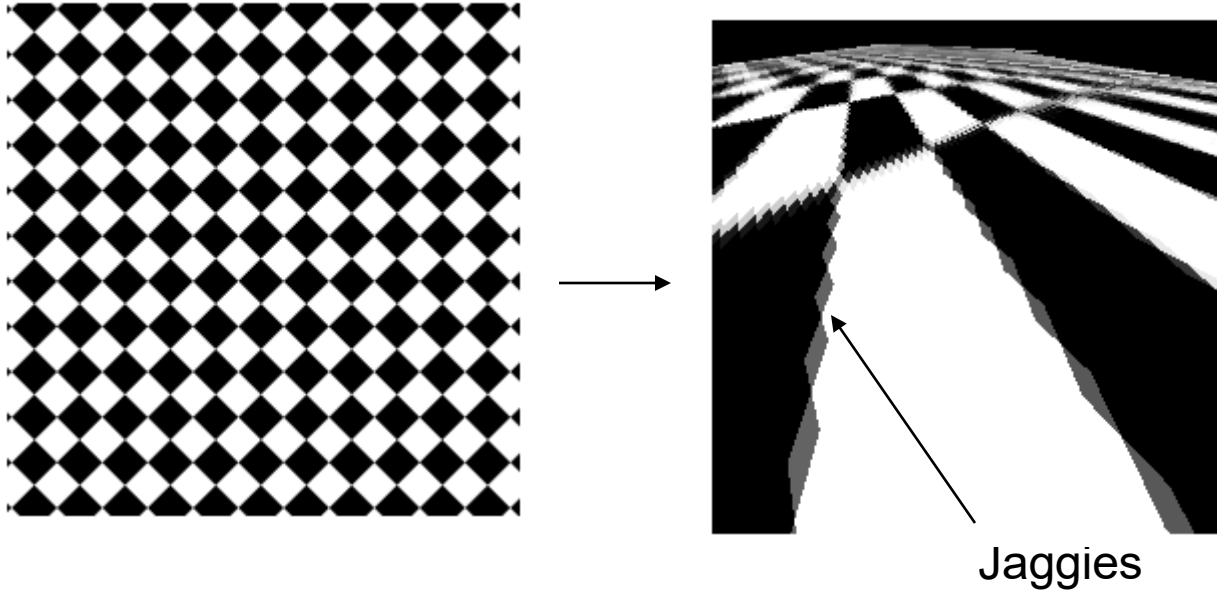
Aliasing



How do we fix the aliasing in the back of this rendered image?

- A - Need to turn on bi-linear interpolation
- B - Need to turn on mipmapping
- C - Need to turn on both to fix this
- D - This can't be fixed, both are already on
- E - Don't know

Initial Image

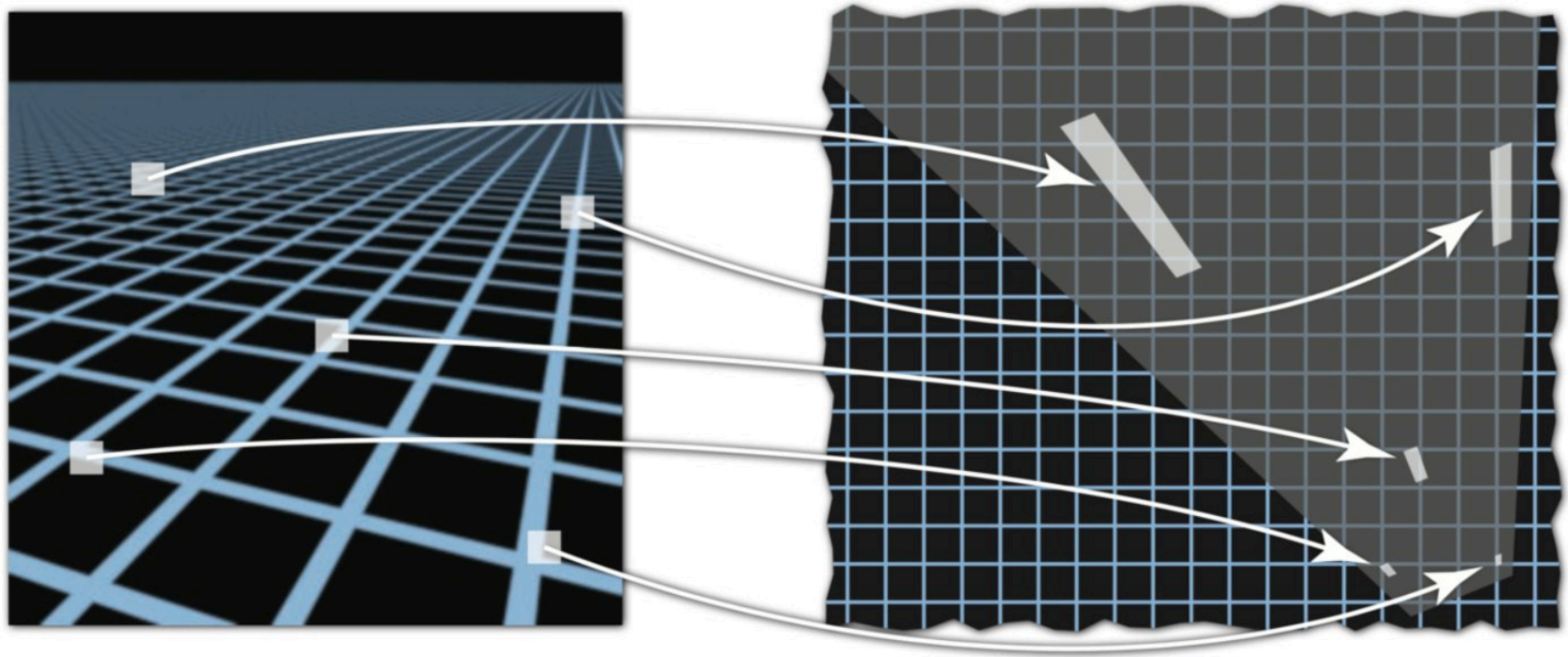


How do we fix the aliasing in the front of this rendered image?

- A - Need to turn on bi-linear interpolation
- B - Need to turn on mipmapping
- C - Need to turn on both to fix this
- D - This can't be fixed, both are already on
- E - Don't know

Anisotropic texturing

Screen pixel footprint in texture space



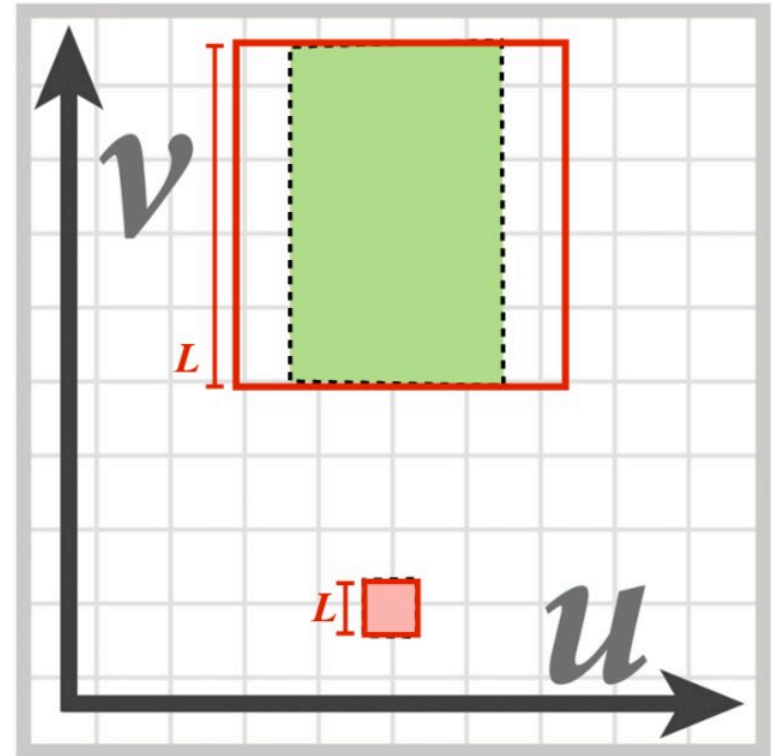
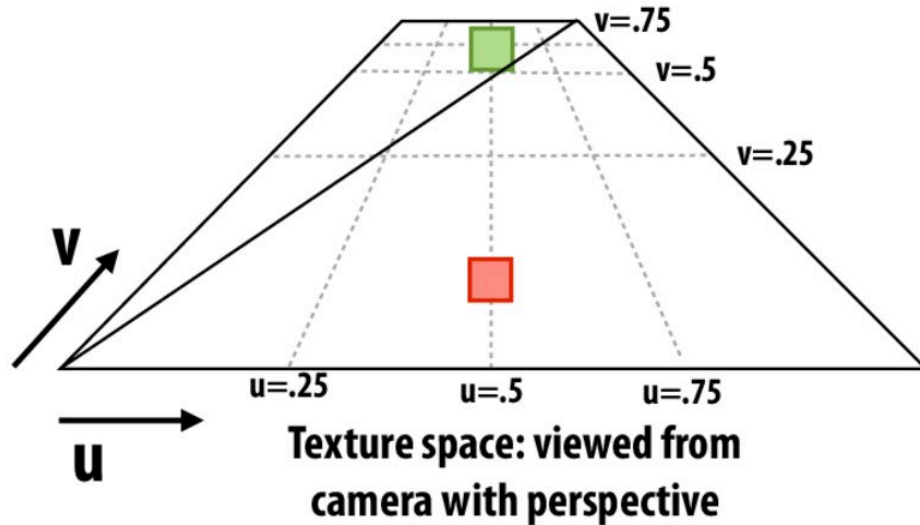
Screen space

Texture space

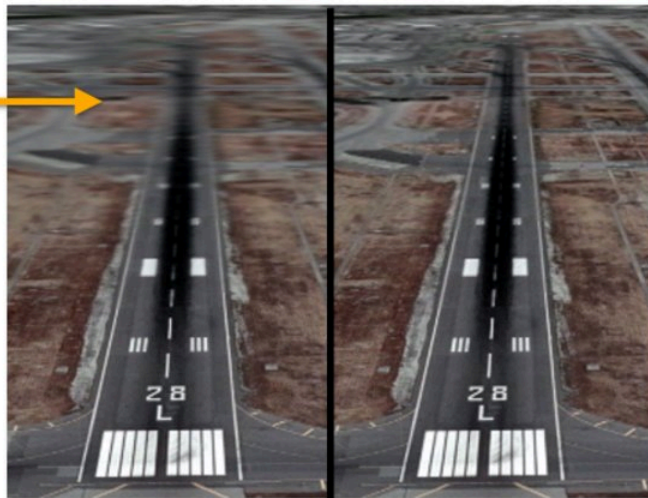
Texture sampling pattern not rectilinear or isotropic

Pixel area may not map to isotropic region in texture space

Proper filtering requires anisotropic filter footprint



Overblurring in u direction



Trilinear (Isotropic) Filtering

Anisotropic Filtering

(Modern anisotropic texture filtering solutions combine multiple mip map samples)

Summed-Area Tables for Texture Mapping

Franklin C. Crow
Computer Sciences Laboratory
Xerox Palo Alto Research Center

Abstract

Texture-map computations can be made tractable through use of precalculated tables which allow computational costs independent of the texture density. The first example of this technique, the "mip" map, uses a set of tables containing successively lower-resolution representations filtered down from the discrete texture function. An alternative method using a single table of values representing the integral over the texture function rather than the function itself may yield superior results at similar cost. The necessary algorithms to support the new technique are explained. Finally, the cost and performance of the new technique is compared to previous techniques.

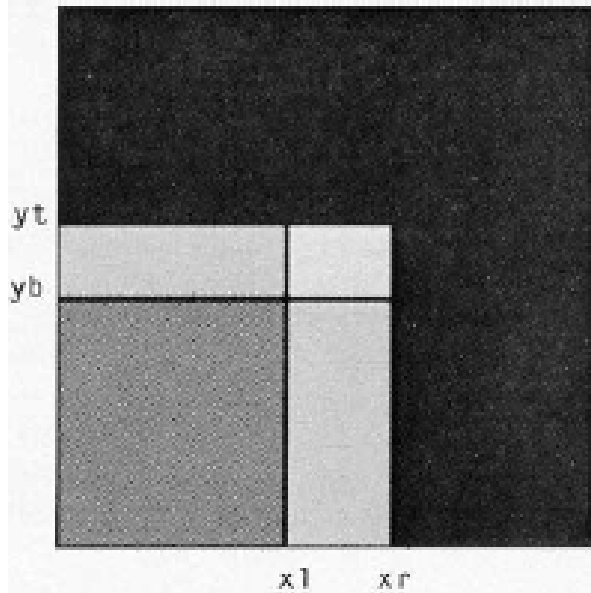


Figure 2: Calculation of summed area from table.

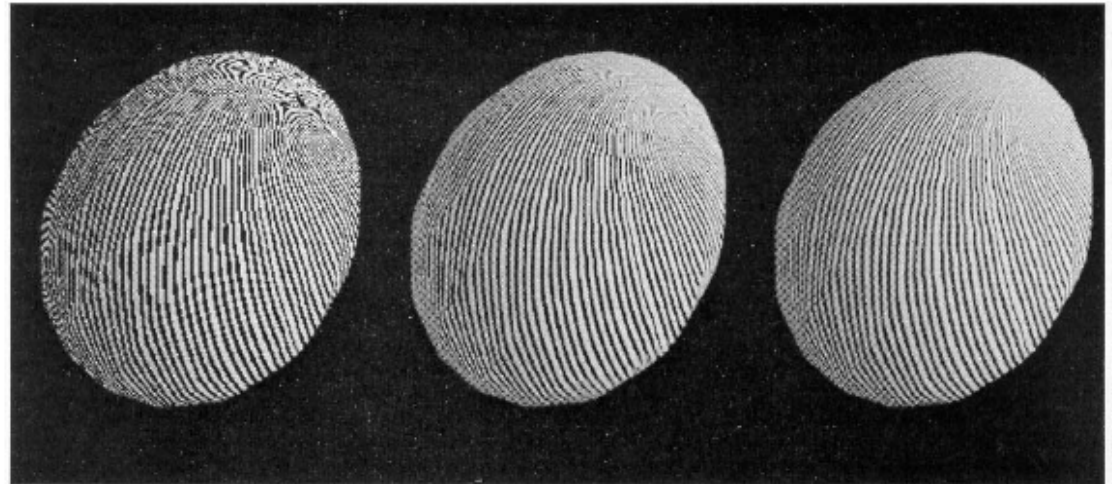
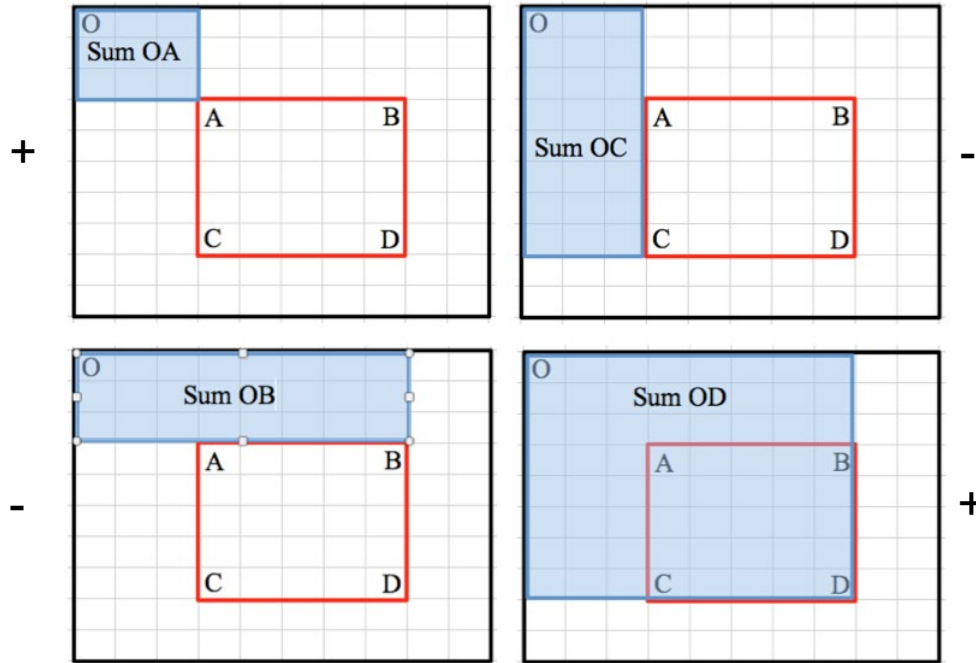


Figure 3: Left: nearest pixel (1 min. CPU time), middle: multiple table (2 1/4 min.), right: summed table (2 min.).

Summed Area Table

Stores the sum in each pixel



1.

31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

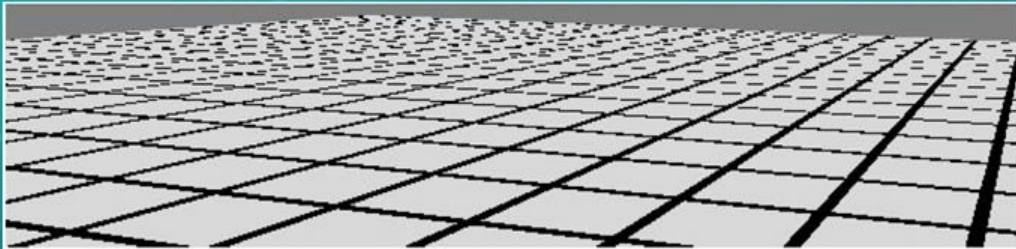
2.

31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

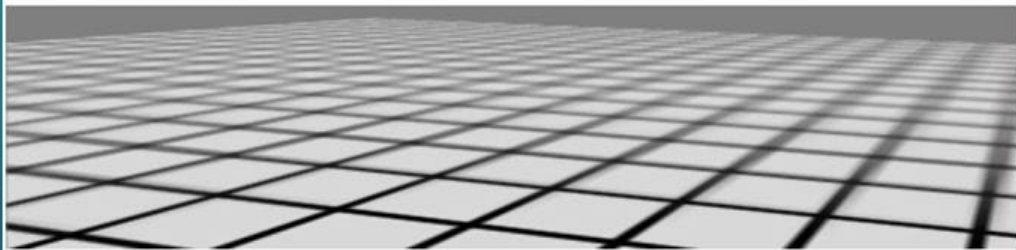
$$15 + 16 + 14 + 28 + 27 + 11 = 111$$

$$101 + 450 - 254 - 186 = 111$$

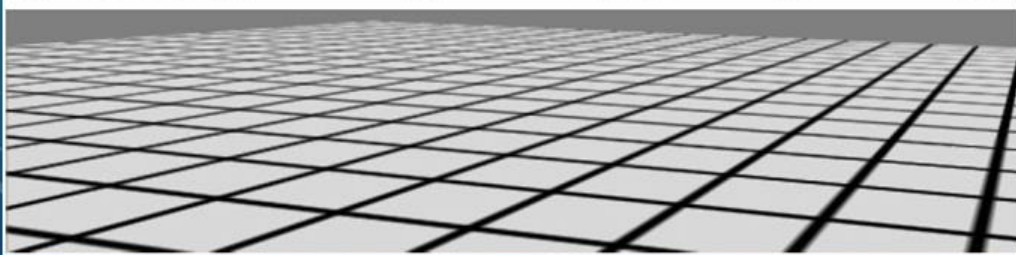
MINIFICATION: COMPARISON OF APPROACHES SO FAR



Nearest neighbor



Mipmapping



Summed area tables

Shape of the region

mipmap

Summed-area

EWA

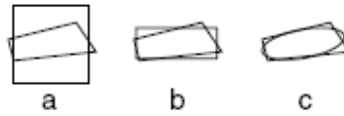
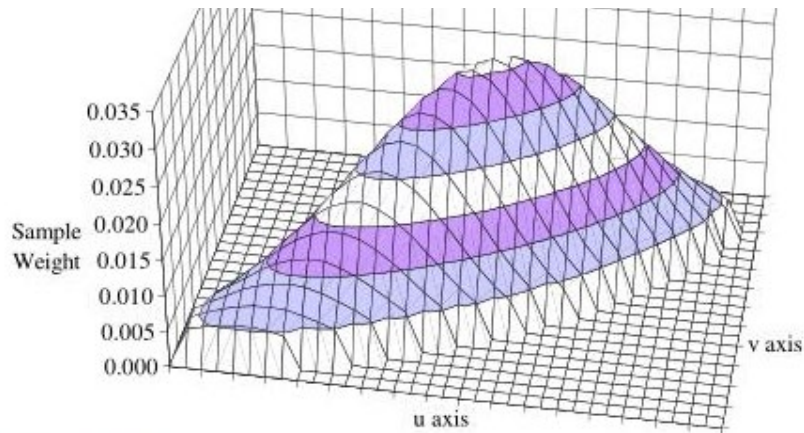


Figure 4: Approximating a quadrilateral texture area with (a) a square, (b) a rectangle, and (c) an ellipse. Too small an area causes aliasing; too large an area causes blurring.



The weighting when we average pixels together

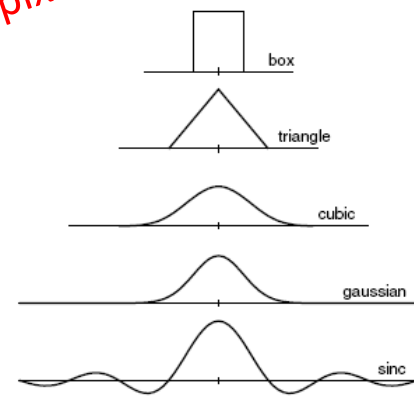
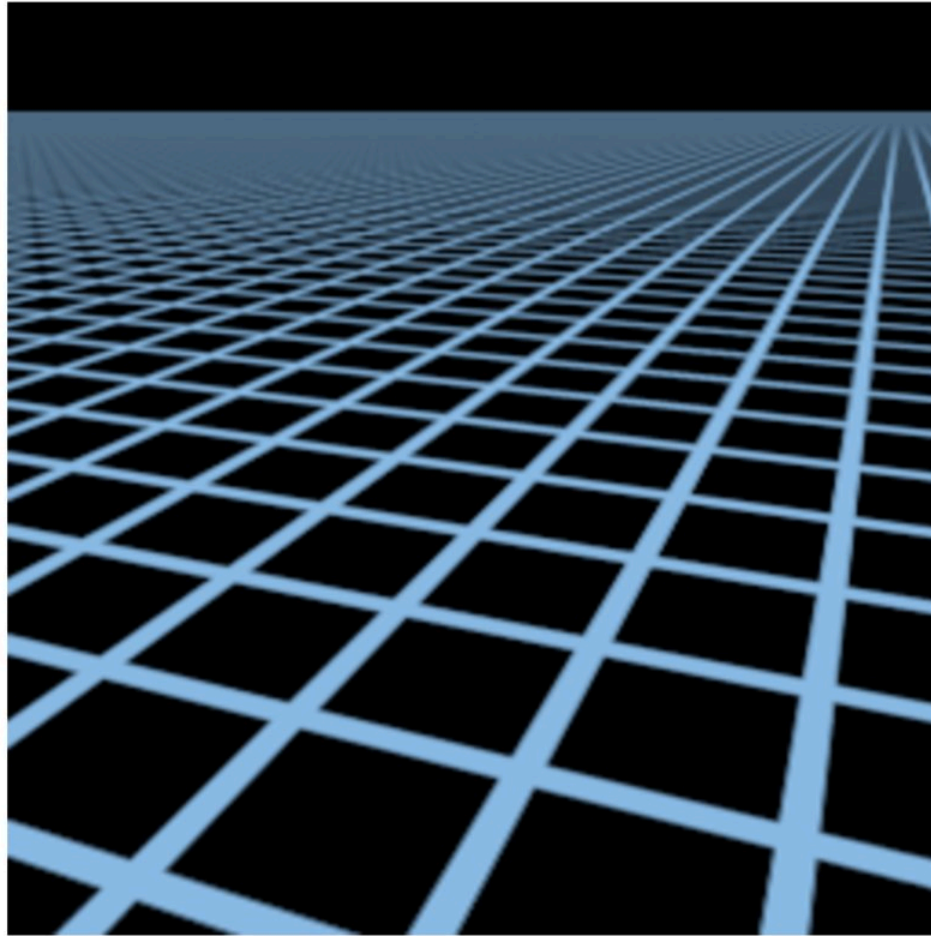
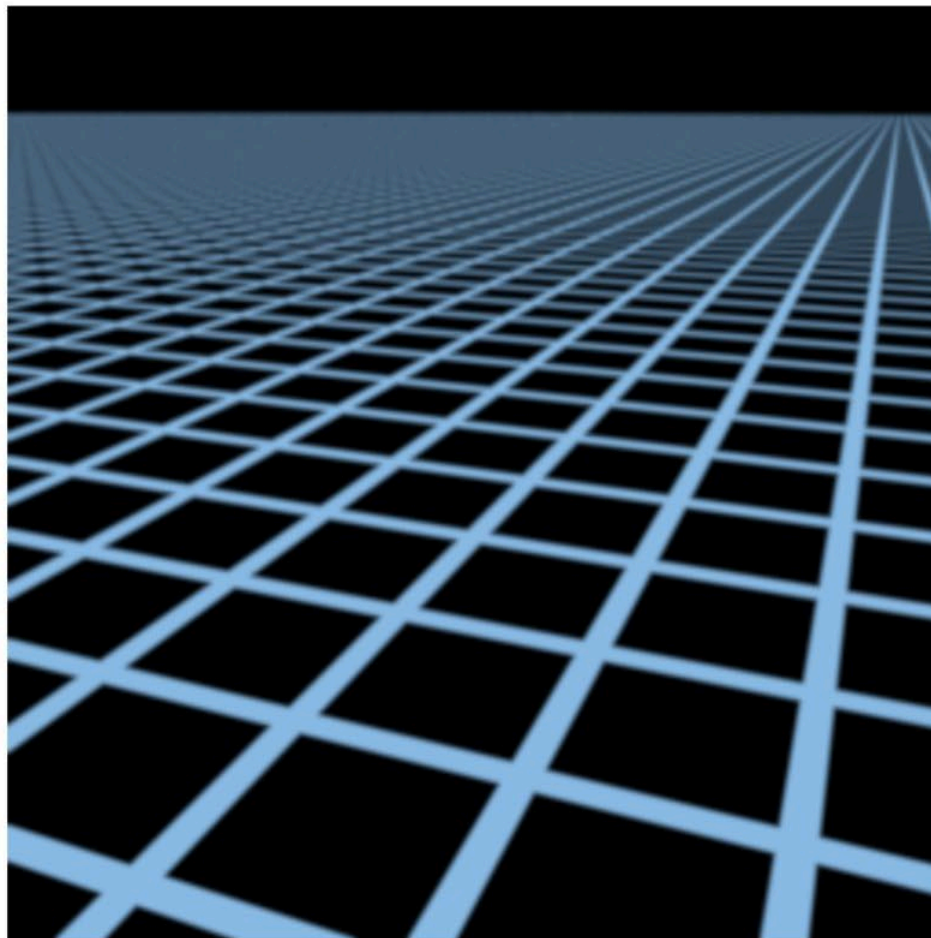


Figure 3: Cross sections of some common texture filters, ordered by quality. The top three are finite impulse response filters.

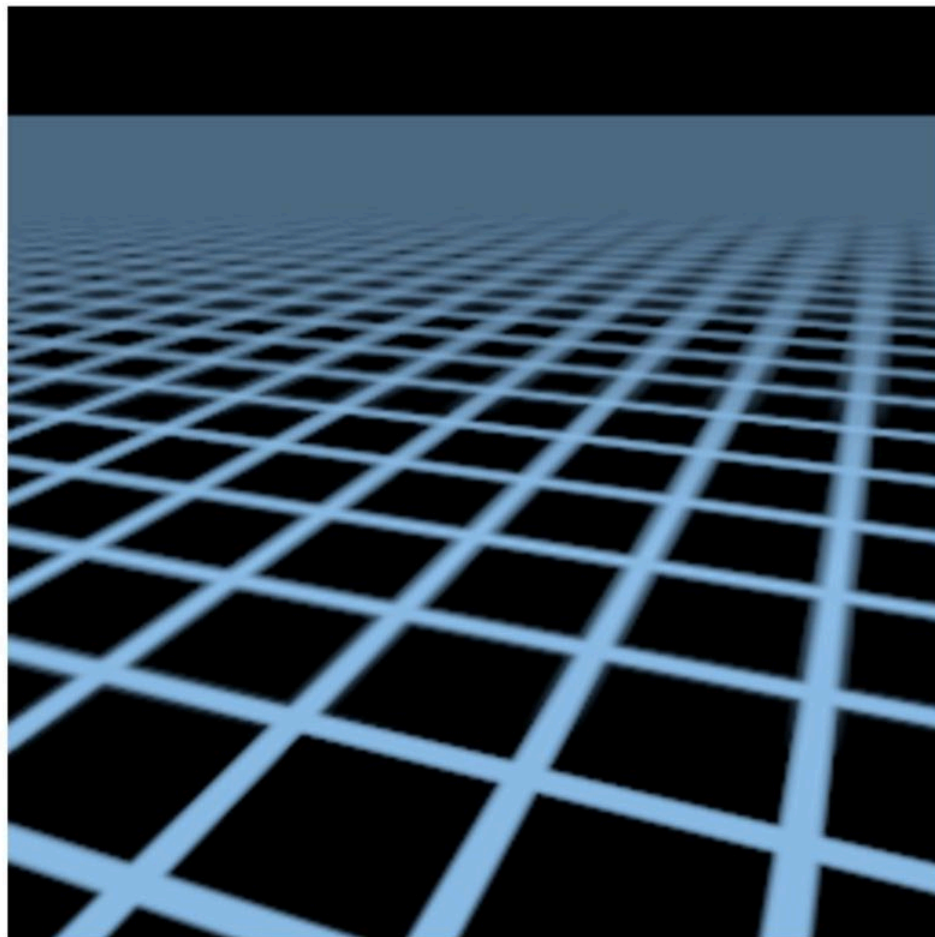
Anisotropic filtering



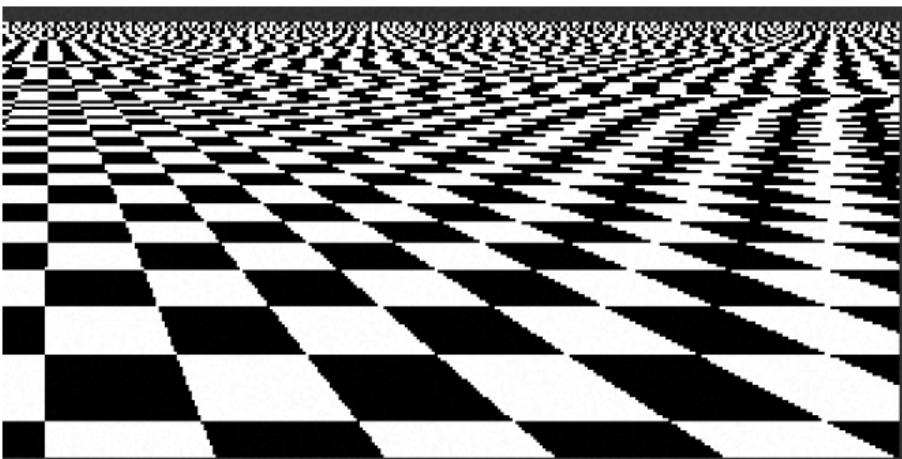
**Elliptical weighted average (EWA) filtering
(uses multiple lookups into mip-map to approximate filter region)**



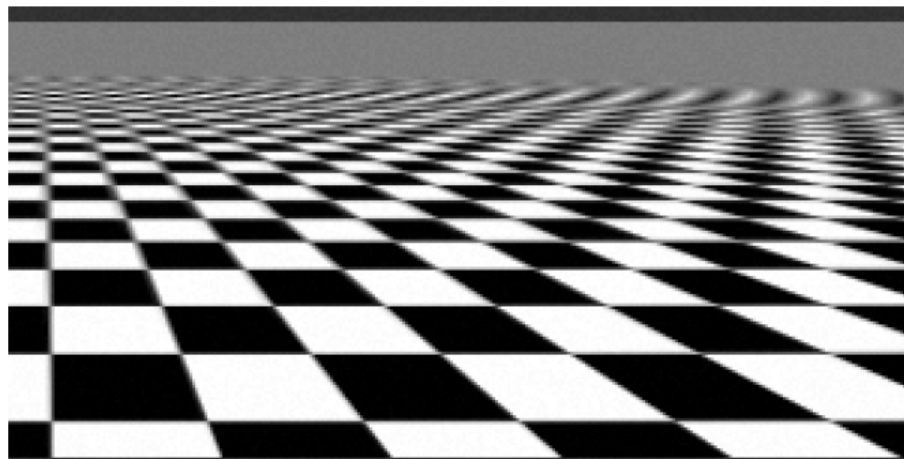
**Supersampling 512x
(desired answer)**



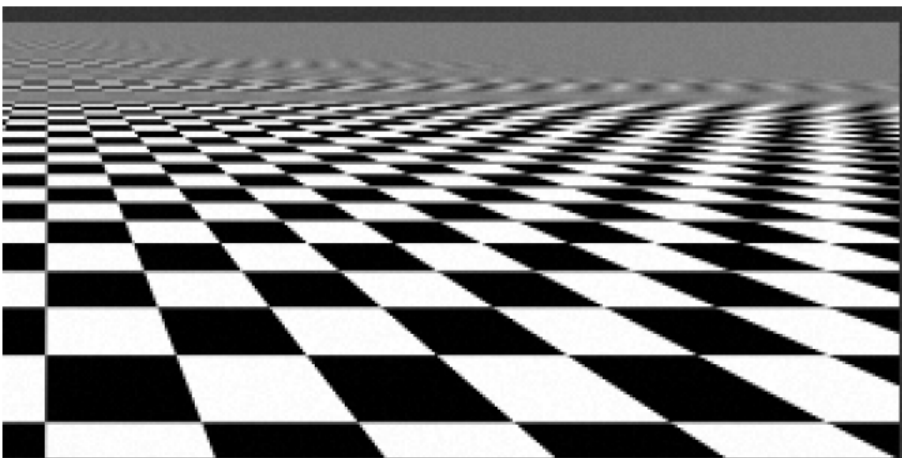
Mipmap trilinear sampling



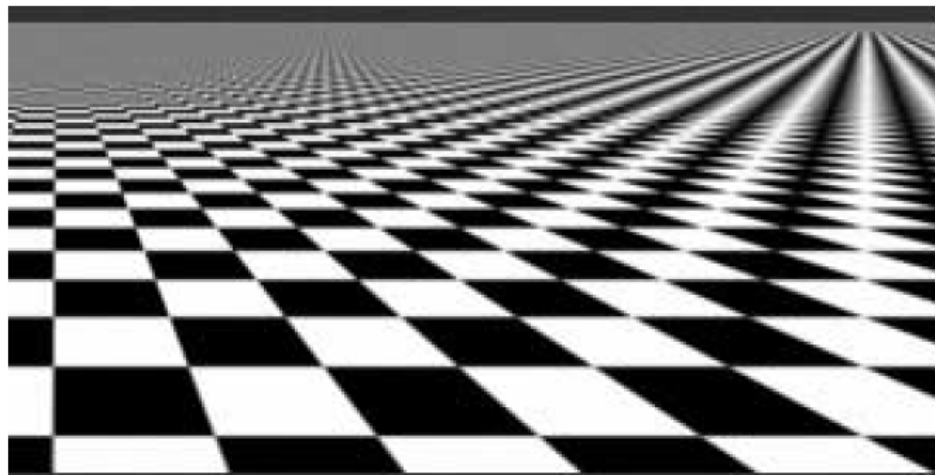
(a) Point sampling.



(b) Trilinear interpolation on a pyramid.



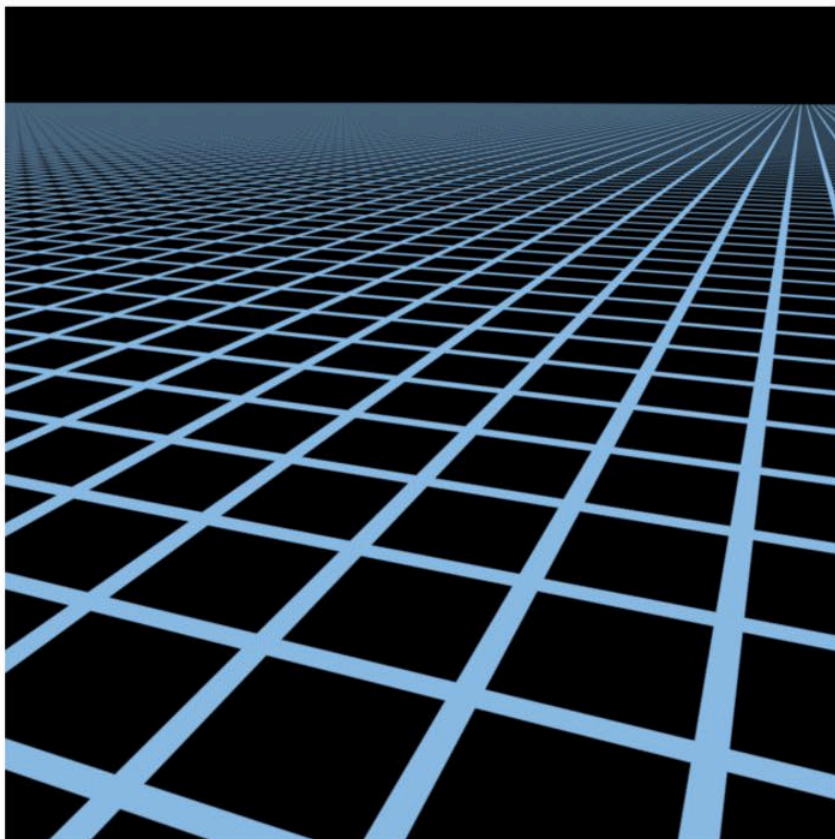
(c) First-order repeated integration (summed area table).



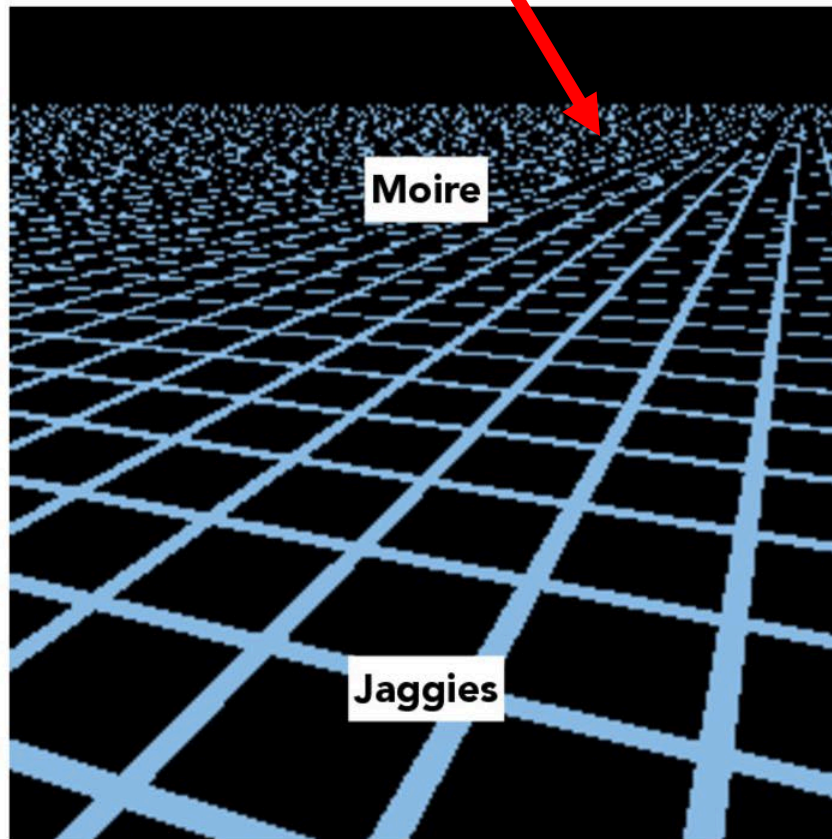
(e) EWA filter with Gaussian cross section on a pyramid.

Antialiasing

Aliasing

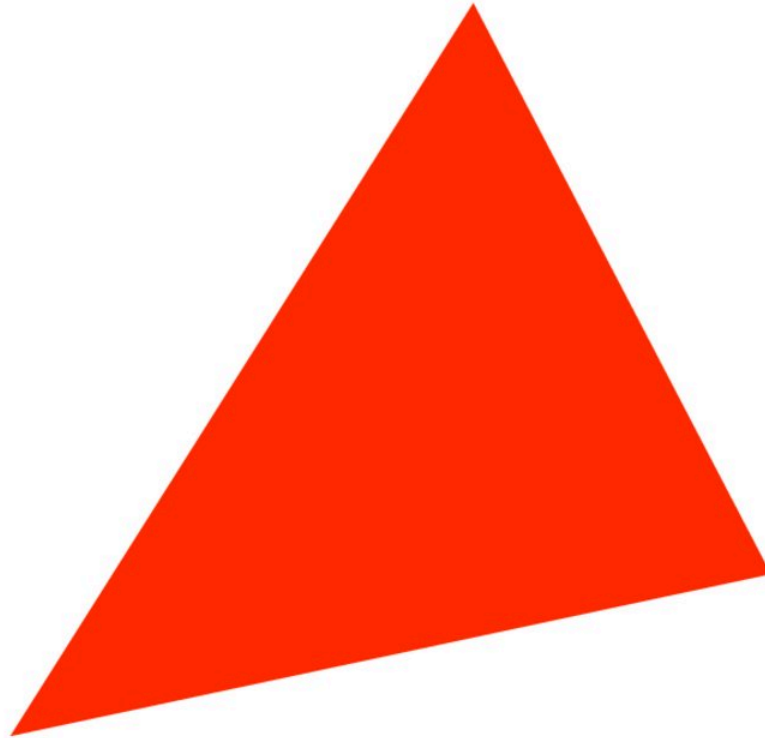


Source image: 1280x1280 pixels

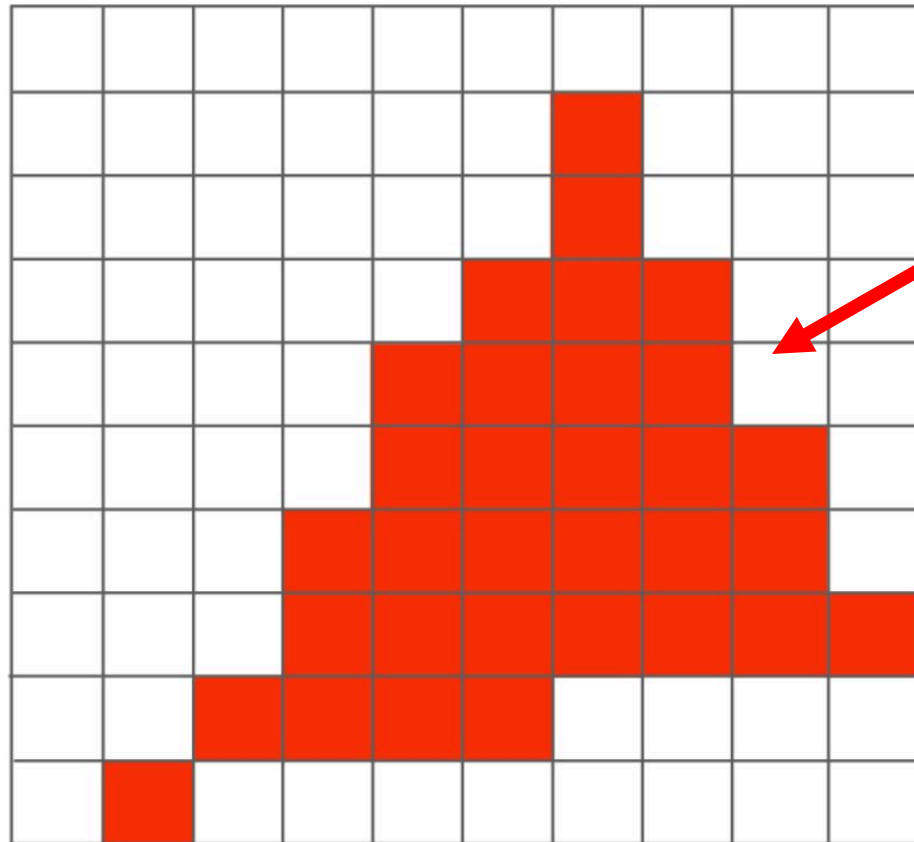


Rendered image: 256x256 pixels

Compare: the continuous triangle function



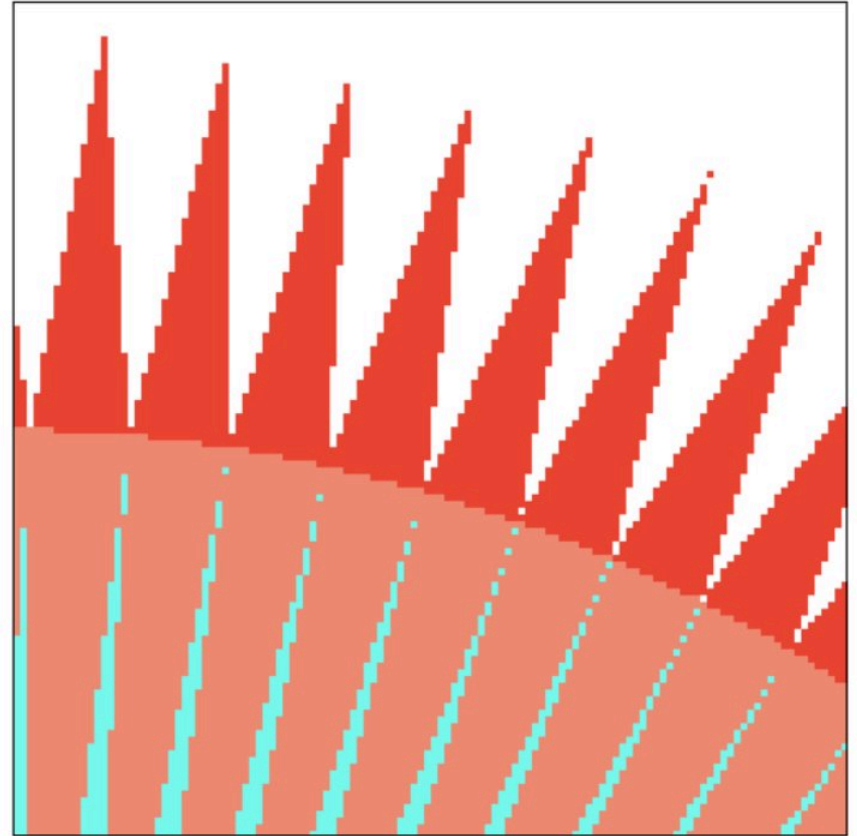
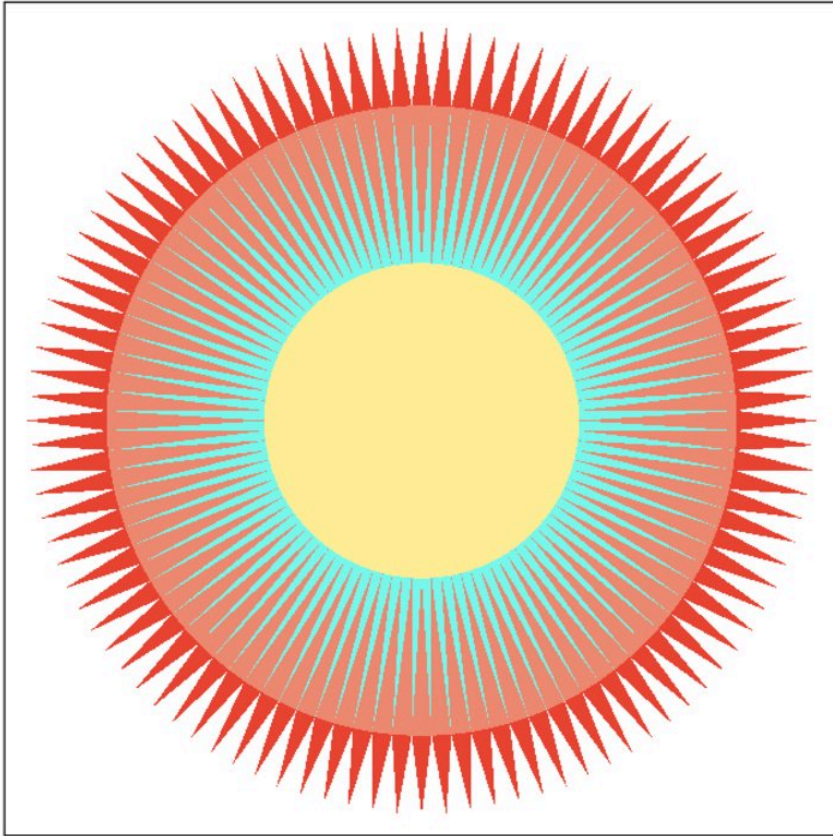
What's wrong with this picture?



Aliasing

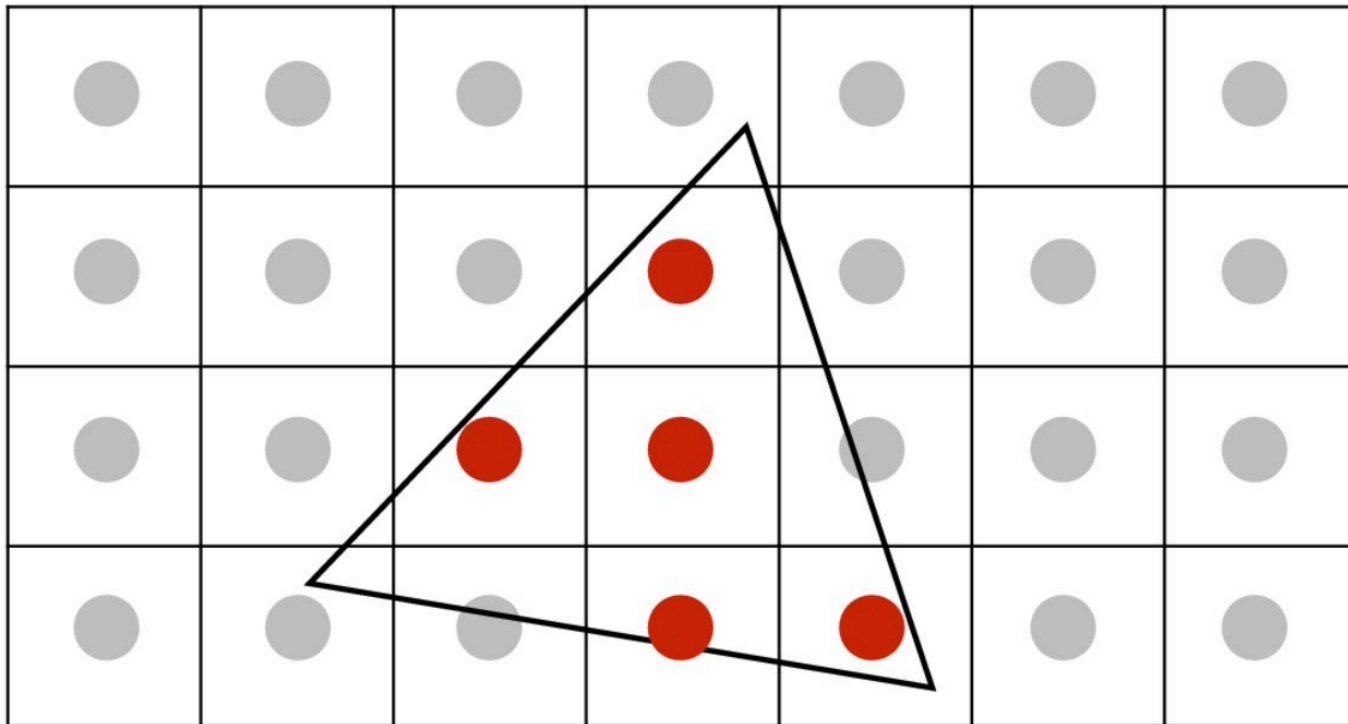
Jaggies!

Jaggies (staircase pattern)



Is this the best we can do?

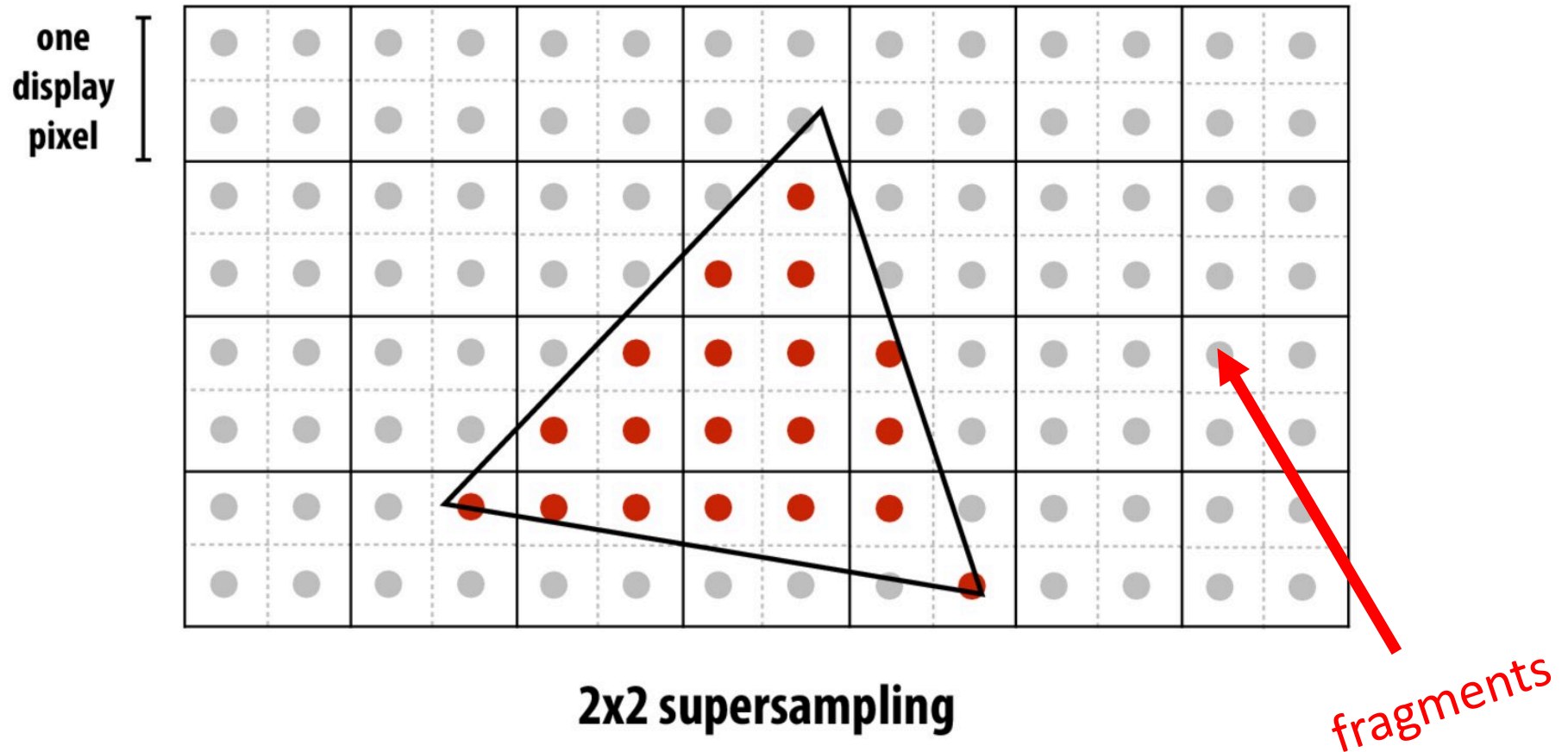
Point sampling: one sample per pixel



Supersampling: step 1

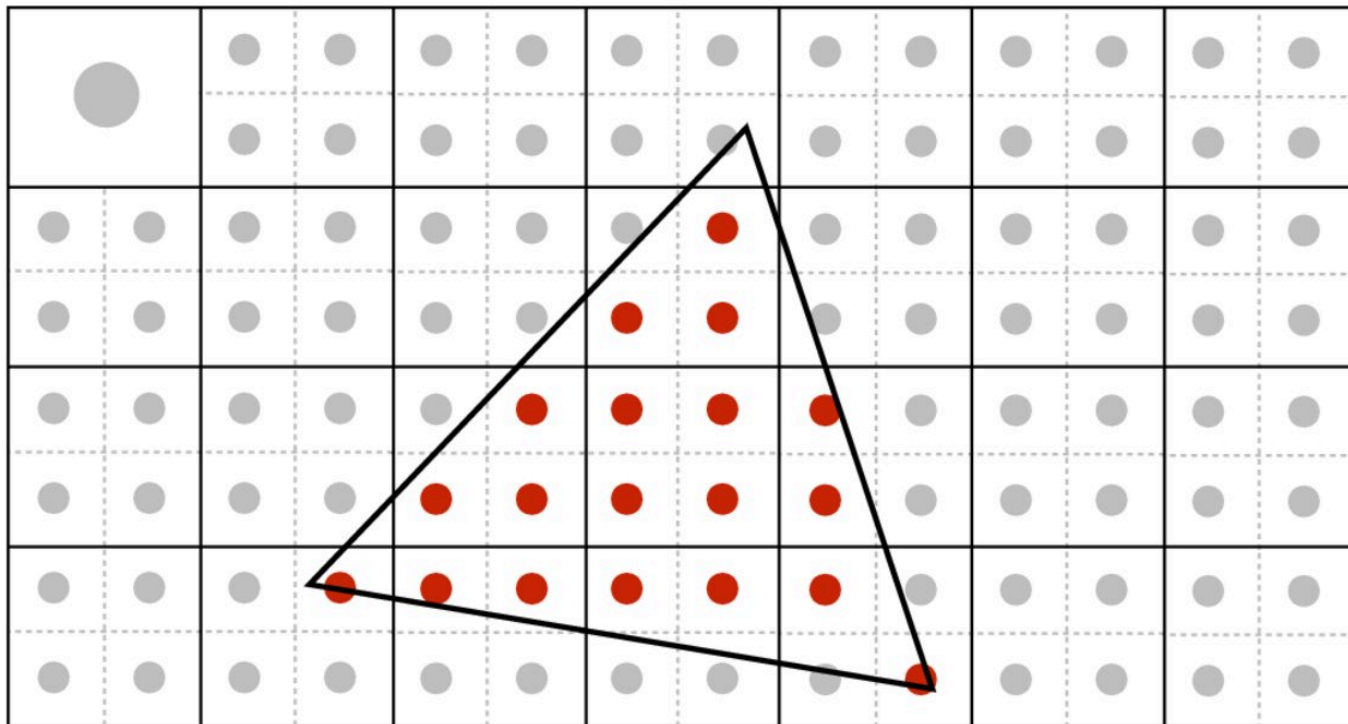
Take $N \times N$ samples in each pixel

(but... how do we use these samples to drive a display, since there are four times more samples than display pixels!)



Supersampling: step 2

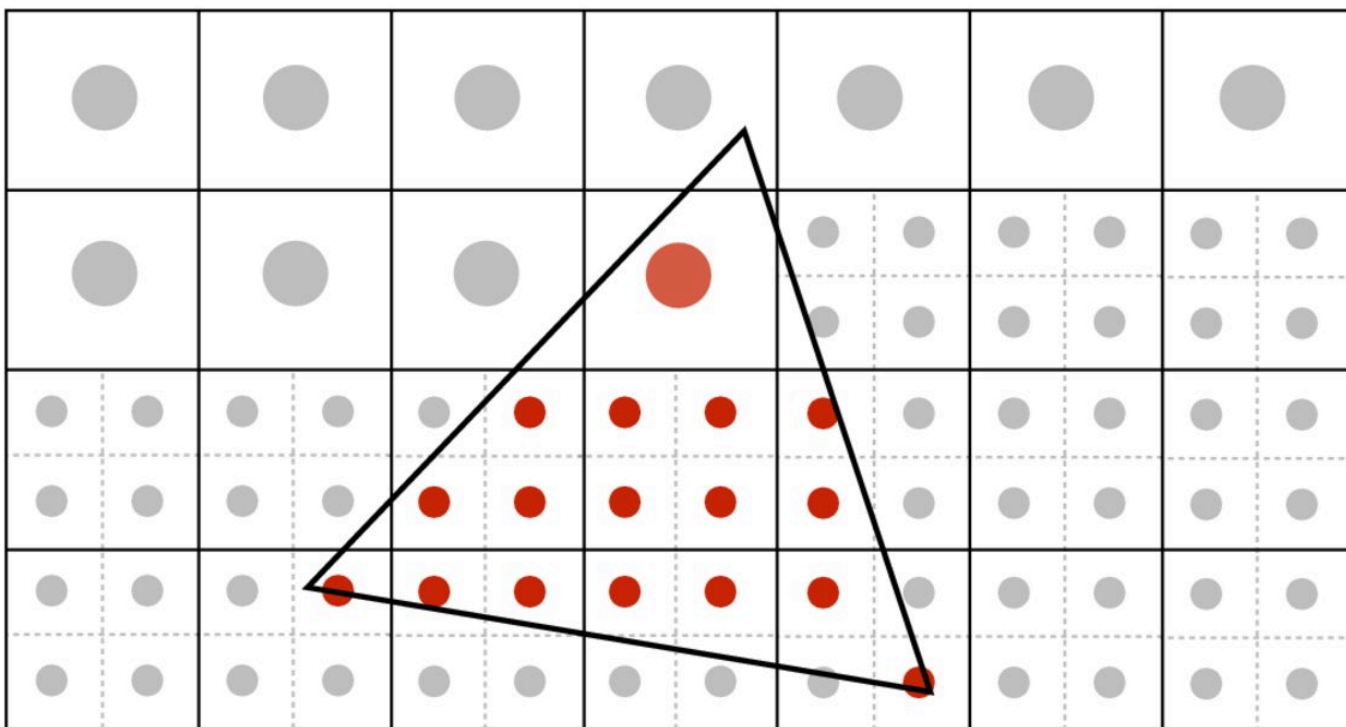
Average the $N \times N$ samples “inside” each pixel



Averaging down

Supersampling: step 2

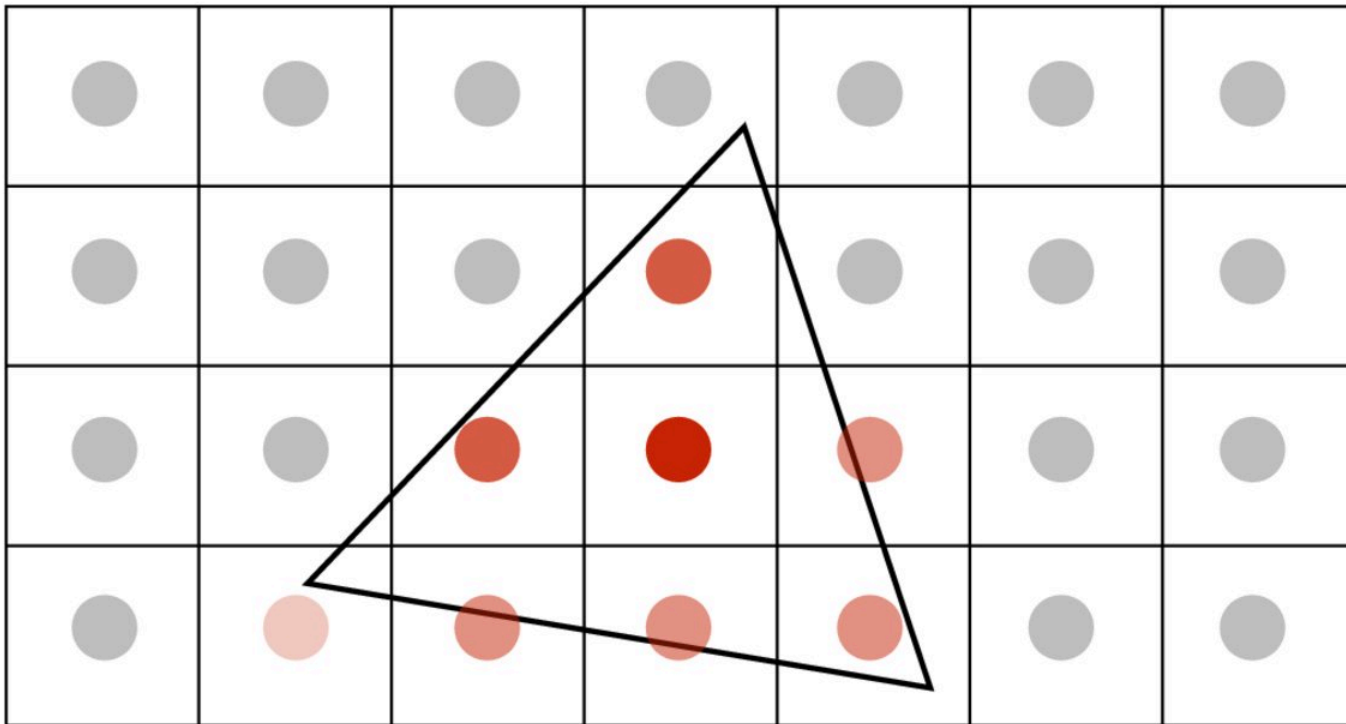
Average the $N \times N$ samples “inside” each pixel



Averaging down

Supersampling: step 2

Average the $N \times N$ samples “inside” each pixel

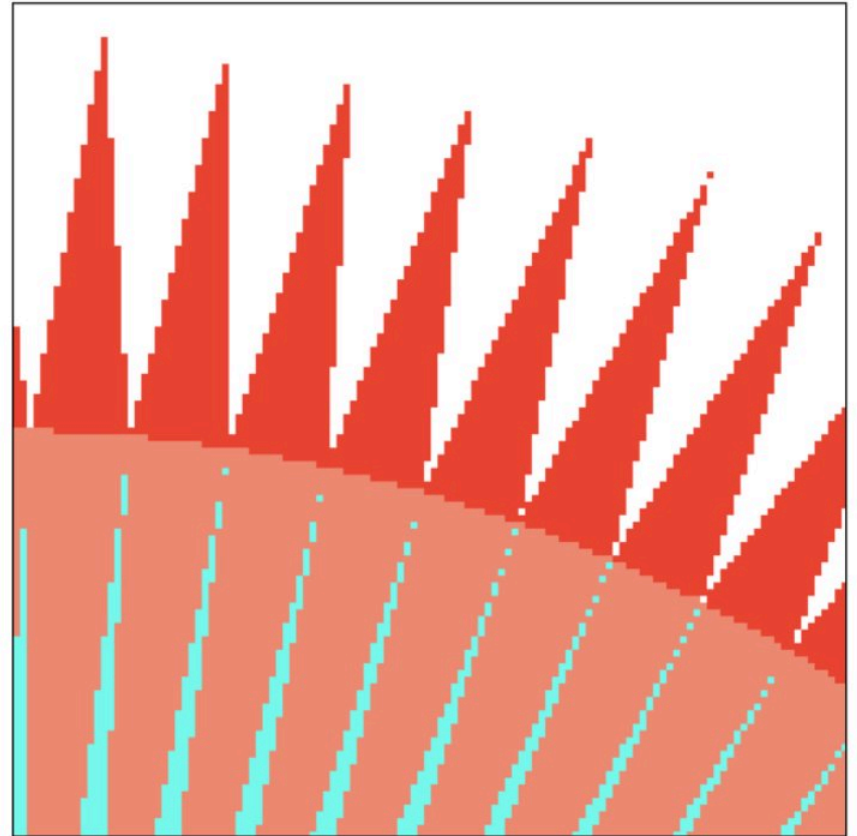
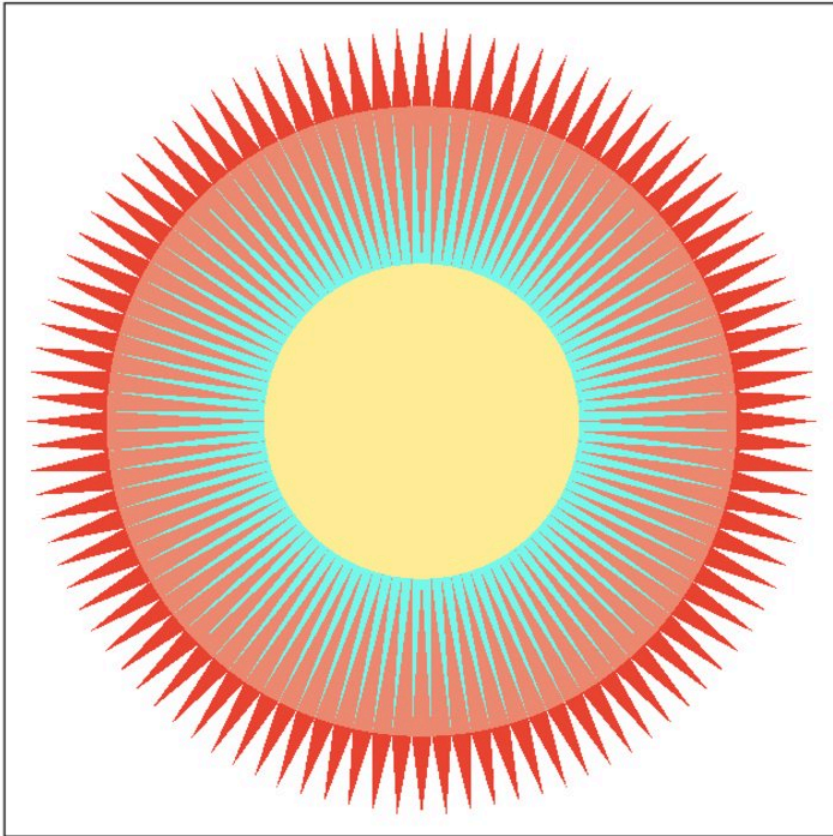


Supersampling: result

This is the corresponding signal emitted by the display

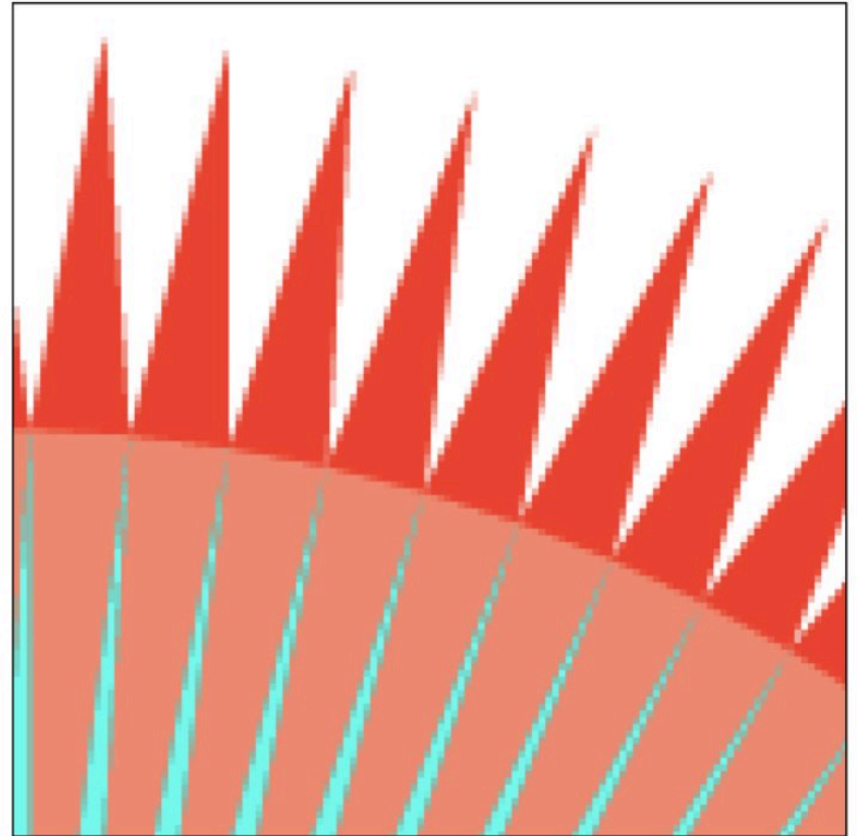
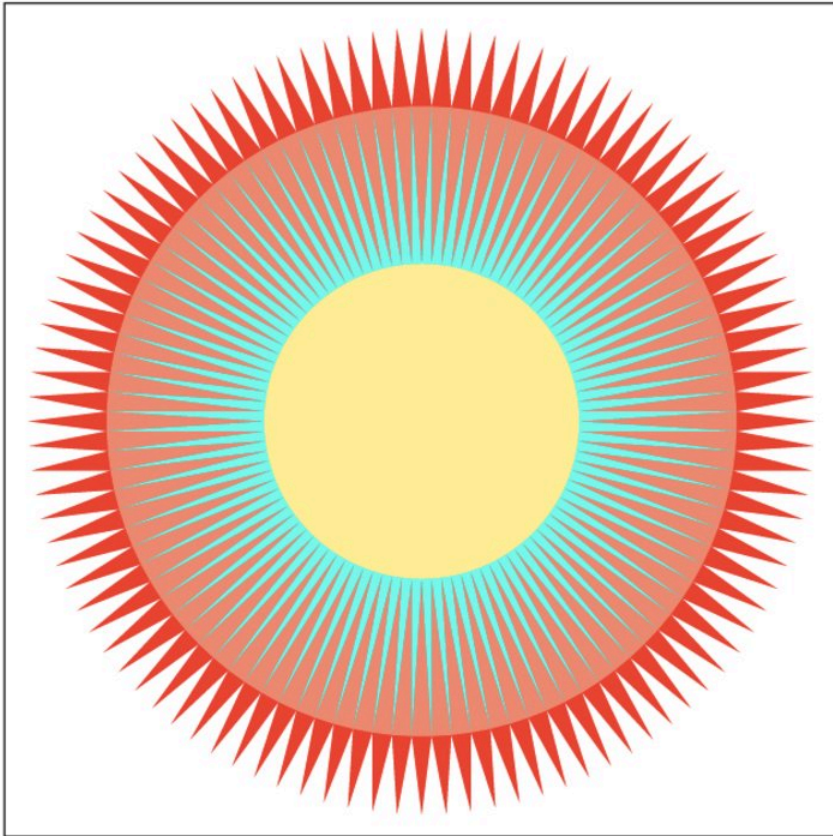
			75%			
		100%	100%	50%		
	25%	50%	50%	50%		

Point sampling



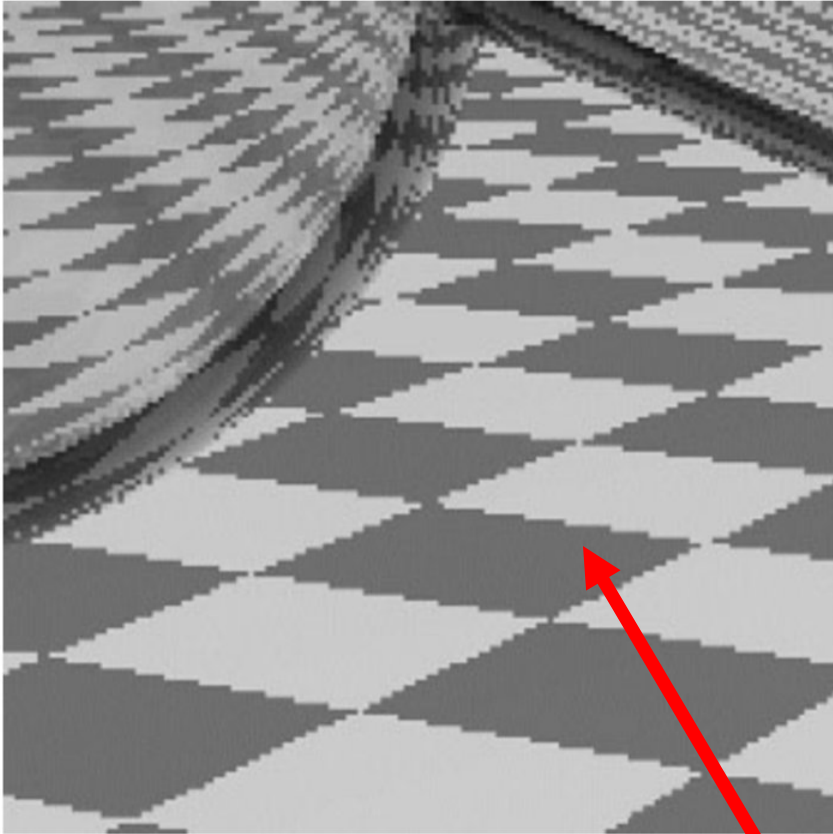
One sample per pixel

4x4 supersampling + downsampling

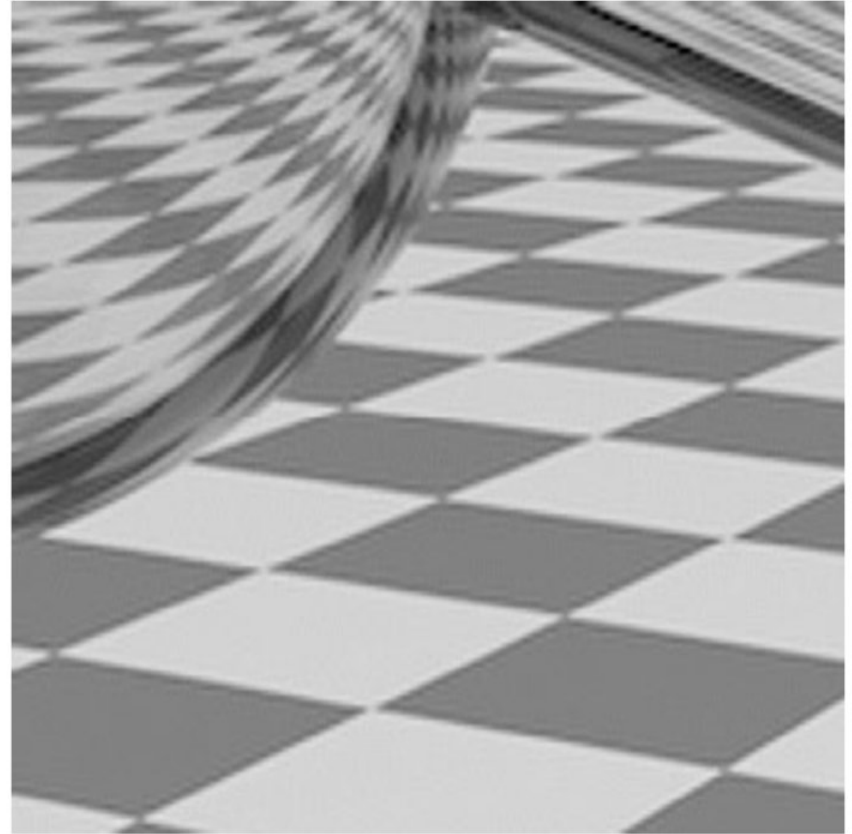


Pixel value is average of 4x4 samples per pixel

Point sampling vs anti-aliasing



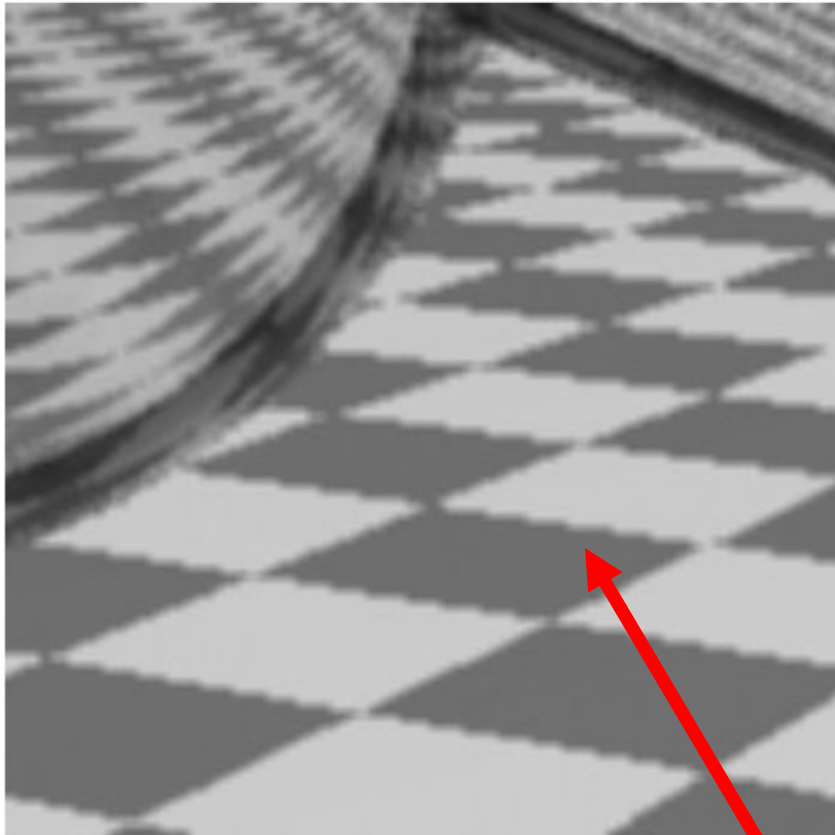
Jaggies



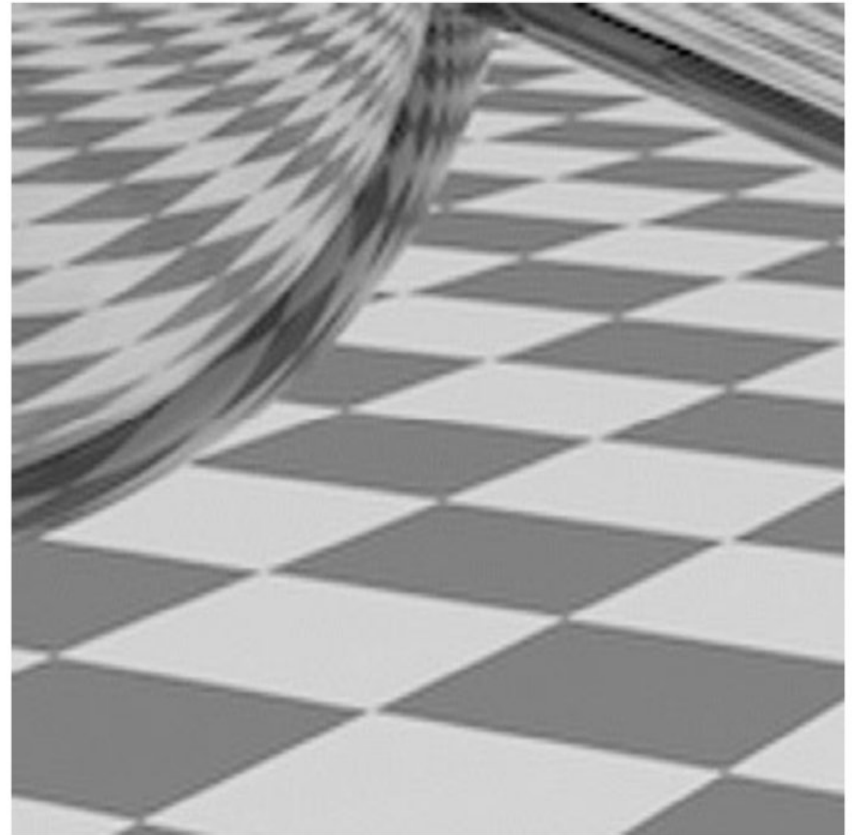
Pre-filtered

*Could I just blur
this and fix it?*

Anti-aliasing vs blurring an aliased result



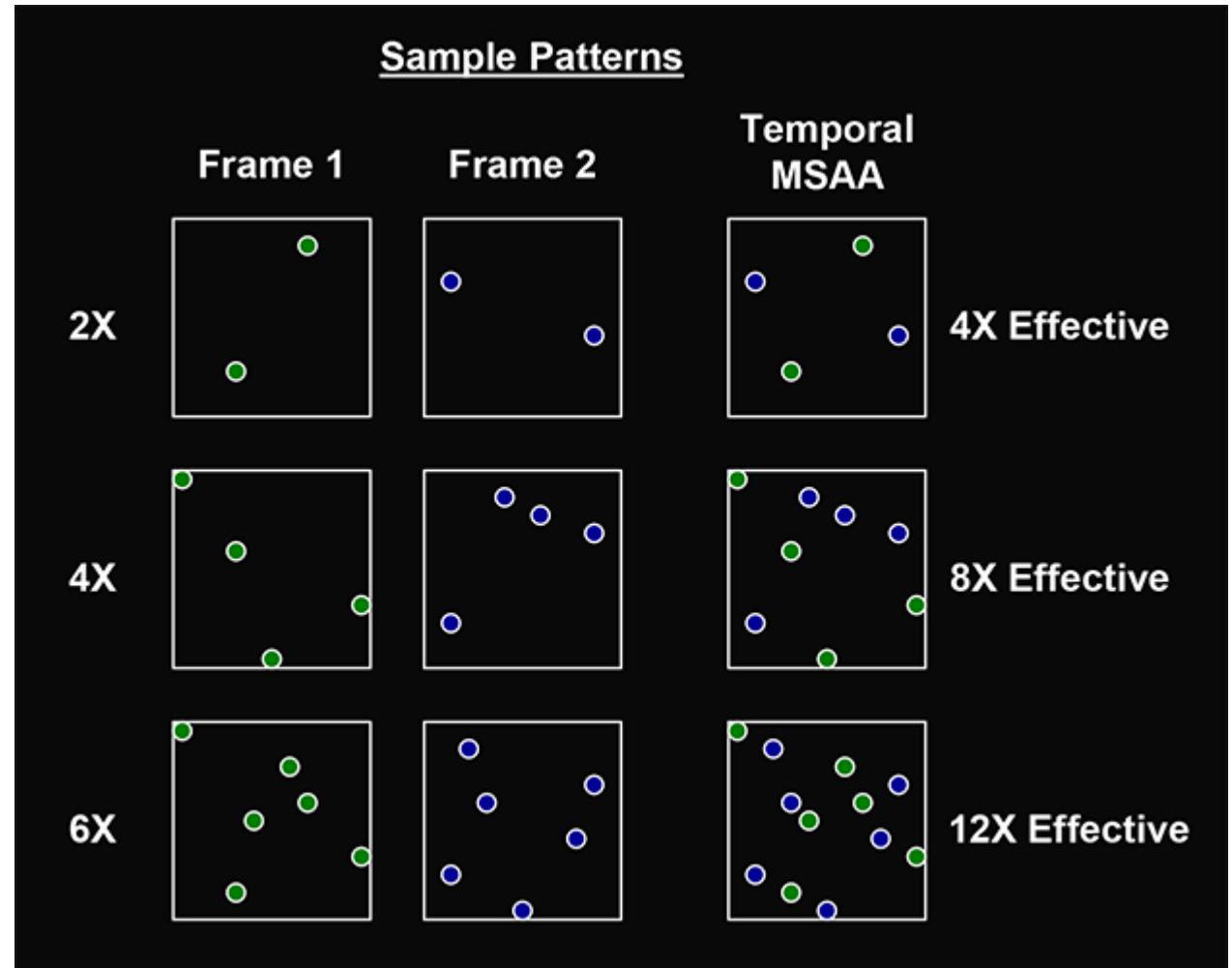
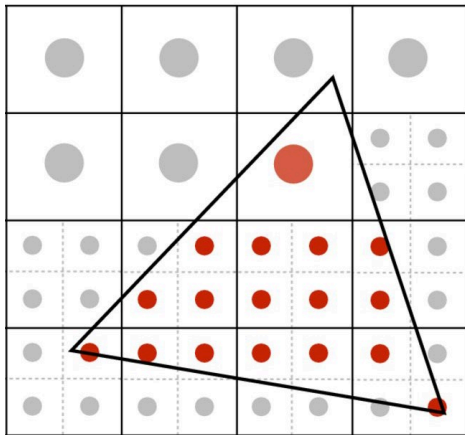
Blurred Jaggies
(Sample then filter)

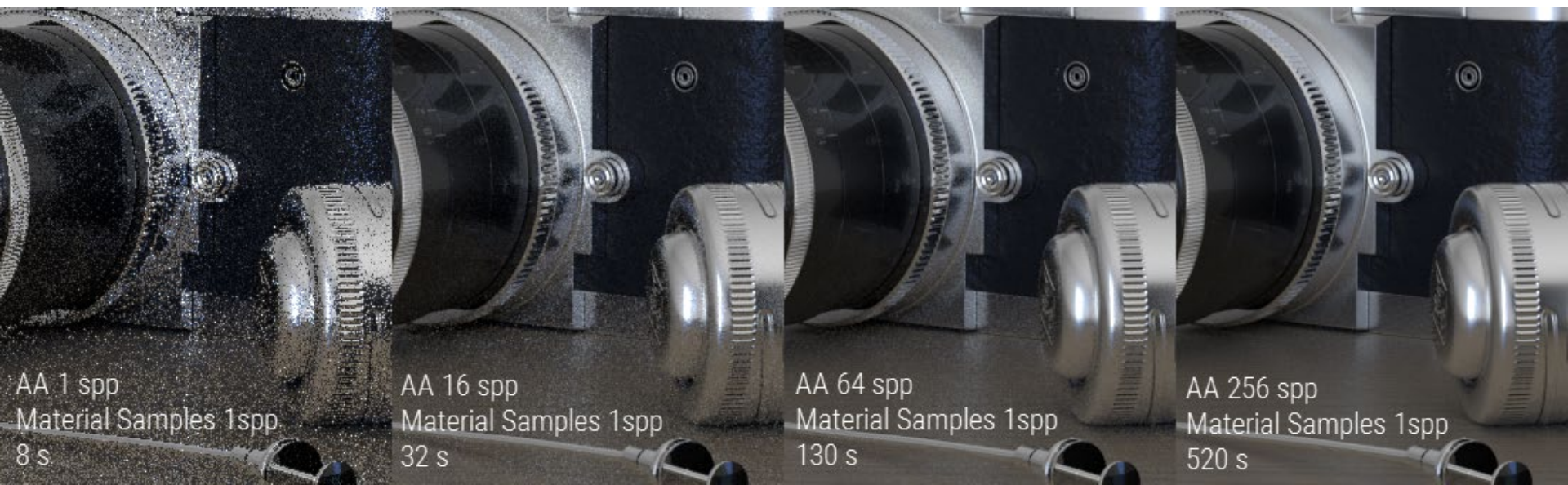
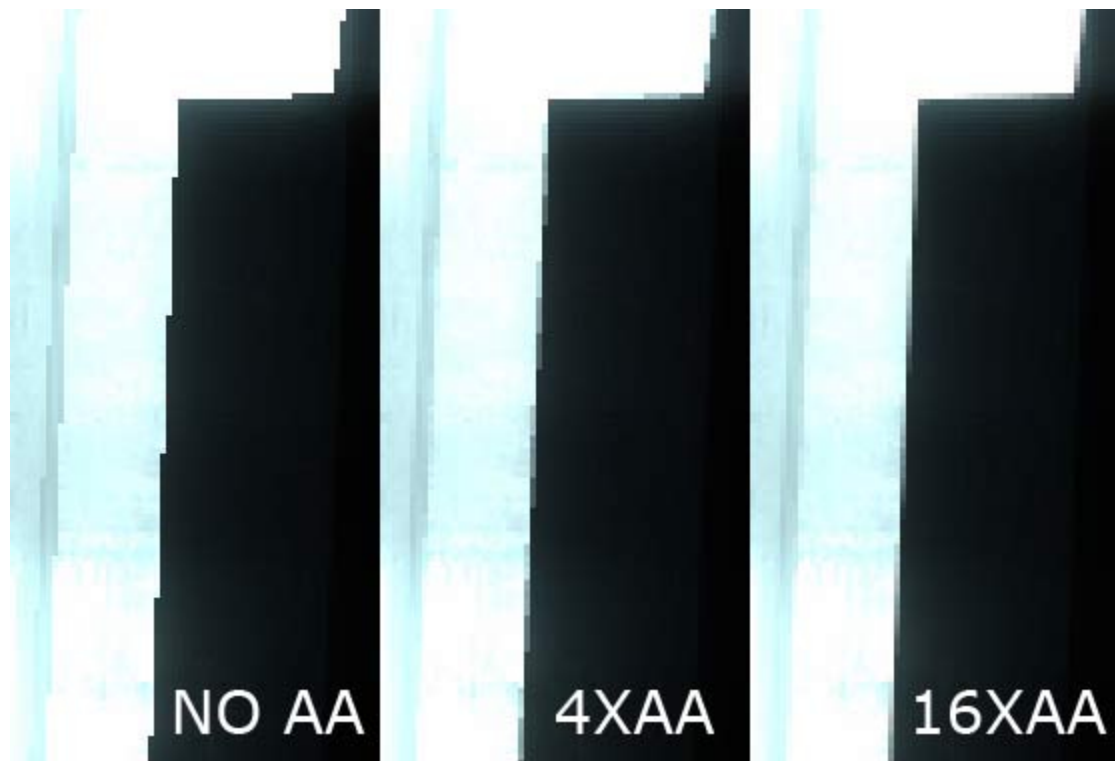


Pre-Filtered
(Filter then sample)

*Once you have
aliasing, its too
late*

How many samples per pixel and where?





no aa	post aa	8x msaa

OpenGL

Specifying Texture Coordinates in OpenGL

```
glBegin(GL_TRIANGLES);  
    glNormal3fv(n1);  
    glTexCoord2f(s1,t1);  
    glVertex3fv(v1);  
  
    glNormal3fv(n2);  
    glTexCoord2f(s2,t2);  
    glVertex3fv(v2);  
  
    glNormal3fv(n3);  
    glTexCoord2f(s3,t3);  
    glVertex3fv(v3);  
glEnd();
```

Example Setup

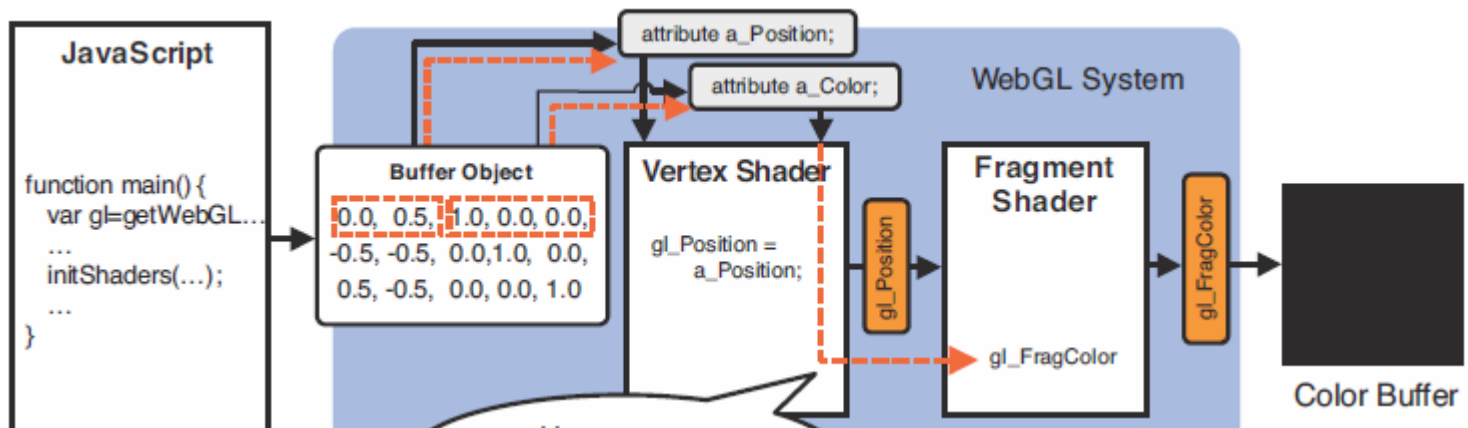
```
glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

gluBuild2DMipmaps(GL_TEXTURE_2D, 3, width, height, GL_RGB,
                 GL_UNSIGNED_BYTE, image);

glTexImage2D(GL_TEXTURE_2D, 0, 3, width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE, image);
```

Multiple attributes



Passing data from vertex to fragment shader

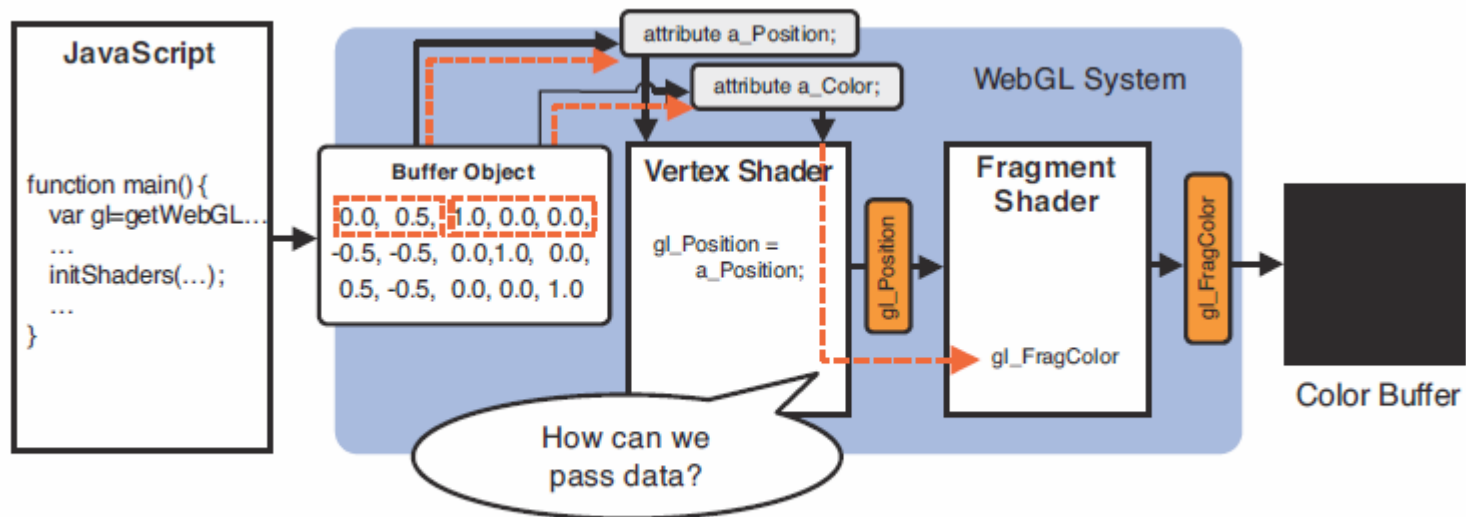


Figure 5.6 Passing data from a vertex shader to a fragment shader

Varying variable

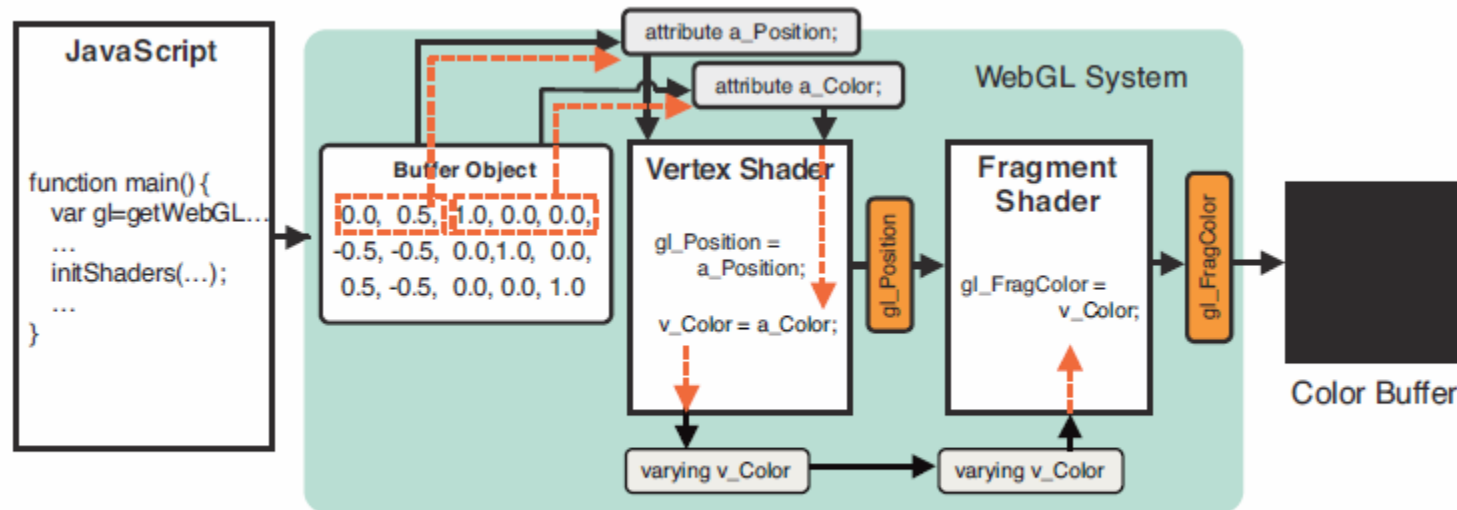


Figure 5.7 The behavior of a varying variable

```

3 var VSHADER_SOURCE =
4   'attribute vec4 a_Position;\n' +
5   'attribute vec2 a_TexCoord;\n' +
6   'varying vec2 v_TexCoord;\n' +
7   'void main() {\n' +
8   '   gl_Position = a_Position;\n' +
9   '   v_TexCoord = a_TexCoord;\n' +
10  '}\n';

```

```

13 var FSHADER_SOURCE =
    ...
17   'uniform sampler2D u_Sampler;\n' +
18   'varying vec2 v_TexCoord;\n' +
19   'void main() {\n' +
20   '   gl_FragColor = texture2D(u_Sampler, v_TexCoord);\n' +
21   '}\n';

```



```
100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture();  // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image();  // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){  <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }
```

Takes a while to load, so
set up a callback

```
100 function initTextures(gl, n)                                     <- (Part4)
101   var texture = gl.createTexture(); // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image(); // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){  <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }
```

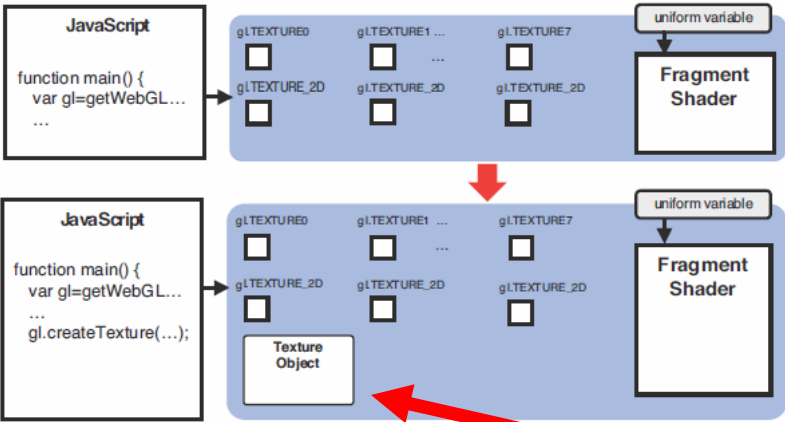


Figure 5.22 Create a texture object

```

100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture();  // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image();  // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){    <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }

```

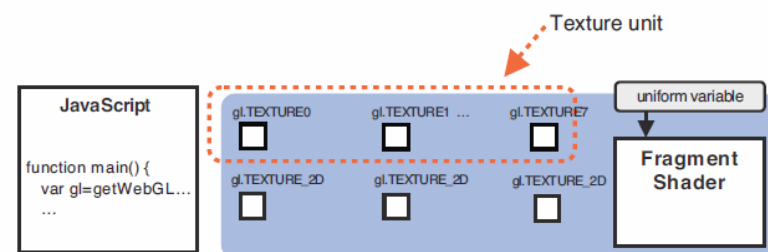


Figure 5.25 Multiple texture units managed by WebGL

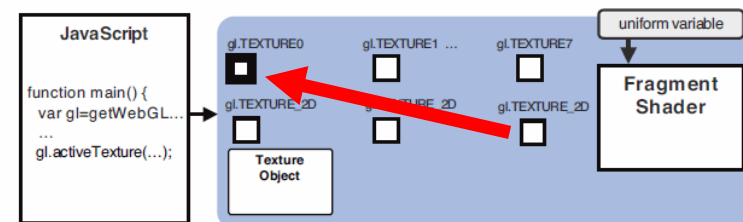


Figure 5.26 Activate texture unit (gl.TEXTURE0)

```
100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture();  // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image();  // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){  <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }
```

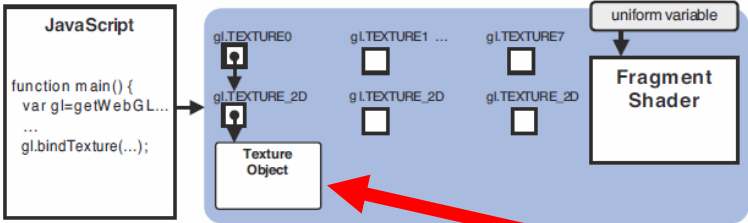


Figure 5.27 Bind a texture object to the target

```

100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture(); // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image(); // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){    <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }

```

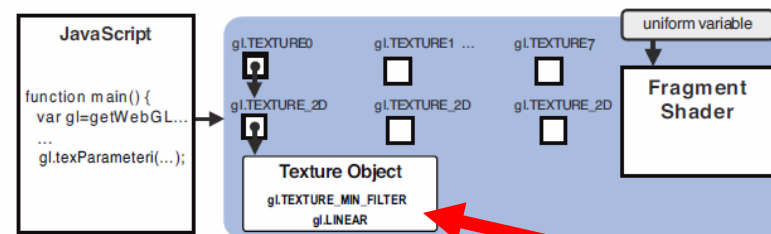


Figure 5.29 Set texture parameter

```

gl.LINEAR, gl.NEAREST, gl.NEAREST_MIPMAP_NEAREST,
gl.LINEAR_MIPMAP_NEAREST, gl.NEAREST_MIPMAP_LINEAR (default value),
gl.LINEAR_MIPMAP_LINEAR.

```



```

100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture();  // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image();  // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){    <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }

```

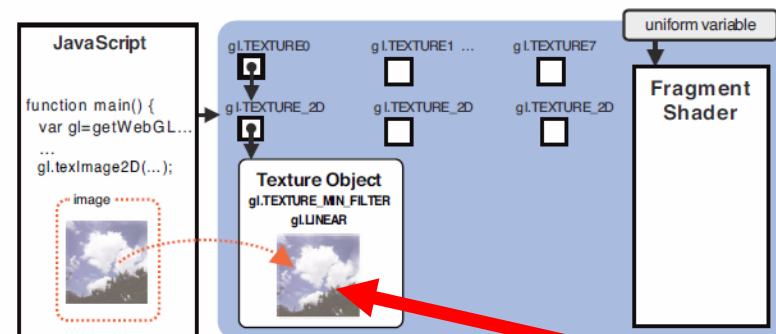


Figure 5.30 Assign an image to the texture object

```

100 function initTextures(gl, n)                                <- (Part4)
101   var texture = gl.createTexture();  // Create a texture object
102   ...
107   // Get the storage location of the u_Sampler
108   var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
109   ...
114   var image = new Image();  // Create an image object
115   ...
119   // Register the event handler to be called on loading an image
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
121   // Tell the browser to load an image
122   image.src = '../resources/sky.jpg';
123
124   return true;
125 }
126
127 function loadTexture(gl, n, texture, u_Sampler, image){    <- (Part5)
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
129   // Enable the texture unit 0
130   gl.activeTexture(gl.TEXTURE0);
131   // Bind the texture object to the target
132   gl.bindTexture(gl.TEXTURE_2D, texture);
133
134   // Set the texture parameters
135   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
136   // Set the texture image
137   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
138
139   // Set the texture unit 0 to the sampler
140   gl.uniform1i(u_Sampler, 0);
141   ...
144   gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
145 }

```

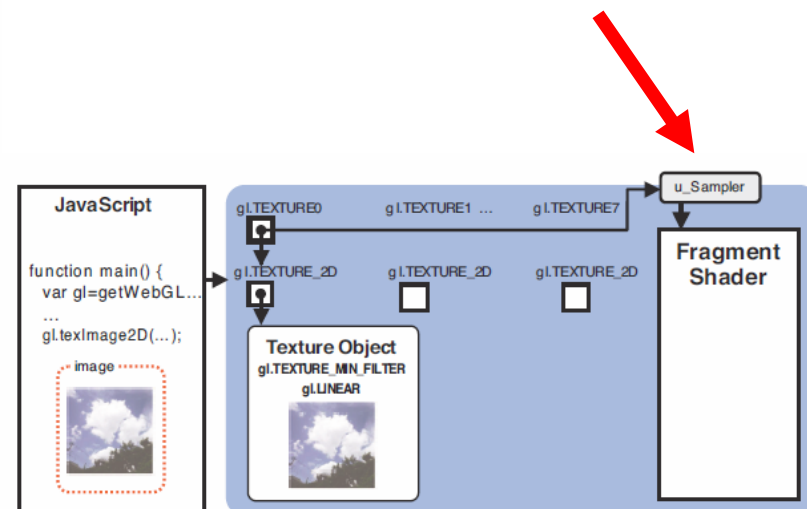


Figure 5.31 Set texture unit to uniform variable

```

13 var FSHADER_SOURCE =
    ...
17 'uniform sampler2D u_Sampler;\n' +
18 'varying vec2 v_TexCoord;\n' +
19 'void main() {\n' +
20 '  gl_FragColor = texture2D(u_Sampler, v_TexCoord);\n' +
21 '}\n';

```

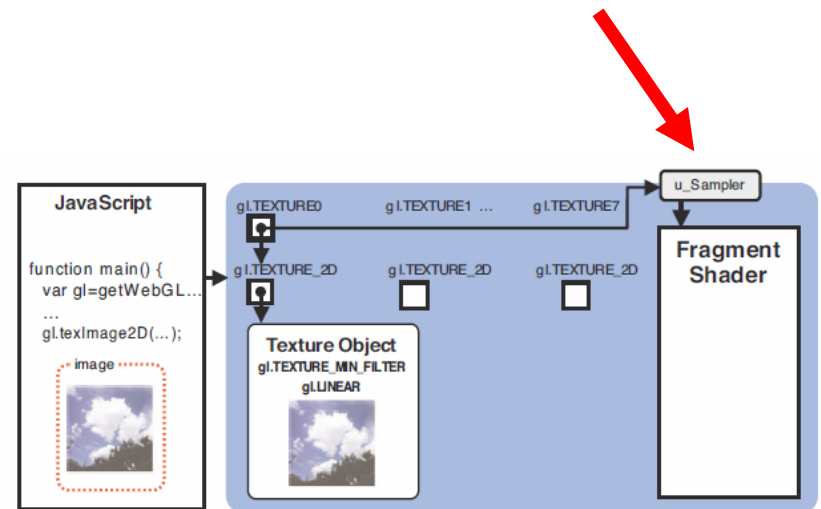


Figure 5.31 Set texture unit to uniform variable

Beyond 2D Colored Surfaces

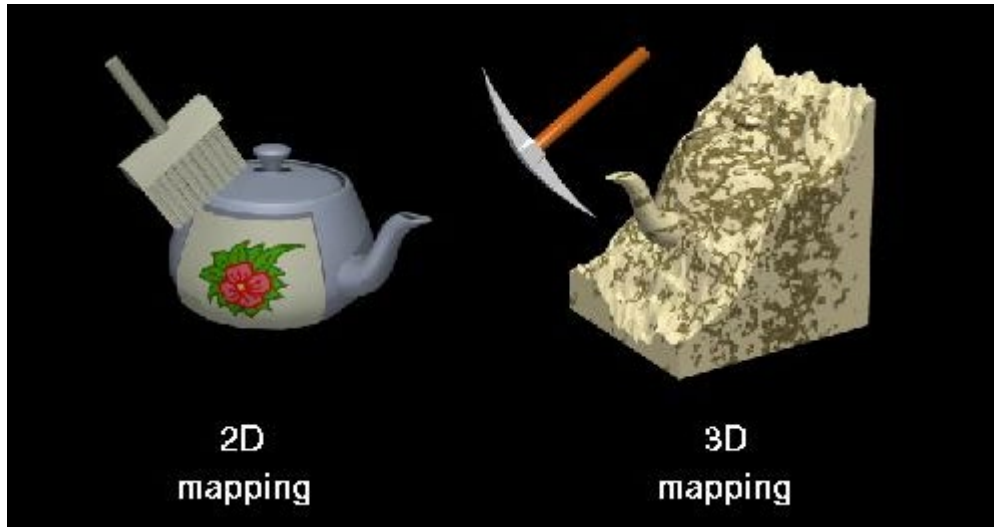
Solid Texturing

3D Texture Maps

- Instead of a texture image, define a texture volume
- Next, define a 3D parameterization:

$$f : (x, y, z) \rightarrow (u, v, w)$$

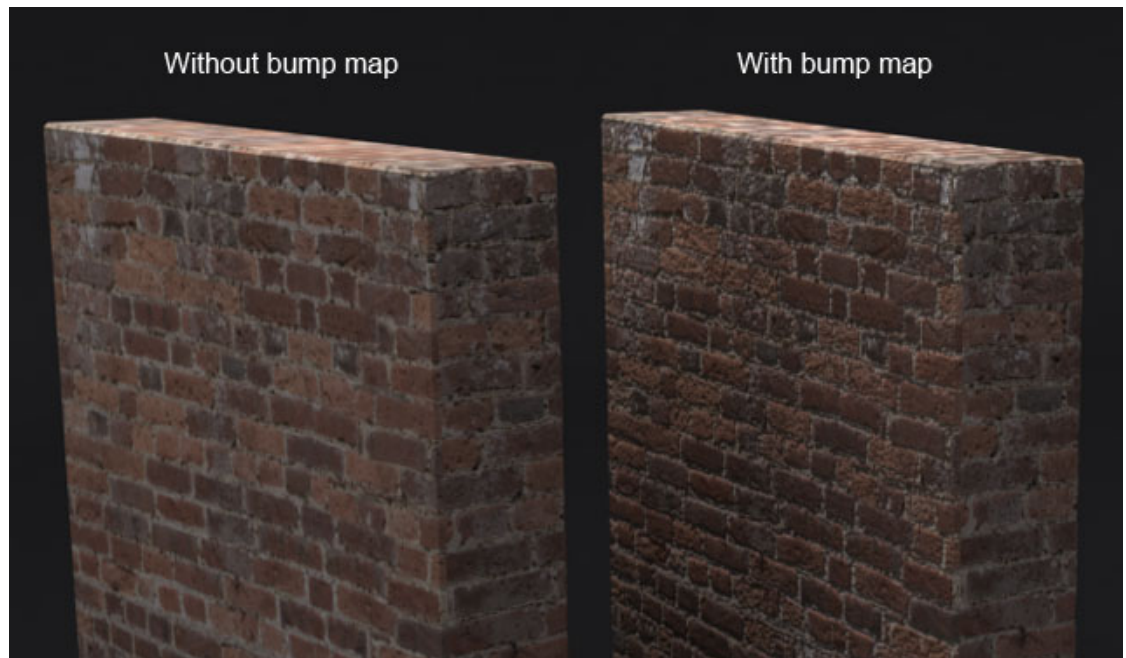
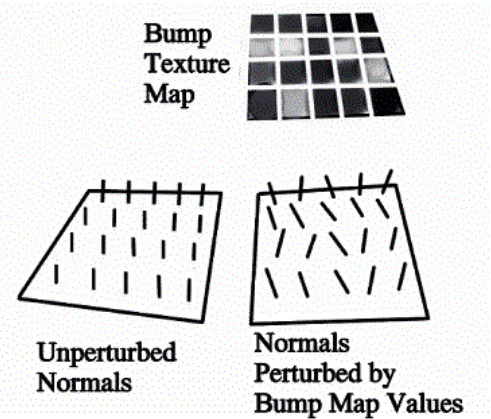
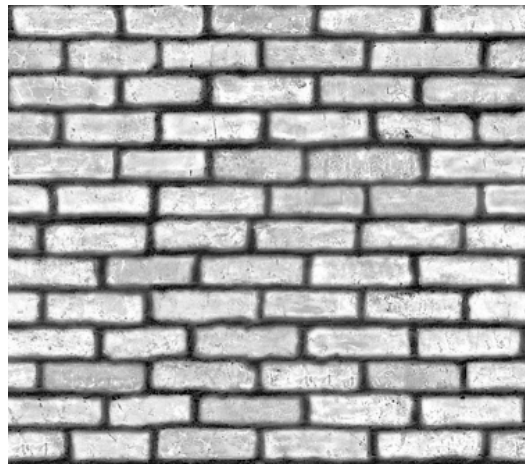
- Useful for procedurally generated textures from noise functions
- Works well for surfaces without natural 2D parameterizations
- But typically results in tremendous amounts of wasted memory



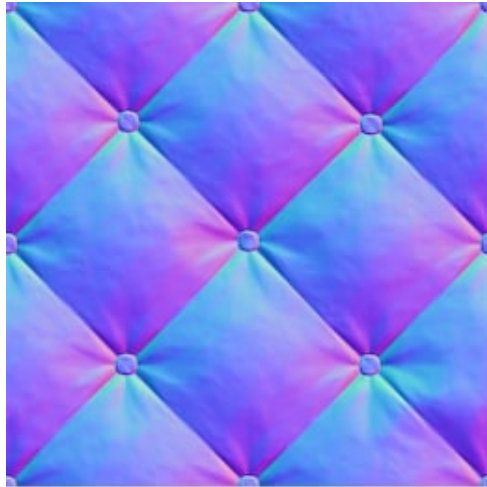
↑
Example of using
Perlin Noise

Bump mapping

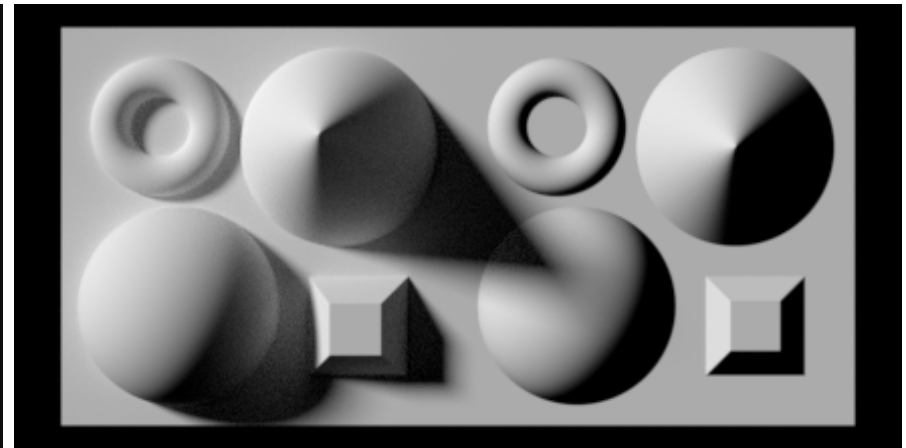
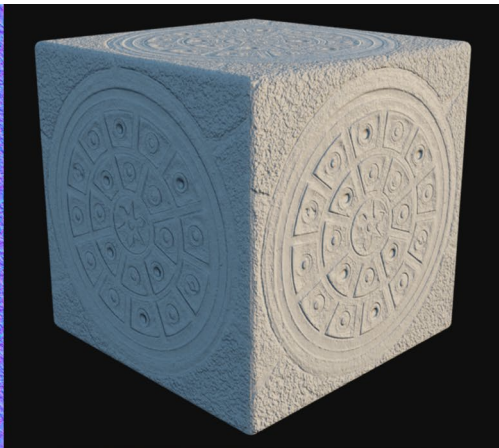
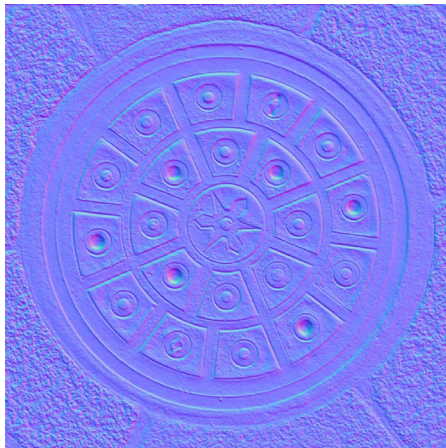
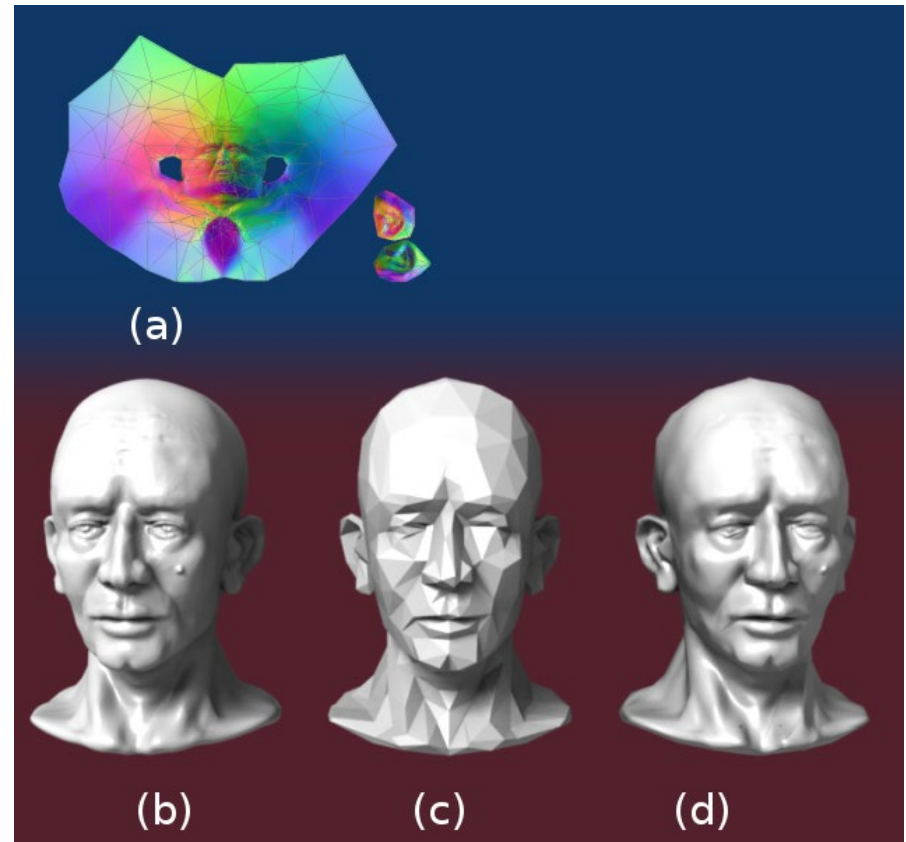
From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Intergraph Computer Systems



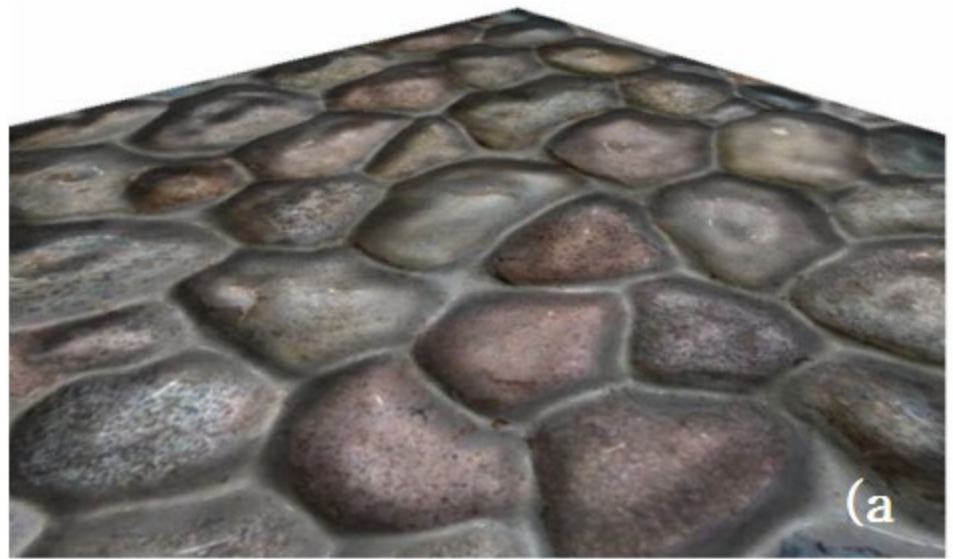
Normal Maps



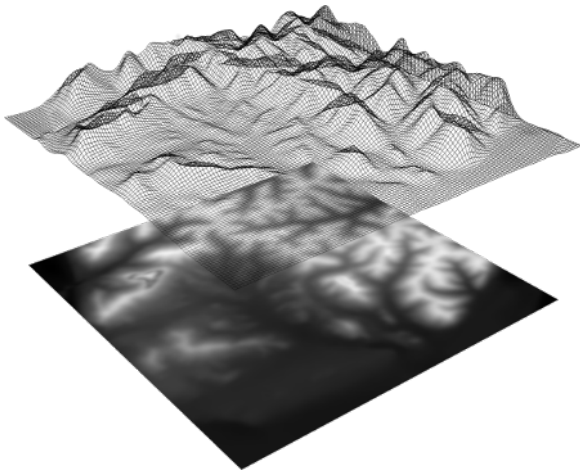
shutterstock.com • 756204394



Color Map



Displacement Map

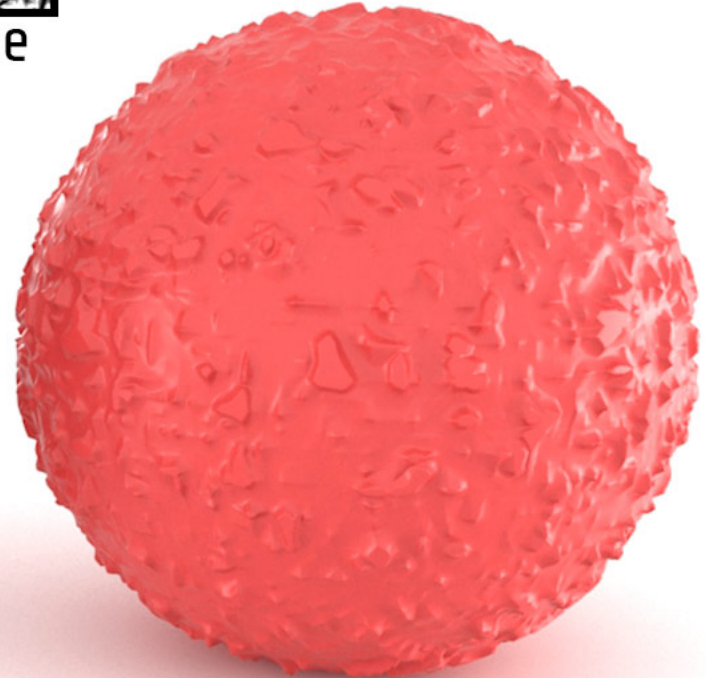




source

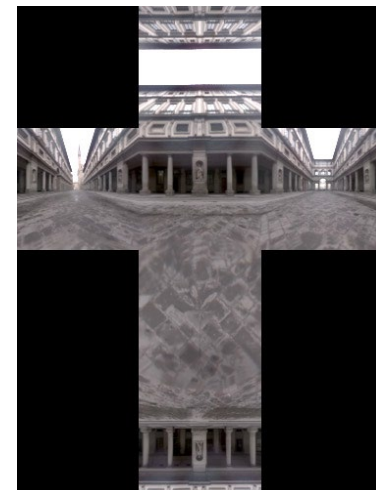
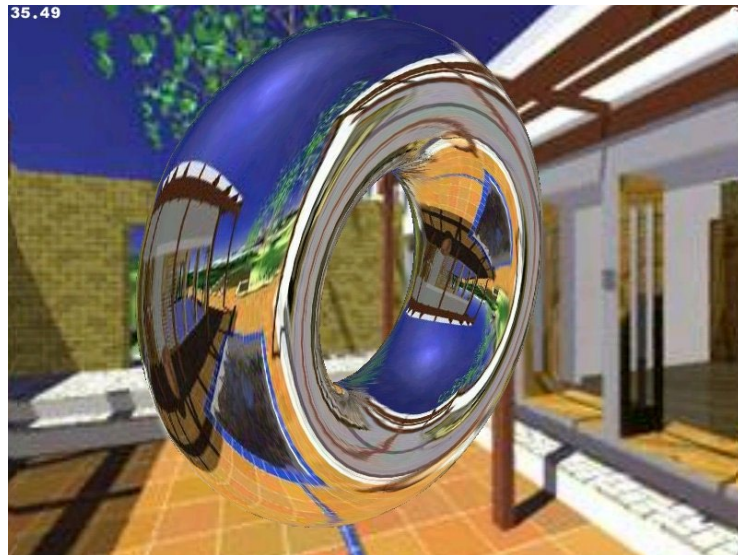
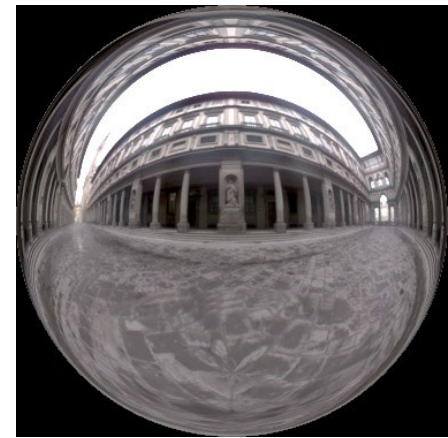
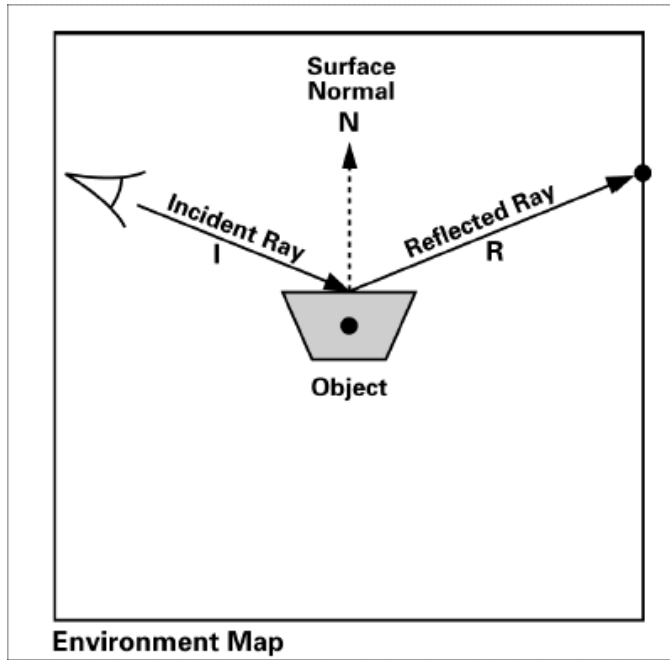


Bump Map

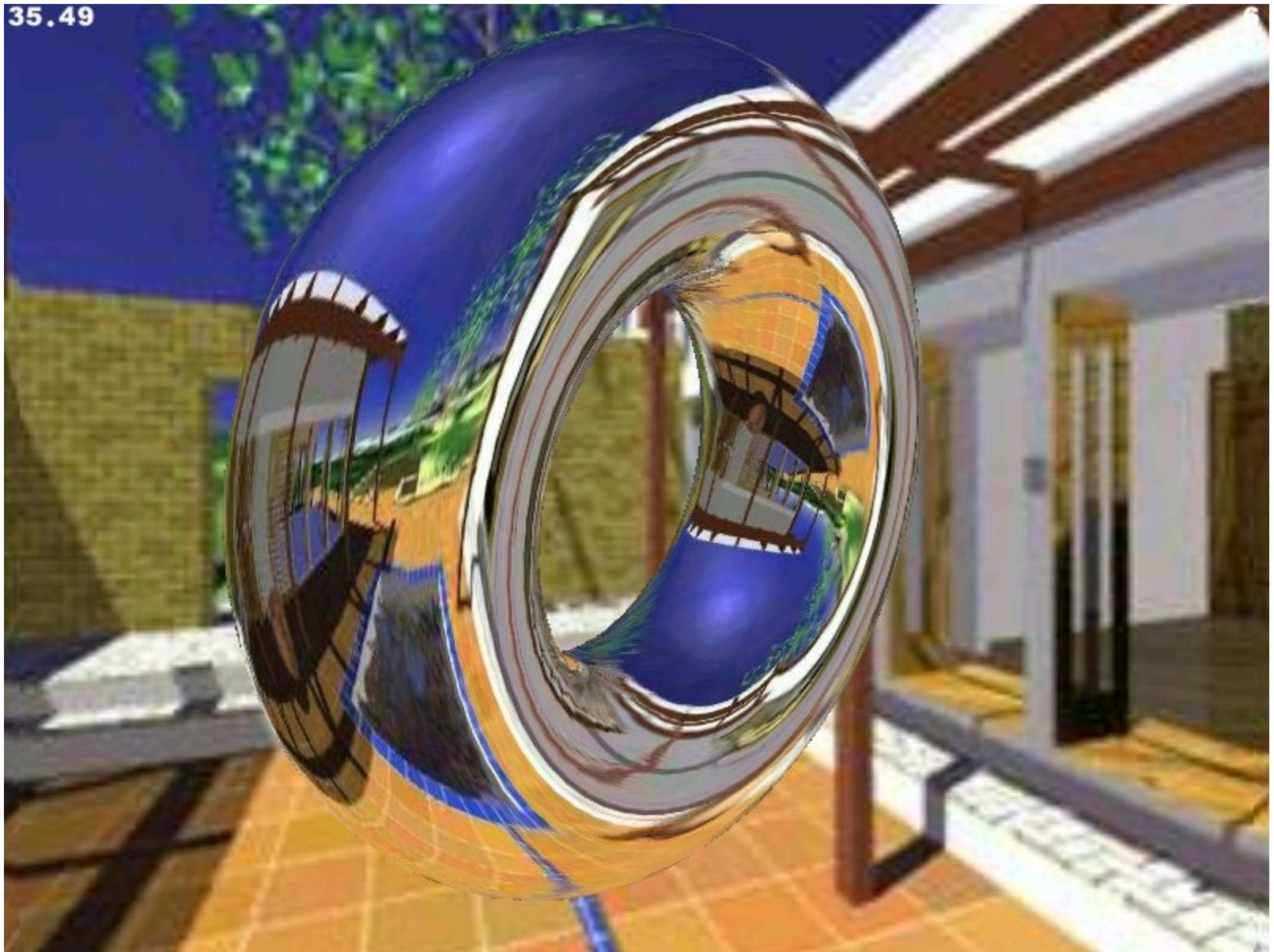


Displacement Map

Environment mapping



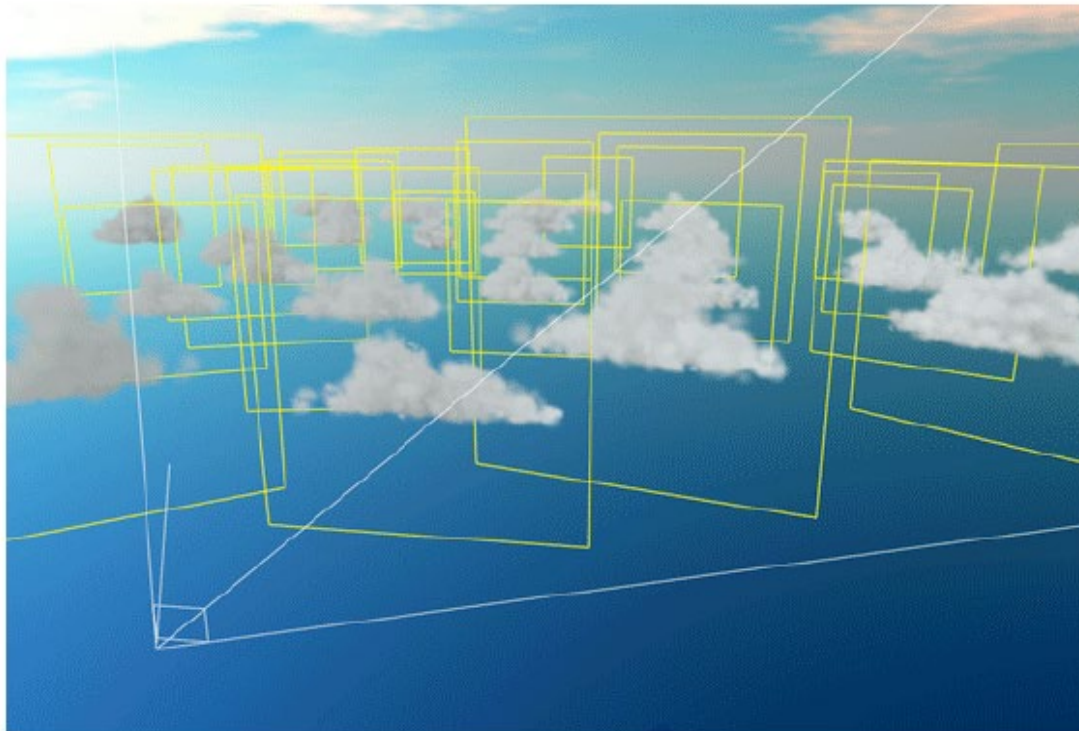
35.49



Billboarding

Texture Sprites

- Place image on a polygon that always rotates to face the viewer
- Use transparency to mask away unwanted geometry
- Useful for small or far-away effects (like clouds and trees)



Administrative

Due Dates

- Due Yesterday
 - Quiz 2
- Due next Monday
 - Lab 2 (Blocky Animal)

Q&A

End