

COMP3271 Computer Graphics

Texture Mapping

2019-20

Objectives

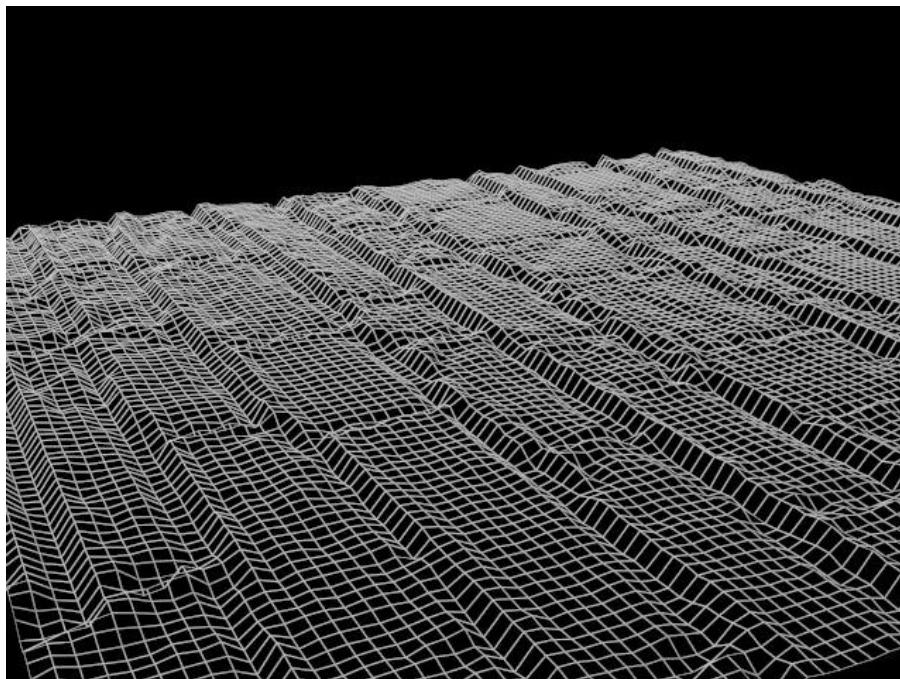
What texture mapping is for

Understand the basic texture mapping process

How Do We Model Intricate Surface Detail?

Approach #1: Explicit geometric representation

- actual polygons that model all the surface variations
- up to some finest level of detail
- may generate a *lot* of polygons

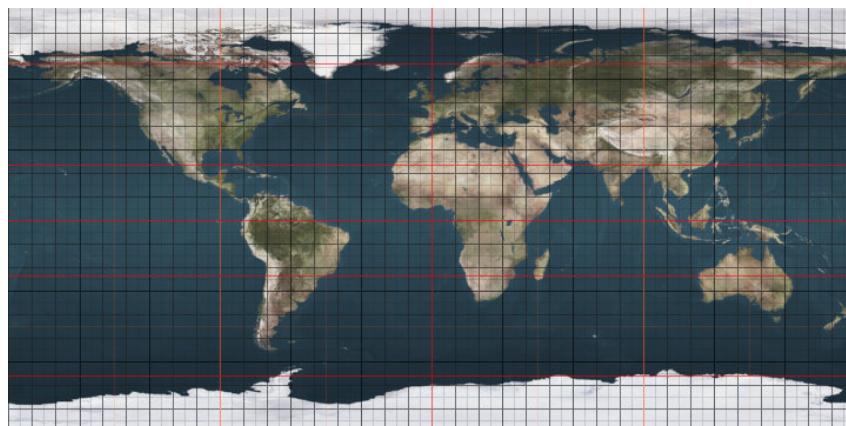


<https://ch.3dexport.com/3dmodel-3d-stone-wall-55616.htm>

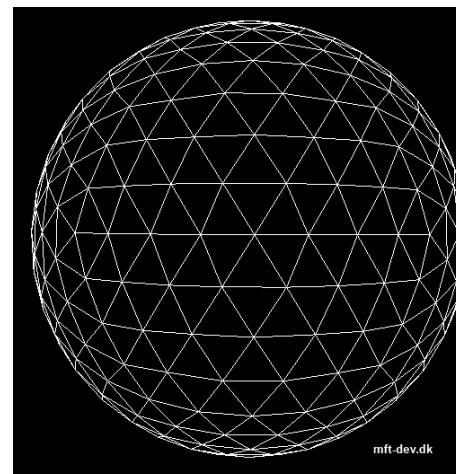
How Do We Model Intricate Surface Detail?

Approach #2: Geometry + texture images

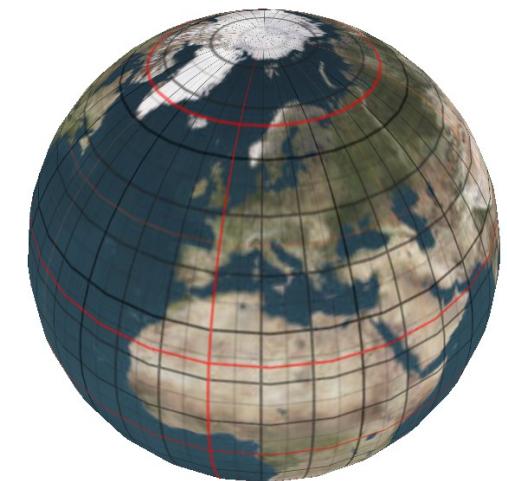
- geometry only describes the general shape of the object
- paste an image onto the surface to give the appearance with fine details



+



=



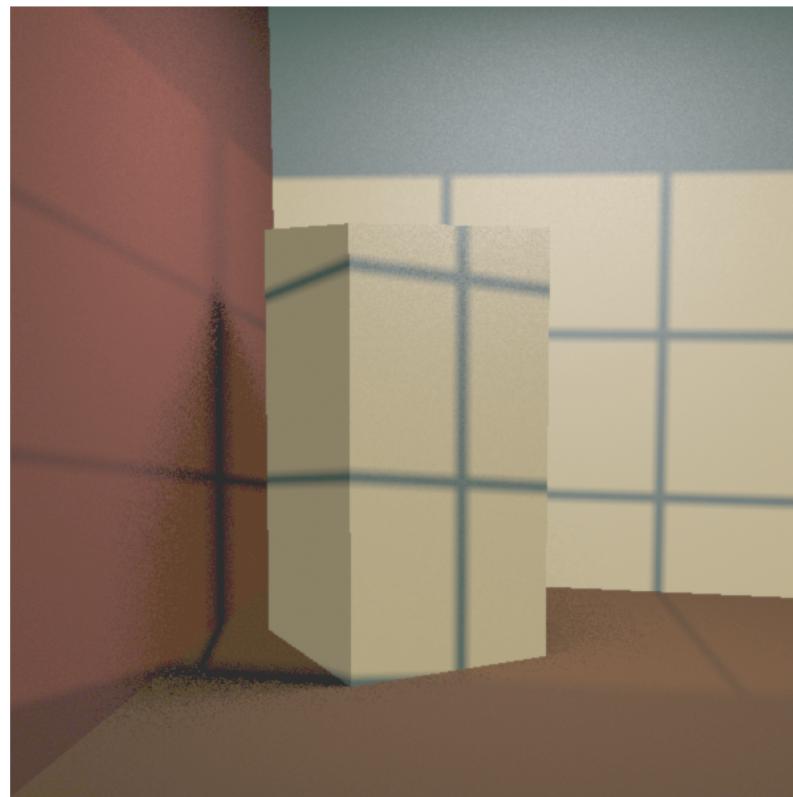
https://commons.wikimedia.org/wiki/File:OpenGL_Tutorial_Textured_Spheres.jpg

<https://mft-dev.dk/uv-mapping-sphere/>

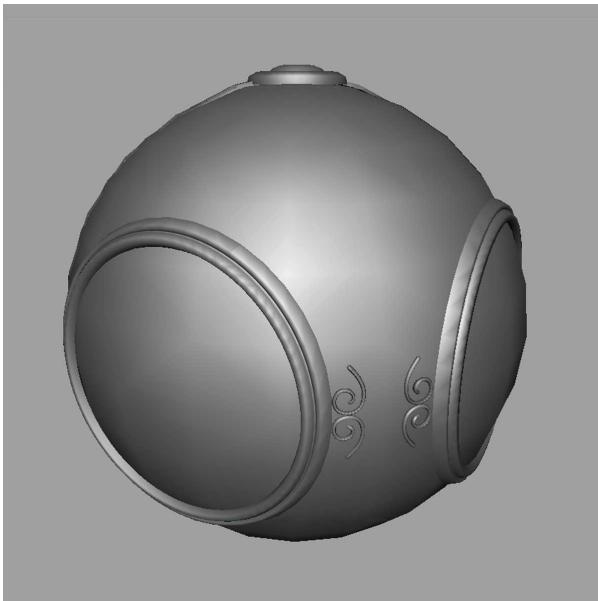
The Importance of Surface Texture

Objects in the real world have rich, detailed surface textures

- to produce as-realistic-as-possible scenes, we must replicate this detail
- uniformly colored surfaces only get us so far



Texture Mapping



geometric model



texture mapped

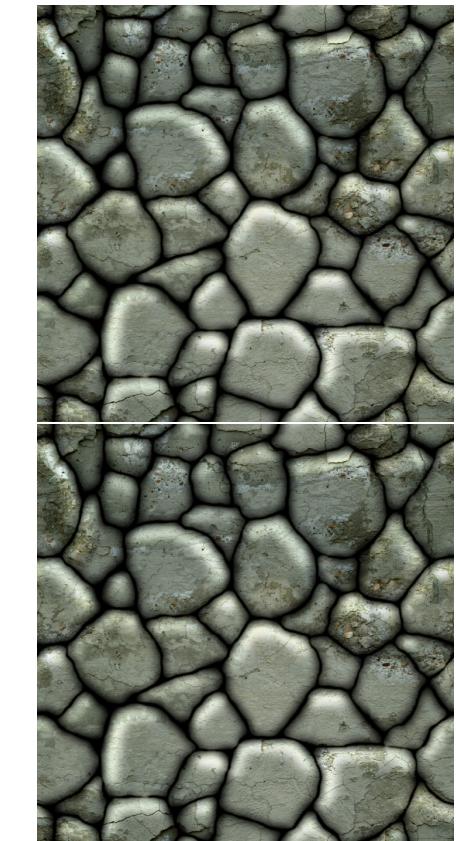


Often We Use Simple Patterns

Generally useful for skin, bricks, stucco, granite, ...

Typically need to repeat texture over the object

- must make sure there are no seams when texture is tiled



Or Given a Model and a Single Texture

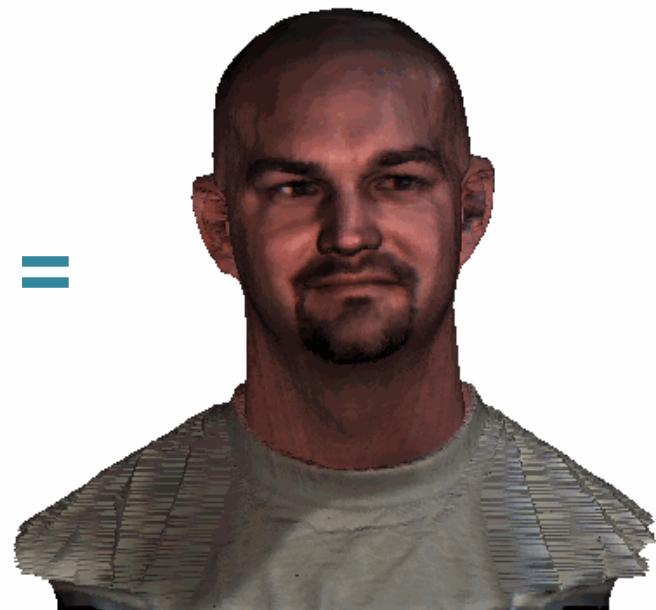
Wrap the Texture onto the Model



+



=



Sample model from www.cyberware.com

Framework for Texture Mapping

The texture itself is just a 2-D raster image

- acquired from reality, hand-painted, or procedurally generated

Establish a correspondence between surface points & texture



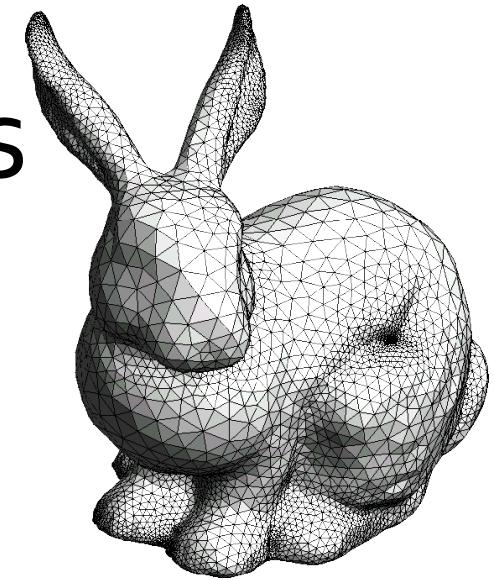
When shading a particular surface point

- look up the corresponding pixel in the texture image
- final color of surface point will be a function of this pixel

Texturing Polygonal Models

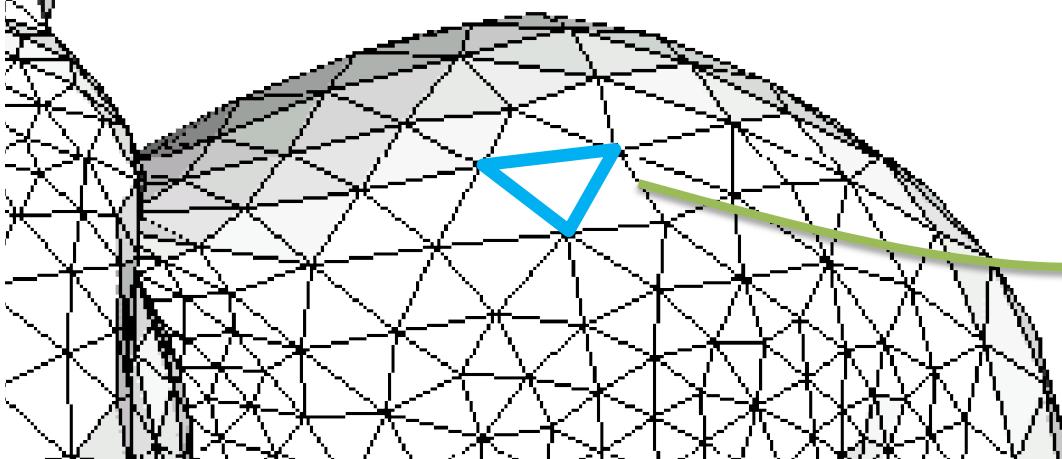
Polygonal models are not so easy

- they don't have a natural 2-D parameterization
- we need to create one



For each vertex, we specify a texture coordinate

- a (u, v) pair that maps that point into the texture image
- a triangle on the surface will be mapped to a triangle in texture
- can interpolate texture coordinates over the triangle
- note that the size of the triangle may be quite different



Texturing and Rasterization

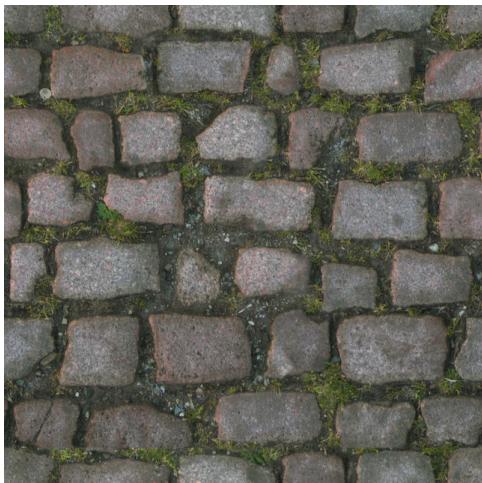
During rasterization, we traverse the pixels of a triangle

- at each pixel we interpolate the correct texture coordinate
- and we retrieve the corresponding **texel** (texture element)

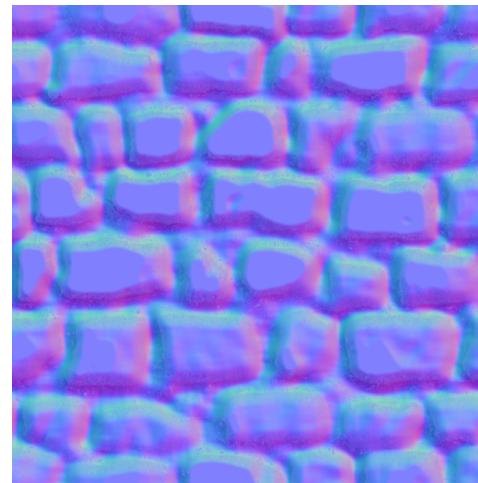
What do we do with the contents of the texel?

- color — use it to fill in the current pixel
- reflectance — coefficient for illumination equation (e.g., k_d)
- transparency — an alpha value
- and many others

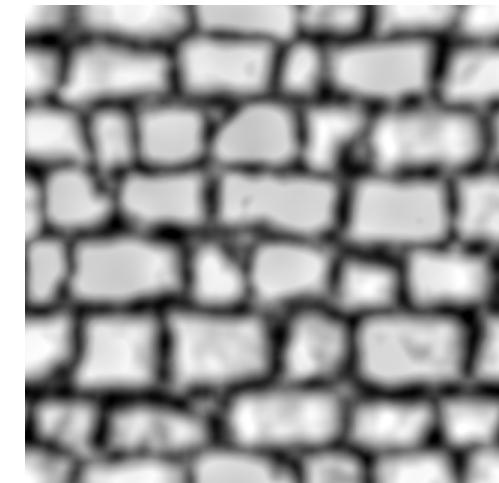
Examples of Other Mappings



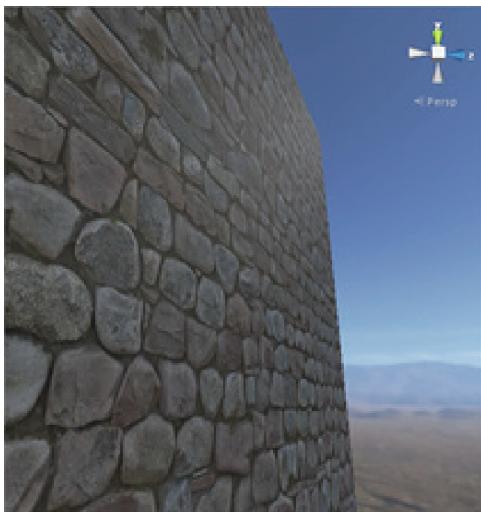
Texture map
texel as diffuse color



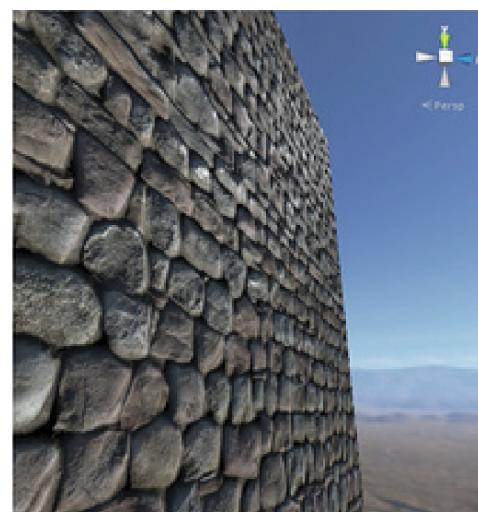
Normal map
texel as normal vectors



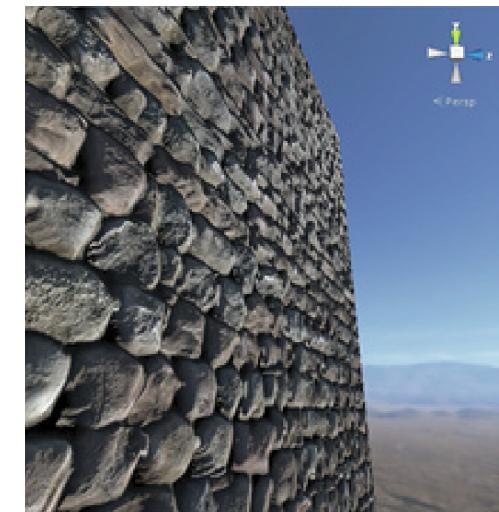
Displacement map
texel as distance from surface



Texture map only



Texture map + normal map



Texture map + normal map
+ displacement map

[Unity]

Texturing with OpenGL

First, turn on texturing — `glEnable(GL_TEXTURE_2D)`

Next, pass the actual texture image to OpenGL

- `glTexImage2D(GL_TEXTURE_2D, level, channels, width, height, border, format, type, image)`
- for now, `level=0` and `border=0`
- `channels` is usually 3 (RGB)
 - with `format=GL_RGB` and `type=GL_UNSIGNED_BYTE`

Have lots of options to control texturing behavior

- see `glTexEnvf()` and `glTexParameterf()` for details
 - how is the color of the texture applied to the surface?
 - texture coordinates clamped to $[0,1]$ or do they wrap around?

OpenGL Texture Modes

How is texture color applied to surface point?

OpenGL Texture modes:

- Determines how the contents of the texture are interpreted

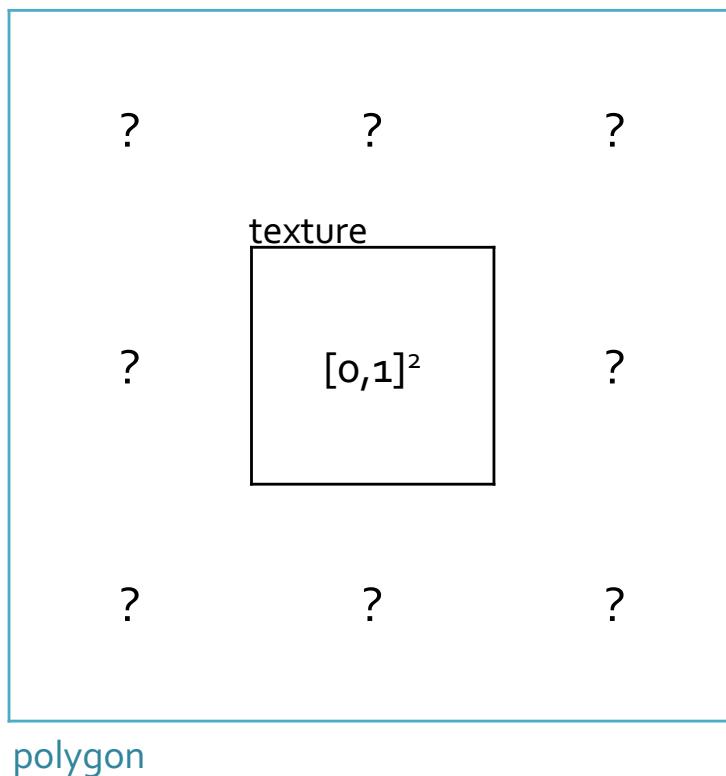
For RGB images:

`GL_MODULATE` — multiply together with surface color

`GL_BLEND` — use as a t value to blend surface color and a predetermined color

`GL_DECAL` and `GL_REPLACE` — use texture color directly

OpenGL Texture Repeat Mode



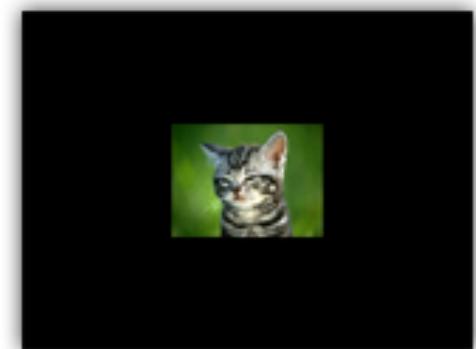
`GL_REPEAT`



`GL_MIRRORED_REPEAT`



`GL_CLAMP_TO_EDGE`

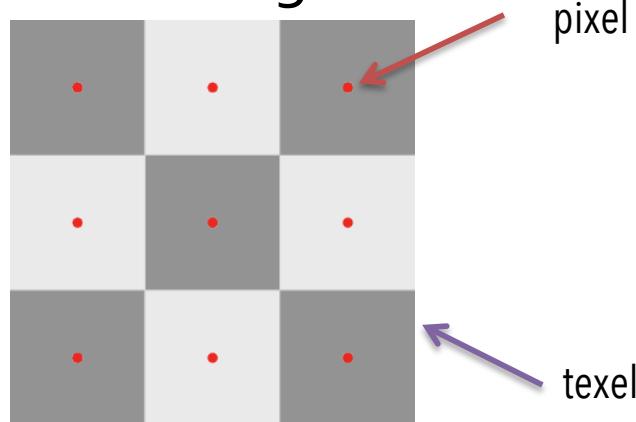


`GL_CLAMP_TO_BORDER`

<https://open.gl/textures>

Texture Sampling

A 256x256 texture on a square that occupies 256x256 pixels on screen looks almost like the original texture.



What if the square covers a screen area of ten times as many pixels as the texture (i.e., a texel covers multiple pixels)?

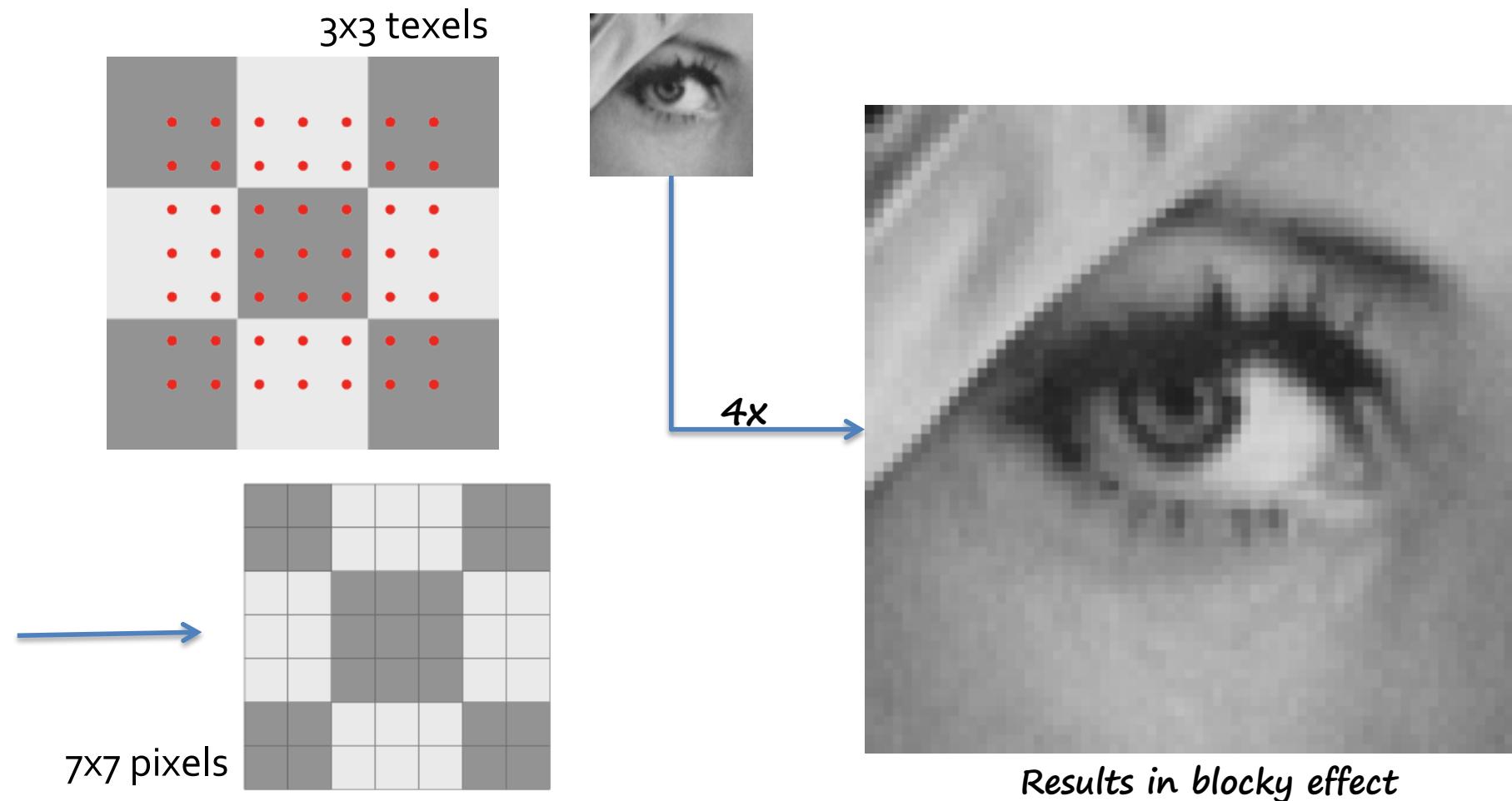
texture magnification

What if the square covers a screen area of only a few pixels (i.e., a pixel covers multiple texels)?

texture minification

Texture Magnification

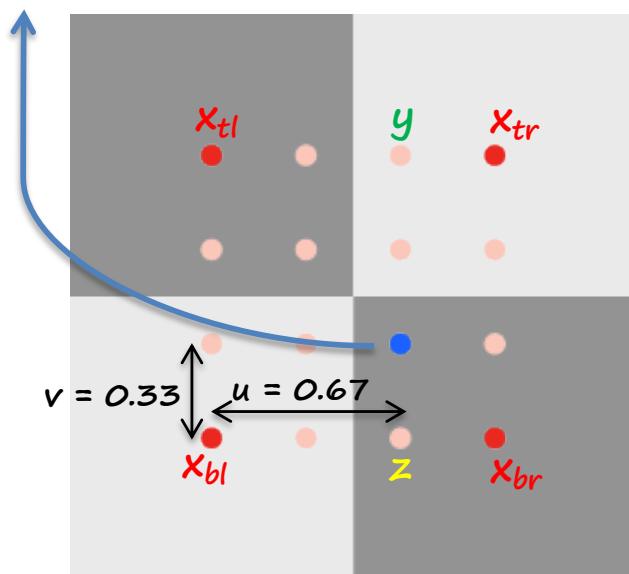
Nearest neighbor: select the nearest texel to each pixel



Texture Magnification

Bilinear interpolation: finds the four neighboring texels and performs linear interpolation in two dimensions

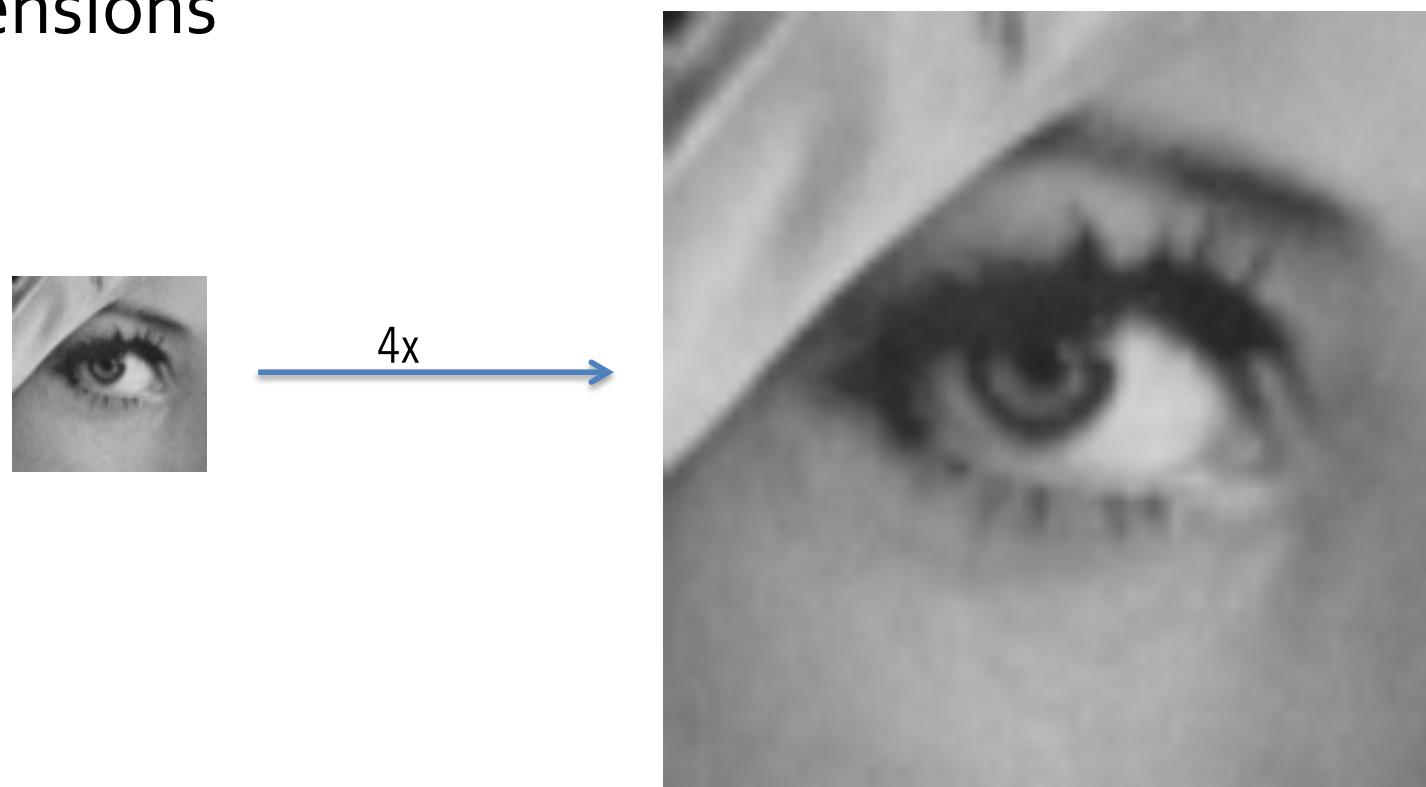
What should be the texture color for this pixel p ?



1. Determine the four neighbour texels:
 $x_{tl}, x_{tr}, x_{bl}, x_{br}$
2. Determine the interpolating parameter (u, v)
3. Let $t(x)$ be the color of a pixel x . Then we have
$$t(y) = (1-u) * t(x_{tl}) + u * t(x_{tr})$$
$$t(z) = (1-u) * t(x_{bl}) + u * t(x_{br})$$
and
$$t(p) = (1-v) * t(z) + v * t(y)$$

Texture Magnification

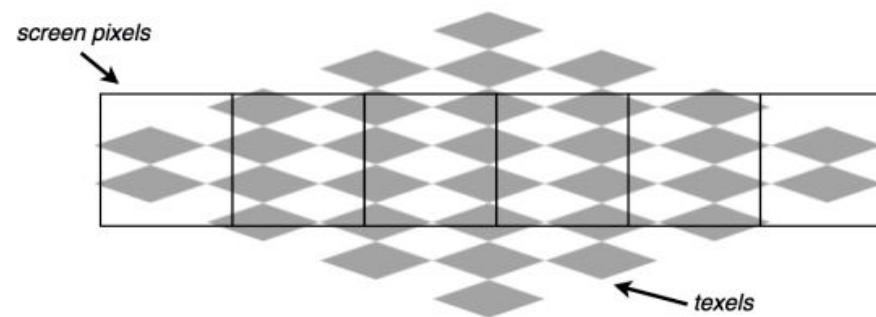
Bilinear interpolation: finds the four neighboring texels and performs linear interpolation in two dimensions



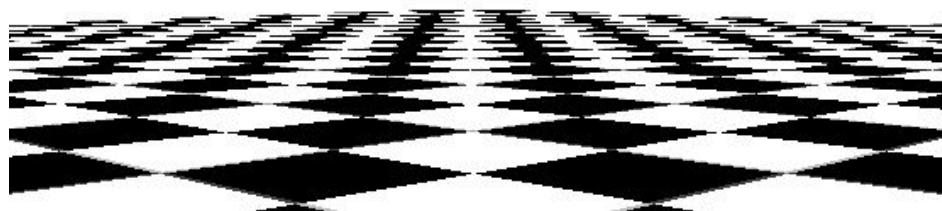
Bilinear interpolation results in a blurred image

Texture Minification

When a texture is minimized, a pixel is covered by several (fractional) texels, but we're only selecting one of them to use in the pixel



Using nearest neighbor sampling and bilinear interpolation leads to severe **aliasing** (i.e., jagged edges) problems.



nearest neighbor sampling



bilinear interpolation works only slightly better

Texturing with OpenGL

Here's an example setup

```
glEnable(GL_TEXTURE_2D);

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

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

This configures the texturing system to

- combine (modulate) the texture color with the surface color
- wrap texture coordinates around outside unit square
- linearly average texels when “magnifying” and “minifying”

Texturing with OpenGL

When drawing, just assign texture coordinates to vertices

```
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();
```

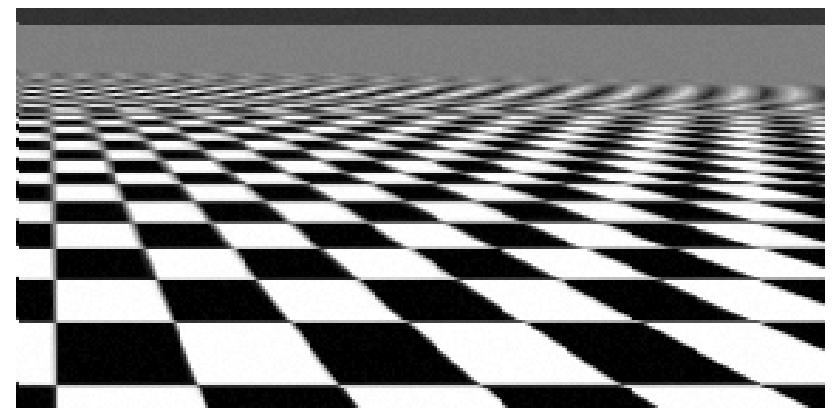
Texture Antialiasing

The antidote is to average (filter) all covered texels together

- need to choose appropriate averaging method

Removes objectionable artifacts

- but it's not magic
- very high frequency details just get smoothed over completely (e.g., gray on horizon)



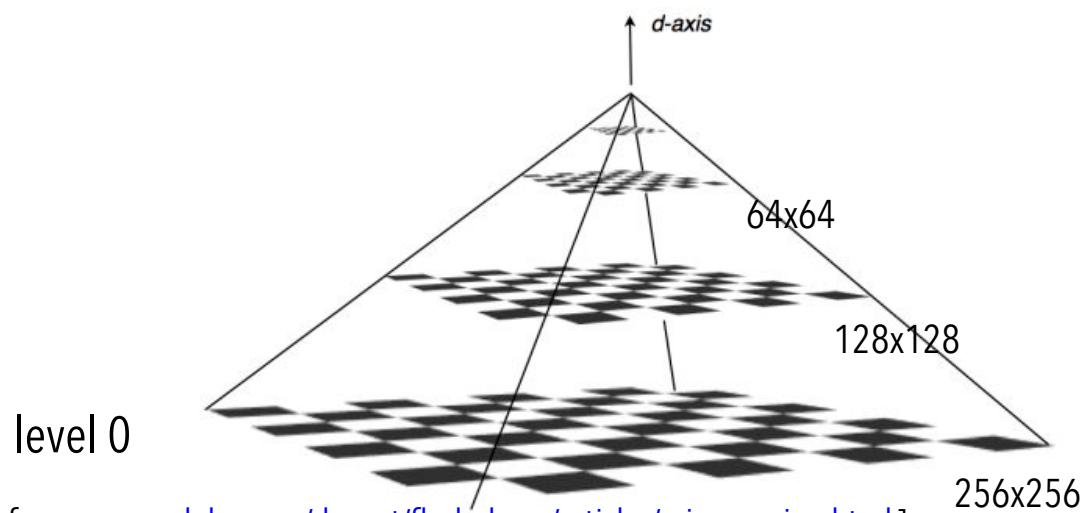
Unfortunately, there's a significant drawback here

- averaging covered texels can be very expensive
- for every pixel, might have to visit $O(n)$ texels
- this would really hurt rendering performance

Mipmaps (Image Pyramids)

An efficient method for **antialiasing** of textures.

- base of pyramid is the original image (level 0)
- level 1 is the image down-sampled by a factor of 2
- level 2 is down-sampled by a factor of 4, and so on
- requires that original dimensions be a power of 2
- total size = $4/3$ the size of the original image

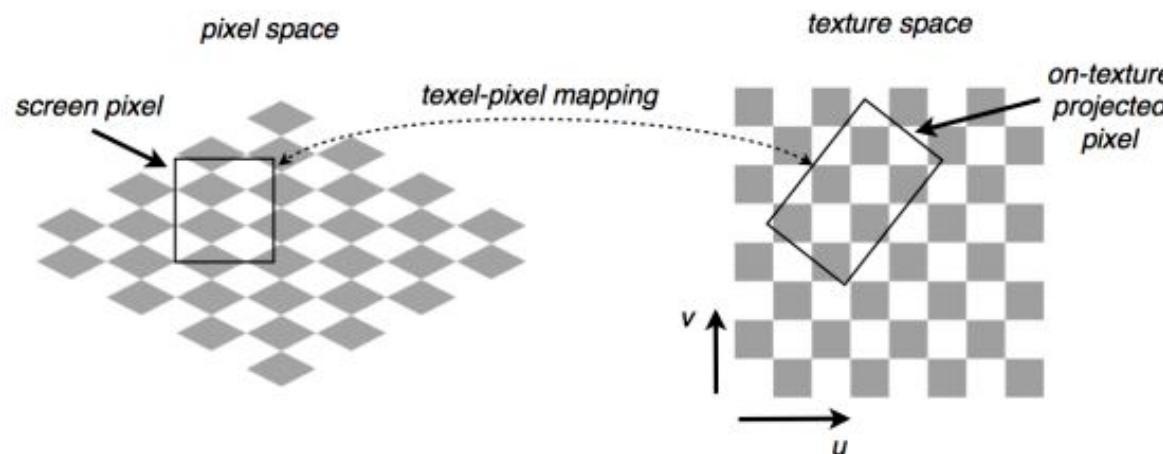


[image from: www.adobe.com/devnet/flashplayer/articles/mipmapping.html]

Mipmaps

While texturing, a screen pixel is mapped to the texture space and see how much area it covers.

If it is about the same size as a texel, the level 0 texture will be used. The larger the area it covers, the higher the texture level in the mipmap is used.



Antialiasing with Mipmaps

Mipmaps let us efficiently average large regions

- each texel in upper levels covers many base texels
 - at level k they are the average of $2^k \times 2^k$ texels
- can quickly assemble appropriate texels for averaging

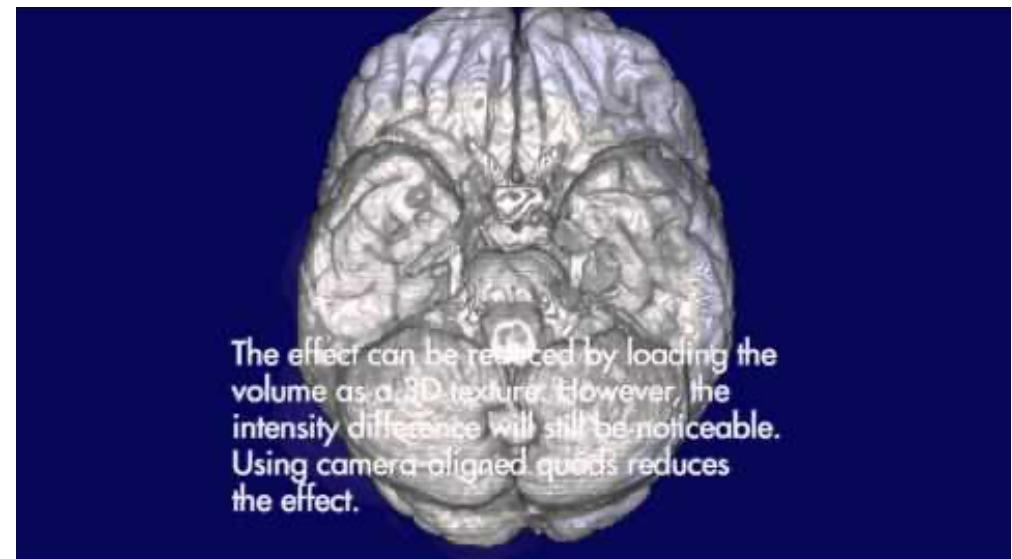
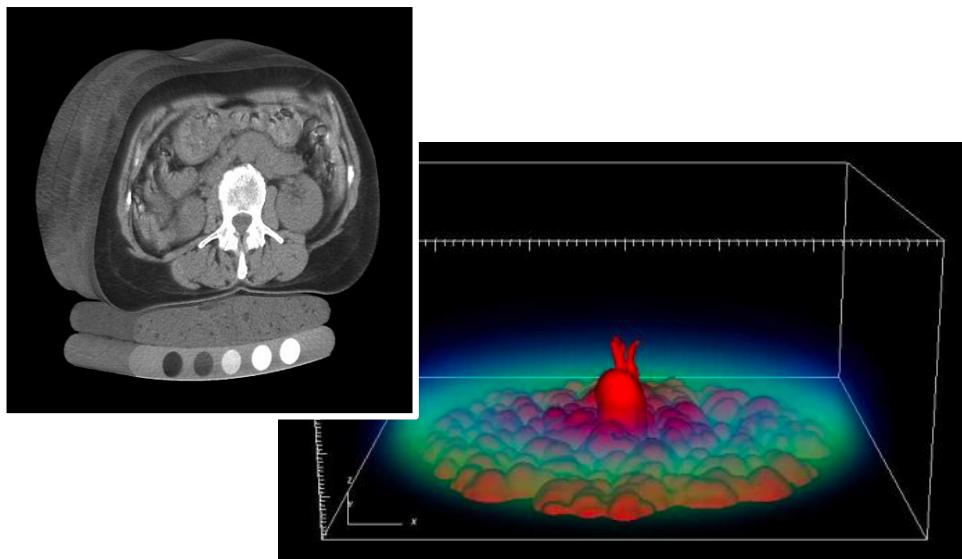
Fortunately, OpenGL can take care of most details

- `glBuild2DMipmaps()` — automatically generate pyramid from base image
- control behavior with `glTexParameter()`
- OpenGL handles all filtering details during rasterization

Solid Texture

Instead of texture images, we can define texture volumes

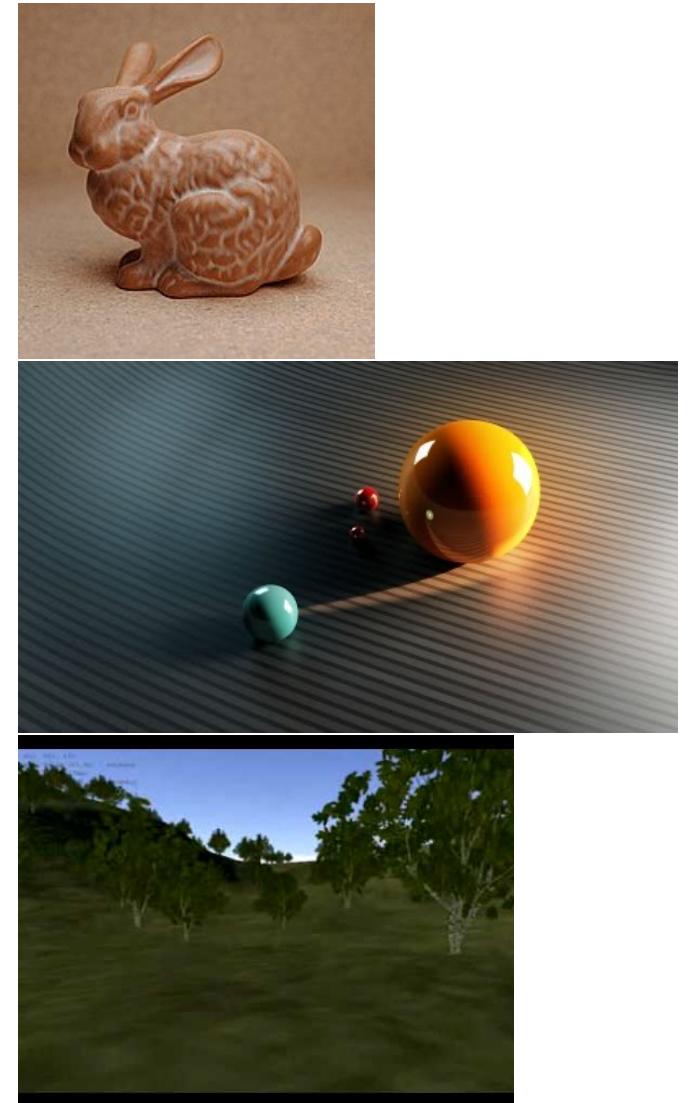
- create a 3-D parameterization (u,v,w) for the texture map this onto the object
- the easiest parameterization $(u,v,w) = (x,y,z)$



Some Texturing Applications

First, there's the obvious one: realistic surface detail

- paste a fur, marble, face scan, ... on a surface



We can also support illumination precalculation

- suppose we precompute some expensive lighting effects
 - soft shadows, indirect light (e.g., radiosity)
- can hard code this lighting into texture maps

Texturing can also be handy for faking objects

- **billboards** — place image on a polygon which always rotates to face the viewer (handy for things like trees)
- sprites used in video games are a similar idea

And texturing is useful for level of detail management

- can decouple resolution of texture from resolution of surface

<https://youtu.be/PeogP2uIEZU>
<https://youtu.be/di16x6ovABY>