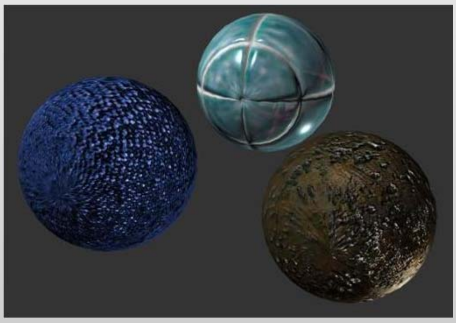
Computer Graphics



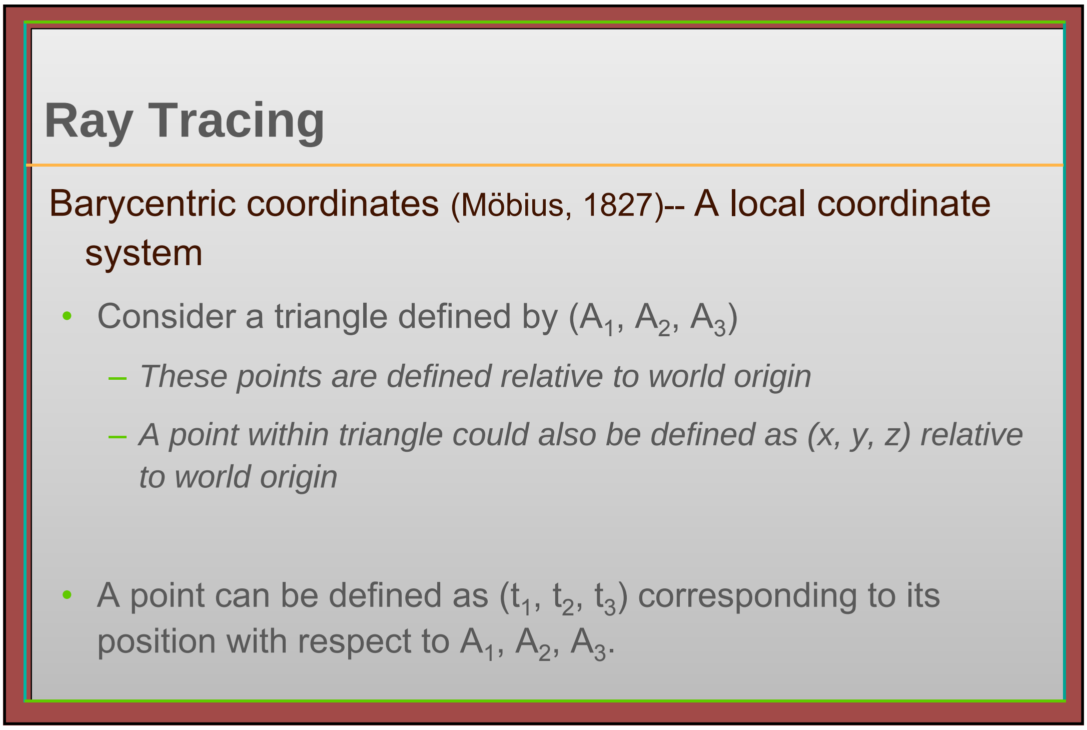
Texture Maps, Shading, Anti-aliasing

Instructor: Dr. MAO Aihua

ahmao@scut.edu.cn

Question 1: Barycentric coordinates

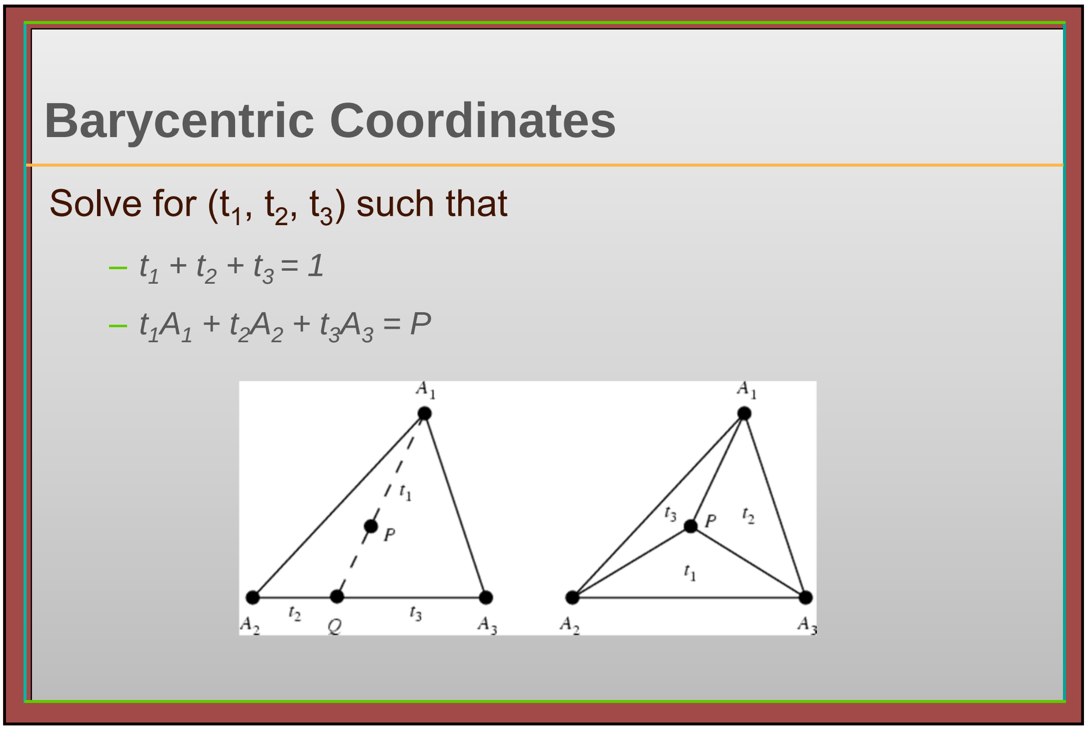
- What is Barycentric coordinates? Why we use it?
- How to calculate Barycentric coordinates?
- Read for more information from the reference: “Blended barycentric coordinates”



Ray Tracing

Barycentric coordinates (Möbius, 1827)-- A local coordinate system

- Consider a triangle defined by (A_1, A_2, A_3)
  - *These points are defined relative to world origin*
  - *A point within triangle could also be defined as (x, y, z) relative to world origin*
- A point can be defined as (t_1, t_2, t_3) corresponding to its position with respect to A_1, A_2, A_3.



Barycentric Coordinates

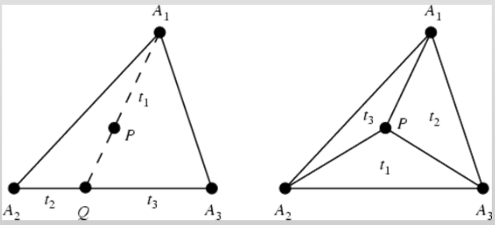
Solve for (t_1, t_2, t_3) such that

- $t_1 + t_2 + t_3 = 1$
- $t_1A_1 + t_2A_2 + t_3A_3 = P$

Barycentric Coordinates

An observation

- t_1, t_2, and t_3 are weights such that when they are used to represent the mass at the vertices of the triangle, P is at its center of mass



- t_1, t_2, and t_3 are weights that represent the ratio of the area of each of the three subtriangles to the area of the whole

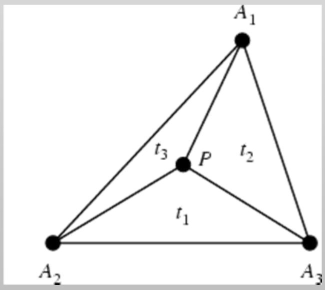
Barycentric Coordinates

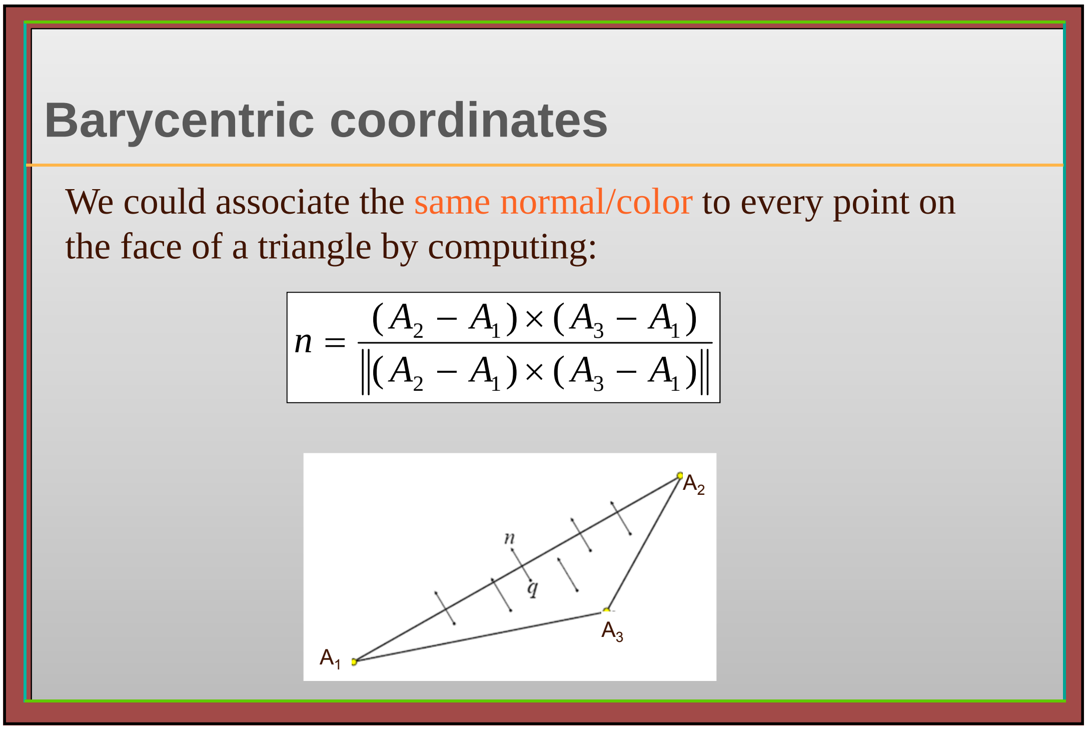
The cross product computes a triangle's area

- $\| (A_2 - A_1) \times (A_3 - A_1) \| = (\text{area of } A_1A_2A_3)*2$

Where || x || = the area of the triangle x… the cross product

$$t_1 = \frac{\|(A_2 - p) \times (A_3 - p)\|}{\|(A_2 - A_1) \times (A_3 - A_1)\|}$$

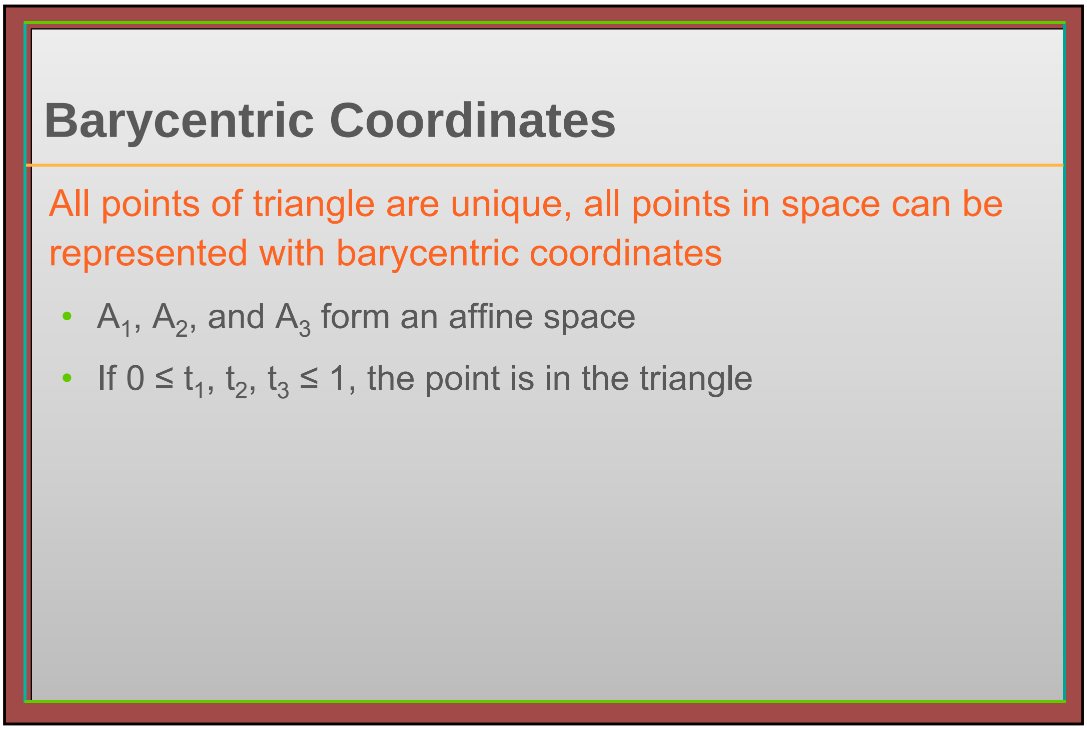




Barycentric coordinates

We could associate the same normal/color to every point on the face of a triangle by computing:

$$n = \frac{(A_2 - A_1) \times (A_3 - A_1)}{\left\| (A_2 - A_1) \times (A_3 - A_1) \right\|}$$



Barycentric Coordinates

All points of triangle are unique, all points in space can be represented with barycentric coordinates

- A_1, A_2, and A_3 form an affine space
- If $0 \leq t_1, t_2, t_3 \leq 1$, the point is in the triangle

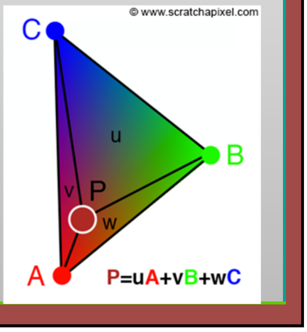
Barycentric Coordinates

Barycentric coordinates are most useful in shading

we can associate any additional information or data (points, color, vectors, etc.) to each one of its vertices in the triangle

Let vertex A =red, vertex B=green and vertex C=blue

The barycentric coordinates are used to compute a point inside of the triangle using the triangle vertices A,B,C, Then we can interpolate the color data of the point by those defined at A,B,C



Question 2: Shading models

- What is shading? And what are the main shading models?
- What are the main difference between them?

Applying Illumination

We have an illumination model for a point on a surface.

Need to shade a smooth surface

Often times, a smooth surface is approximated by polygon patches, which points of polygon should we use for illumination?

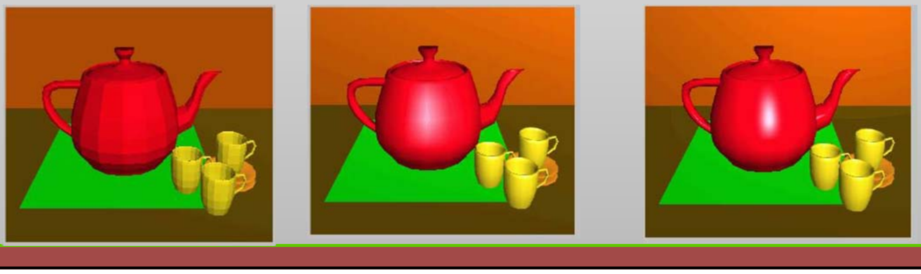
Keep in mind:

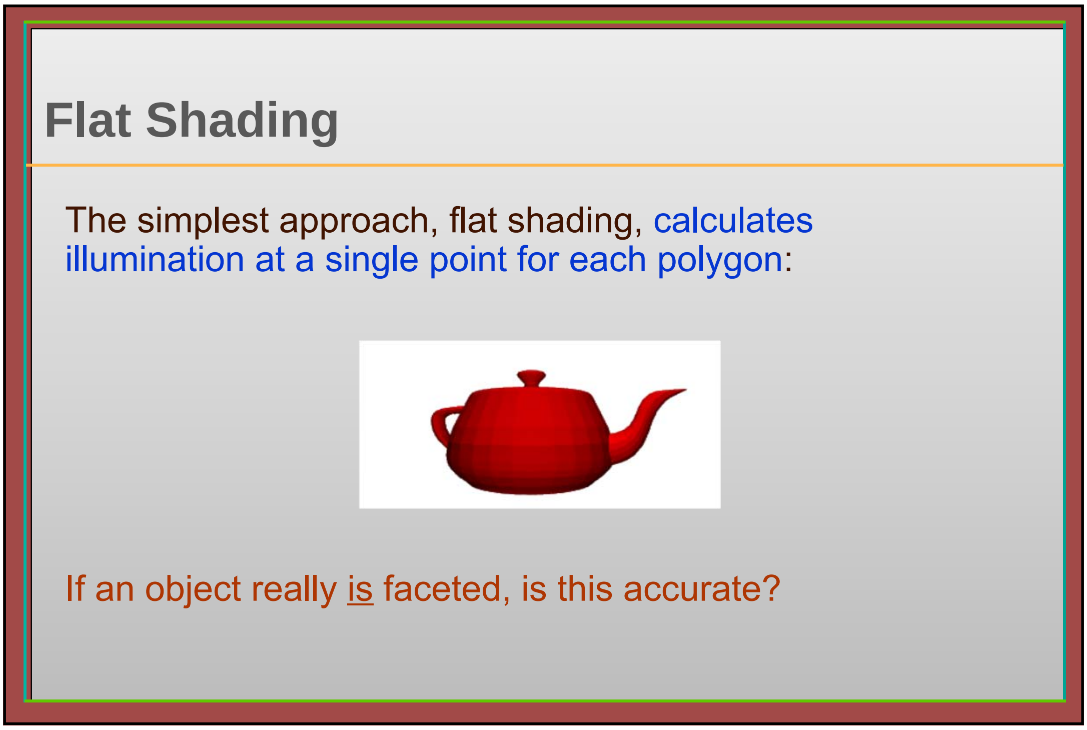
- It's a fairly expensive calculation
- Several possible answers, each with different implications for the visual quality of the result

Applying Illumination

In computer graphics, many scenes are constructed with polygonal/triangular models:

- Three common shading (illumination rendering):
- Flat Shading/Gouraud Shading/Phong Shading

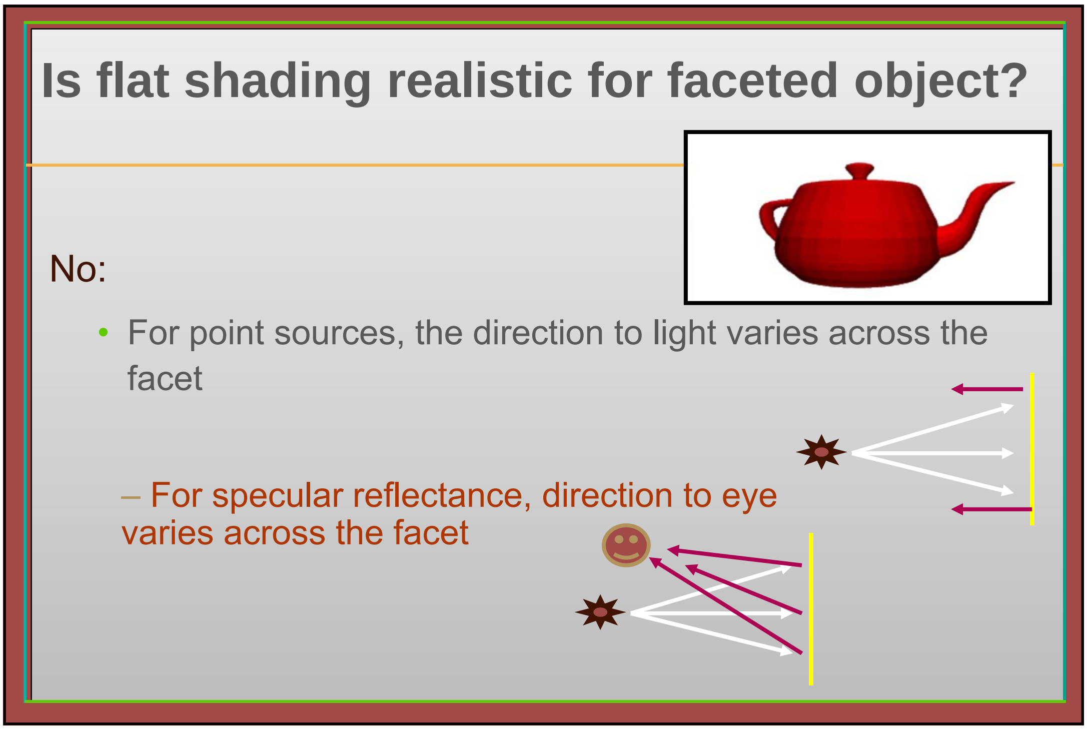




Flat Shading

The simplest approach, flat shading, calculates illumination at a single point for each polygon:

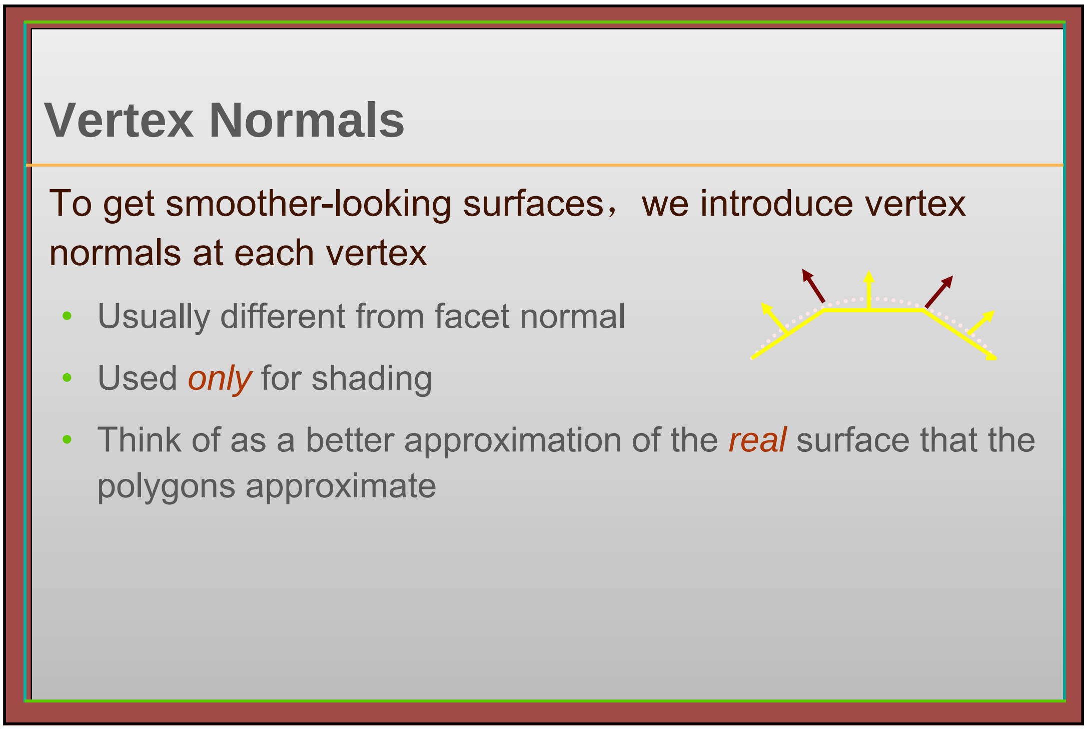
If an object really is faceted, is this accurate?



Is flat shading realistic for faceted object?

No:

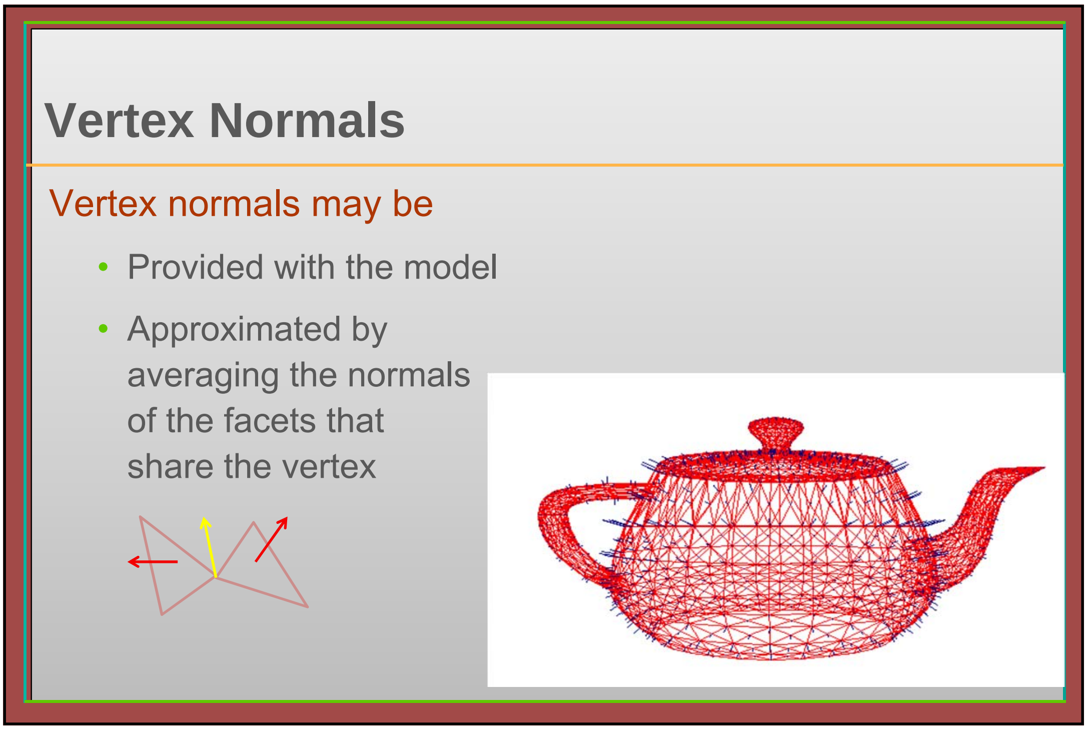
- For point sources, the direction to light varies across the facet
  - For specular reflectance, direction to eye varies across the facet



Vertex Normals

To get smoother-looking surfaces， we introduce vertex normals at each vertex

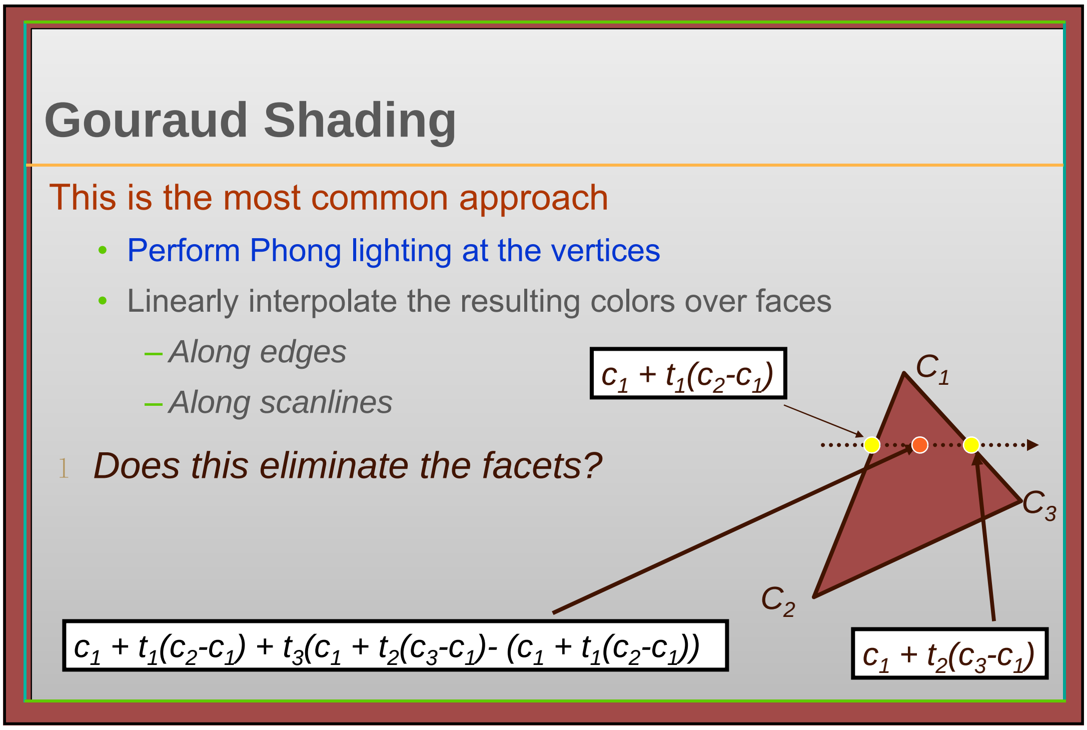
- Usually different from facet normal
- Used *only* for shading
- Think of as a better approximation of the *real* surface that the polygons approximate



Vertex Normals

Vertex normals may be

- Provided with the model
- Approximated by averaging the normals of the facets that share the vertex



Gouraud Shading

This is the most common approach

- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - *Along edges*
  - *Along scanlines*

l *Does this eliminate the facets?*

$c_1 + t_1(c_2-c_1)$

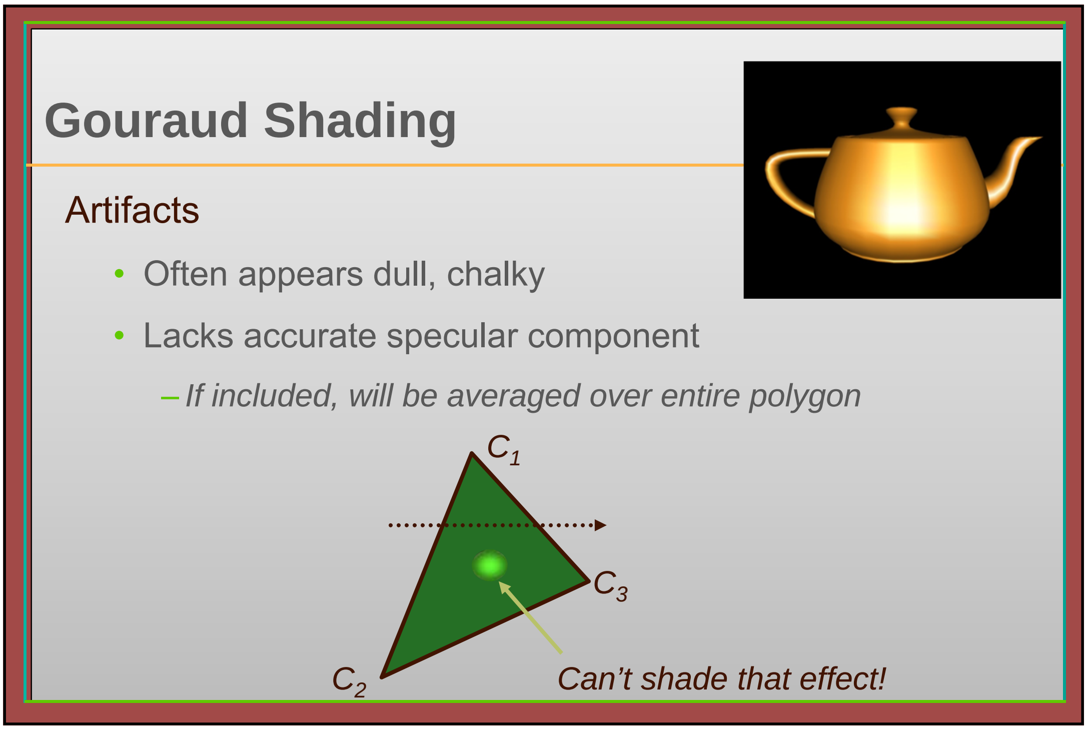
C_1

C_3

C_2

$c_1 + t_1(c_2-c_1) + t_3(c_1 + t_2(c_3-c_1)- (c_1 + t_1(c_2-c_1))$

$c_1 + t_2(c_3-c_1)$



Gouraud Shading

Artifacts

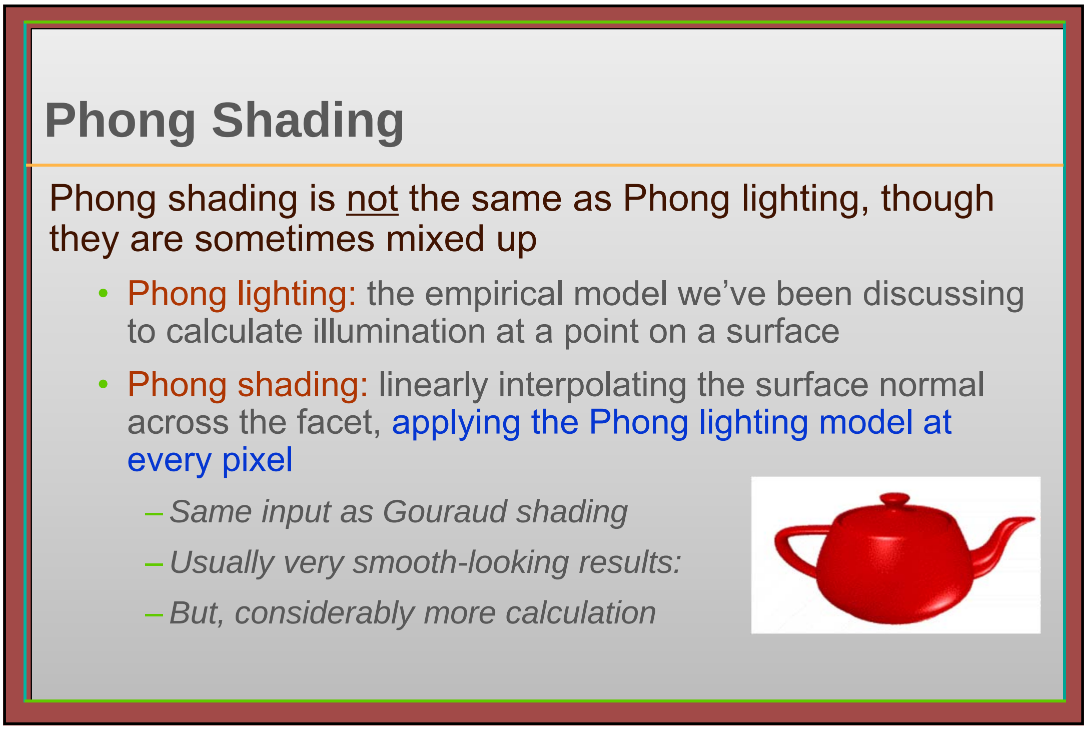
- Often appears dull, chalky
- Lacks accurate specular component
  - *If included, will be averaged over entire polygon*

C_1

C_3

C_2

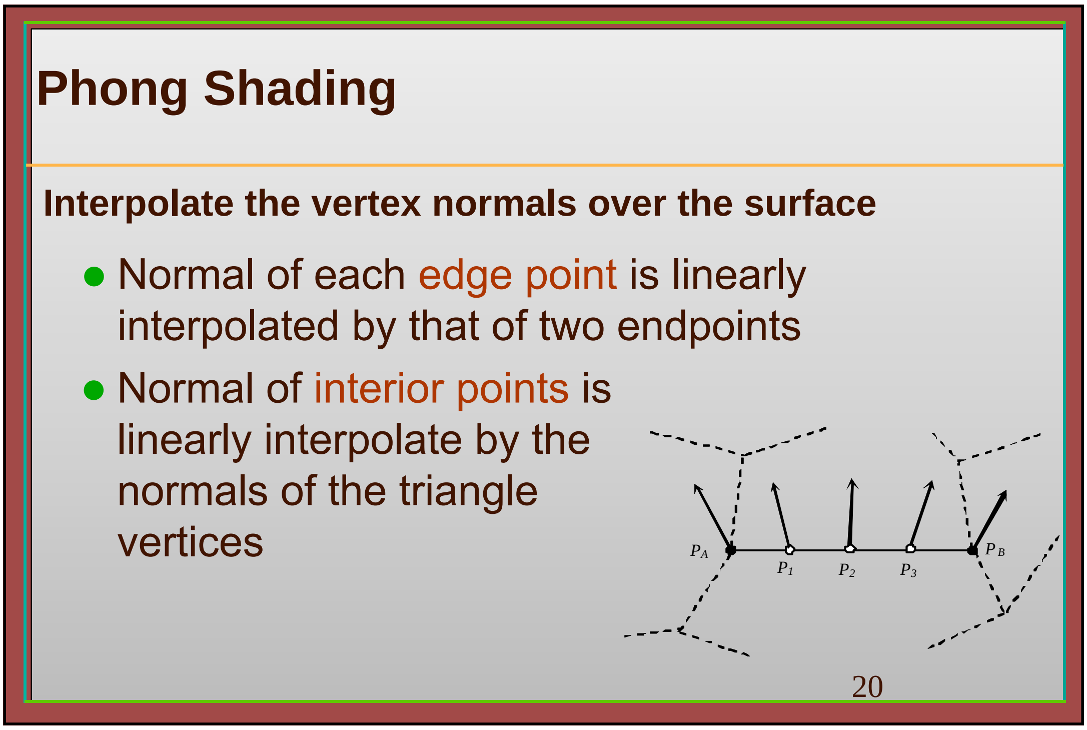
Can't shade that effect!



Phong Shading

Phong shading is not the same as Phong lighting, though they are sometimes mixed up

- Phong lighting: the empirical model we've been discussing to calculate illumination at a point on a surface
- Phong shading: linearly interpolating the surface normal across the facet, applying the Phong lighting model at every pixel
  - *Same input as Gouraud shading*
  - *Usually very smooth-looking results:*
  - *But, considerably more calculation*



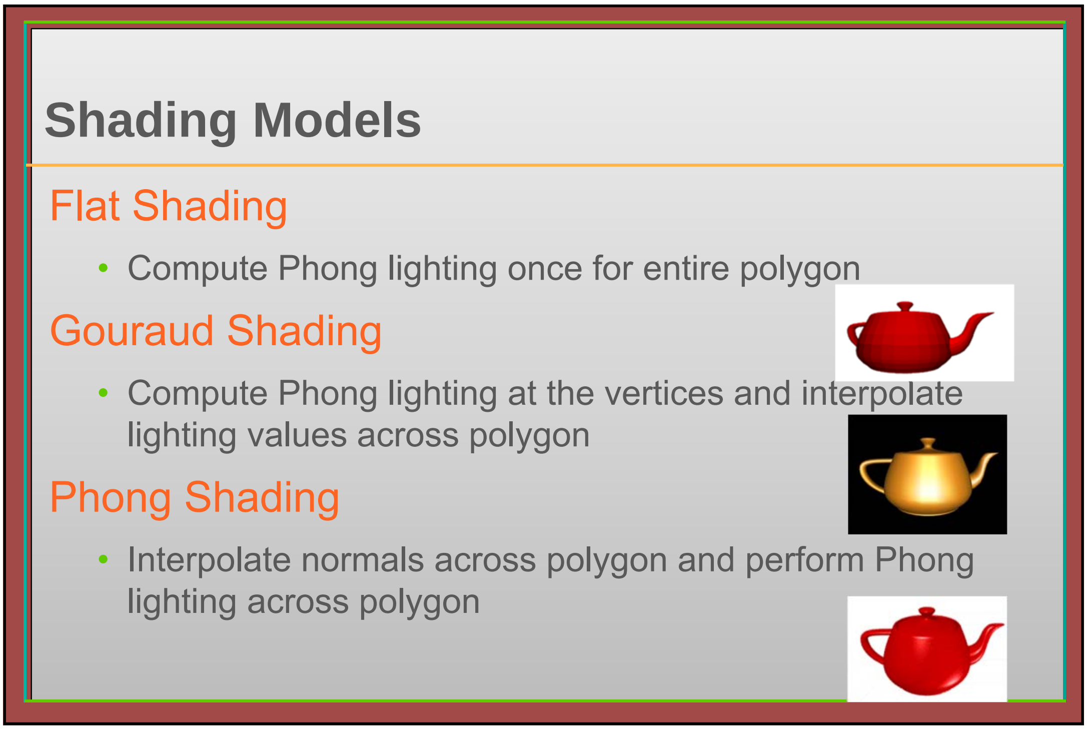
Phong Shading

Interpolate the vertex normals over the surface

- Normal of each edge point is linearly interpolated by that of two endpoints
- Normal of interior points is linearly interpolate by the normals of the triangle vertices

P_A
P_1
P_2
P_3
P_B

20



Shading Models

Flat Shading

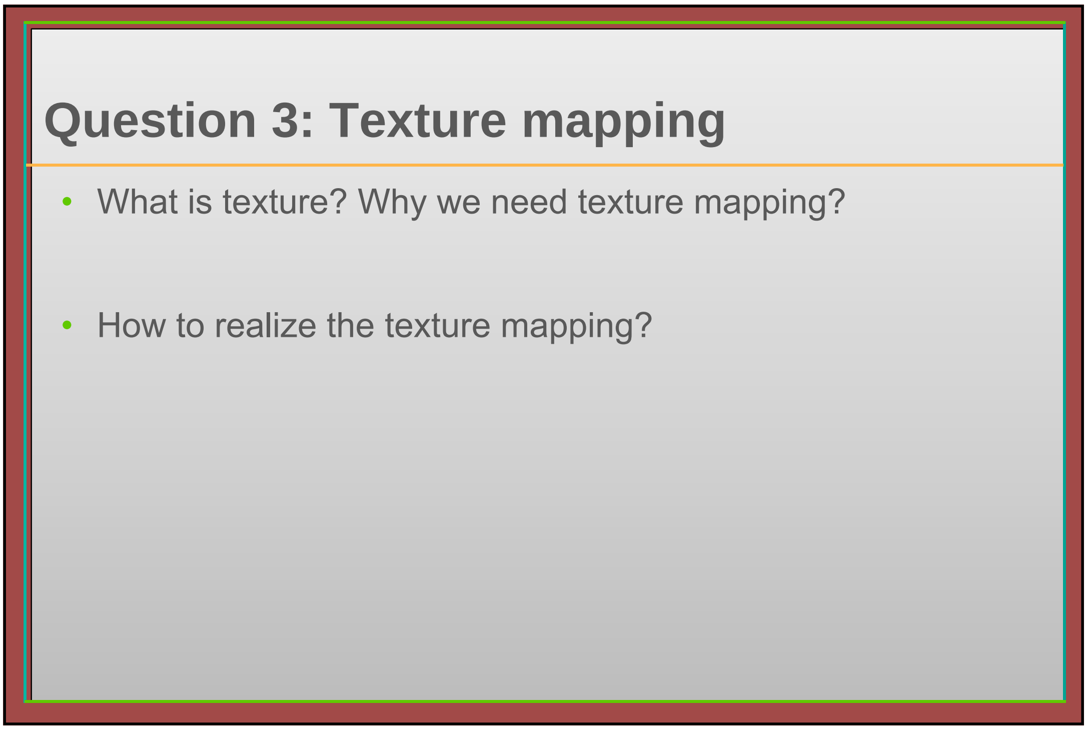
- Compute Phong lighting once for entire polygon

Gouraud Shading

- Compute Phong lighting at the vertices and interpolate lighting values across polygon

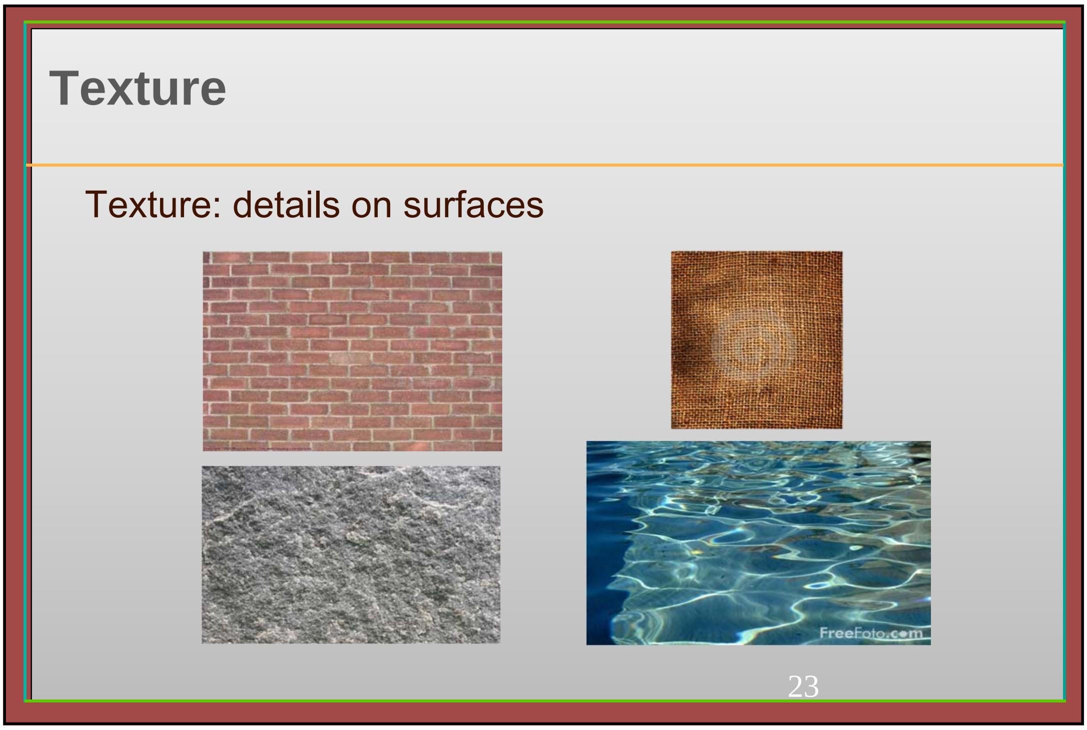
Phong Shading

- Interpolate normals across polygon and perform Phong lighting across polygon



Question 3: Texture mapping

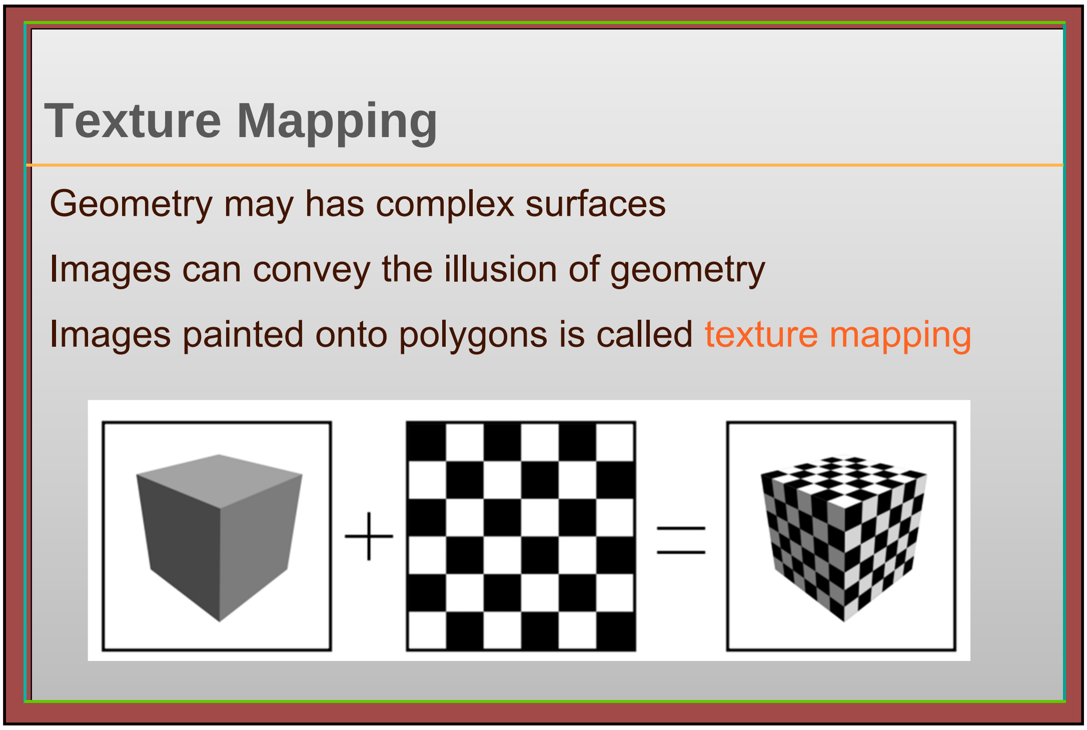
- What is texture? Why we need texture mapping?
- How to realize the texture mapping?



Texture

Texture: details on surfaces

FreeFoto.com



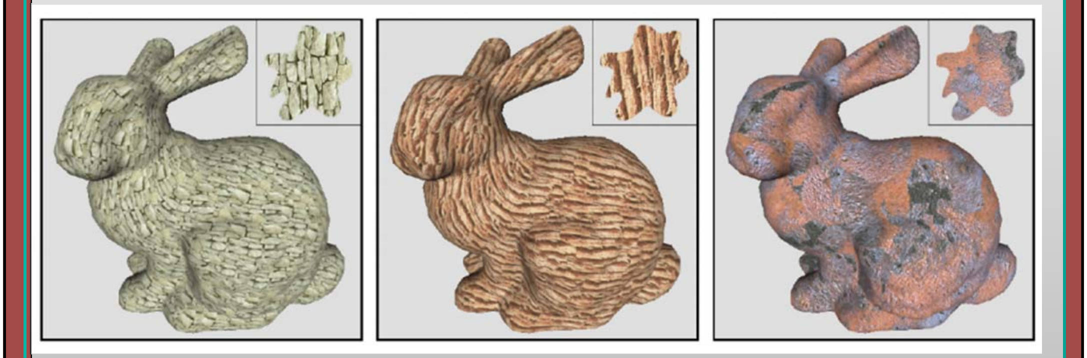
Texture Mapping

Geometry may has complex surfaces

Images can convey the illusion of geometry

Images painted onto polygons is called texture mapping

Texture Mapping

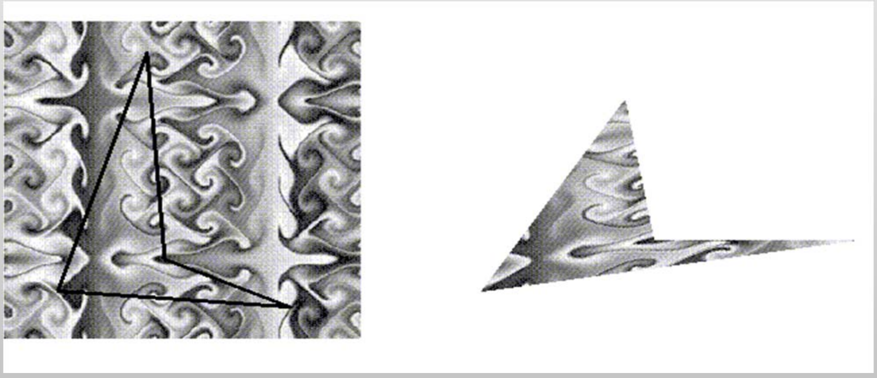


Texture map

Images applied to polygons to enhance the visual effect of a scene

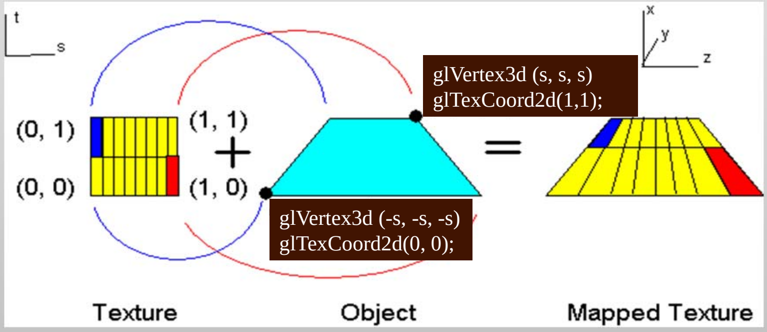
- Is rectangular arrays of data
  - *Color, luminance, alpha*
  - *Components of array called texels*
  - *In 3D, volumetric voxels*

Example Texture Map



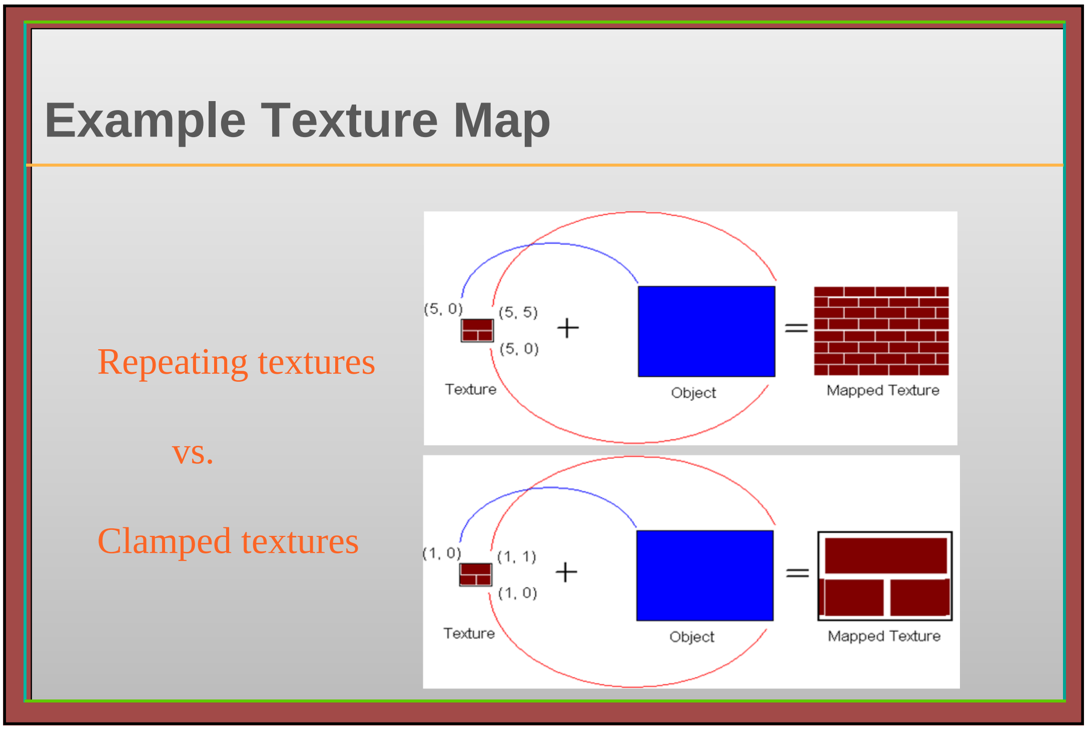
Applied to tilted polygon

Example Texture Map



glVertex3d (s, s, s)
glTexCoord2d(1,1);

glVertex3d (-s, -s, -s)
glTexCoord2d(0, 0);

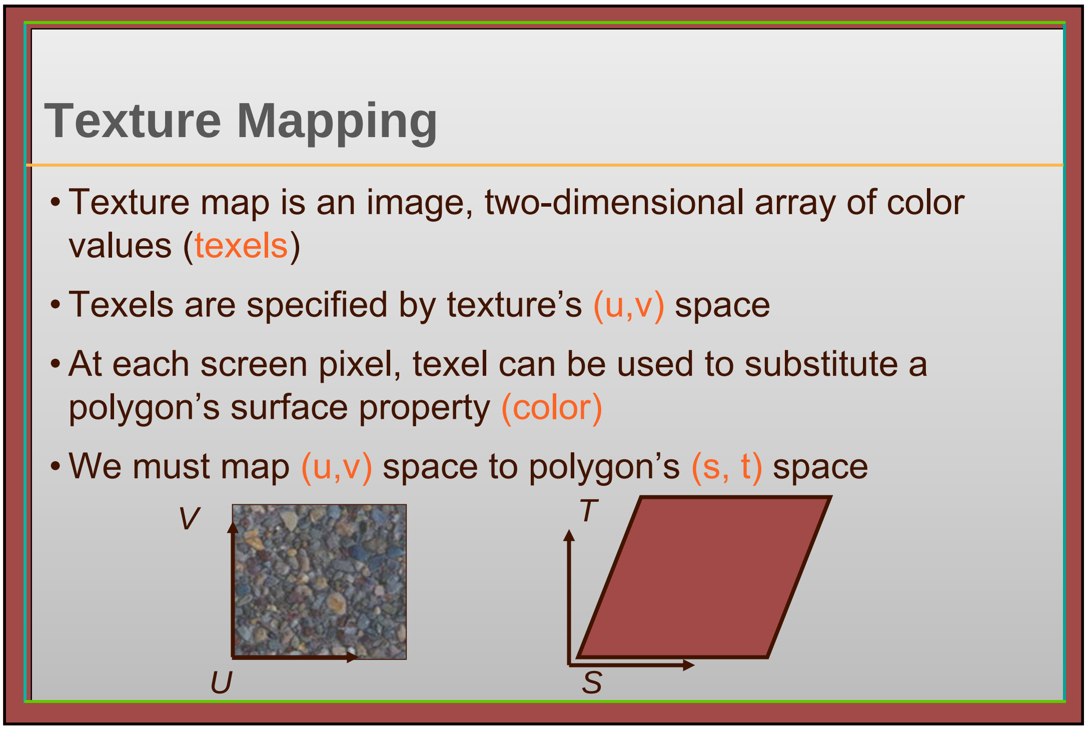


Example Texture Map

Repeating textures

vs.

Clamped textures



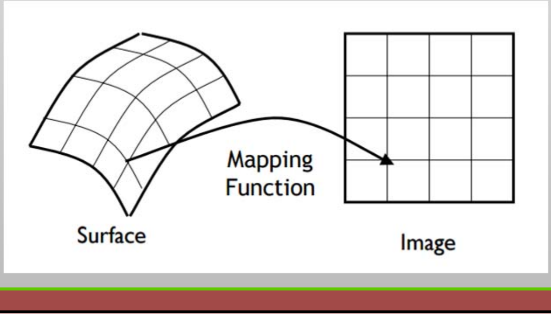
Texture Mapping

- Texture map is an image, two-dimensional array of color values (texels)
- Texels are specified by texture's (u,v) space
- At each screen pixel, texel can be used to substitute a polygon's surface property (color)
- We must map (u,v) space to polygon's (s, t) space

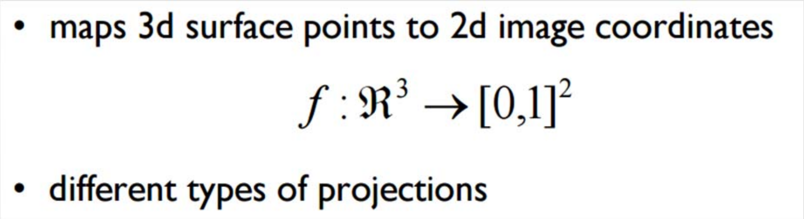
Texture Mapping

(u,v) to (s,t) mapping can be explicitly set at vertices by storing texture coordinates with each vertex

Compute (u,v) to (s,t), mapping for pixels in the image to surface on the polygon by mapping functions



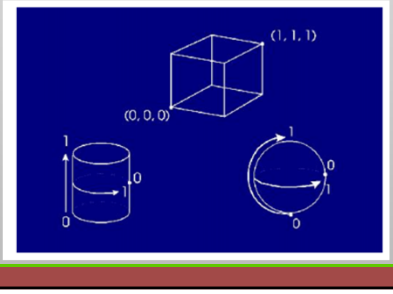
Mapping Function-Projections



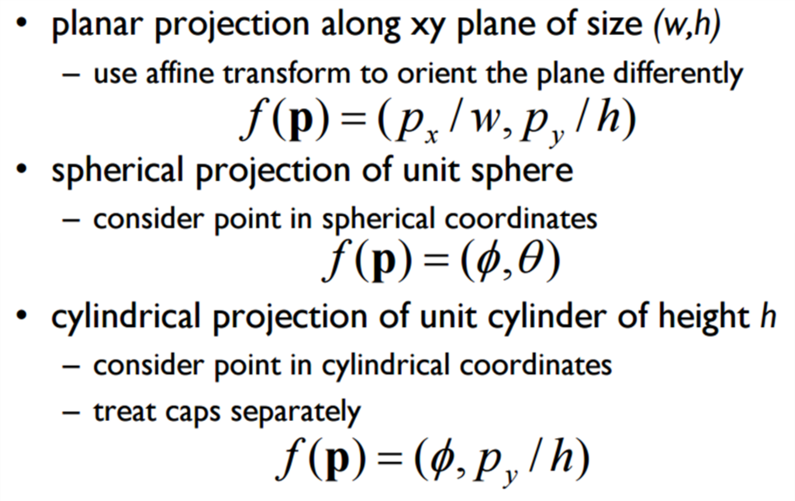
- maps 3d surface points to 2d image coordinates

$$f : \Re^3 \rightarrow [0,1]^2$$

- different types of projections



Projections



- planar projection along xy plane of size *(w,h)*
  - use affine transform to orient the plane differently

$$f(\mathbf{p}) = (p_x / w, p_y / h)$$

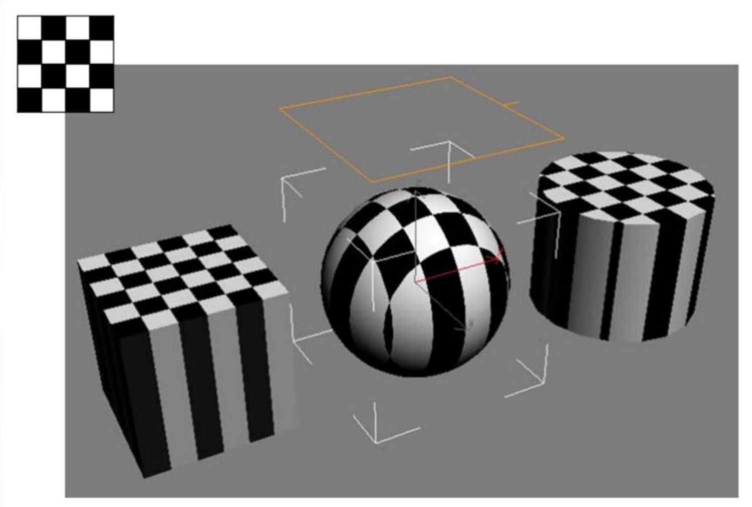
- spherical projection of unit sphere
  - consider point in spherical coordinates

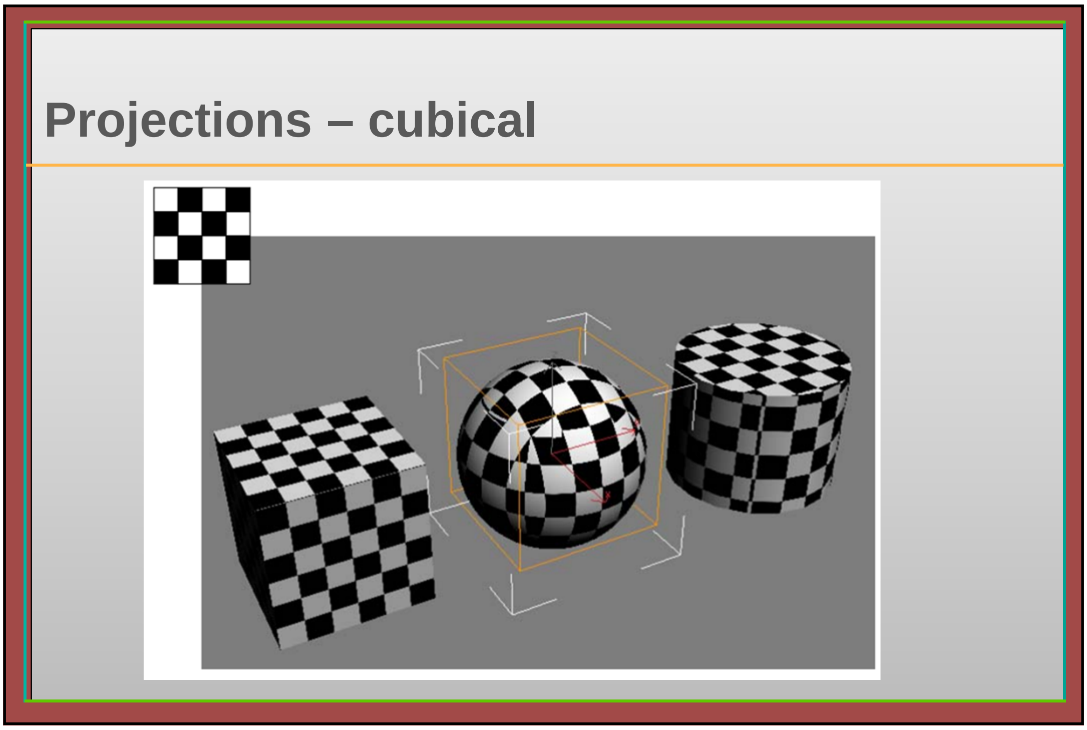
$$f(\mathbf{p}) = (\phi, \theta)$$

- cylindrical projection of unit cylinder of height *h*
  - consider point in cylindrical coordinates
  - treat caps separately

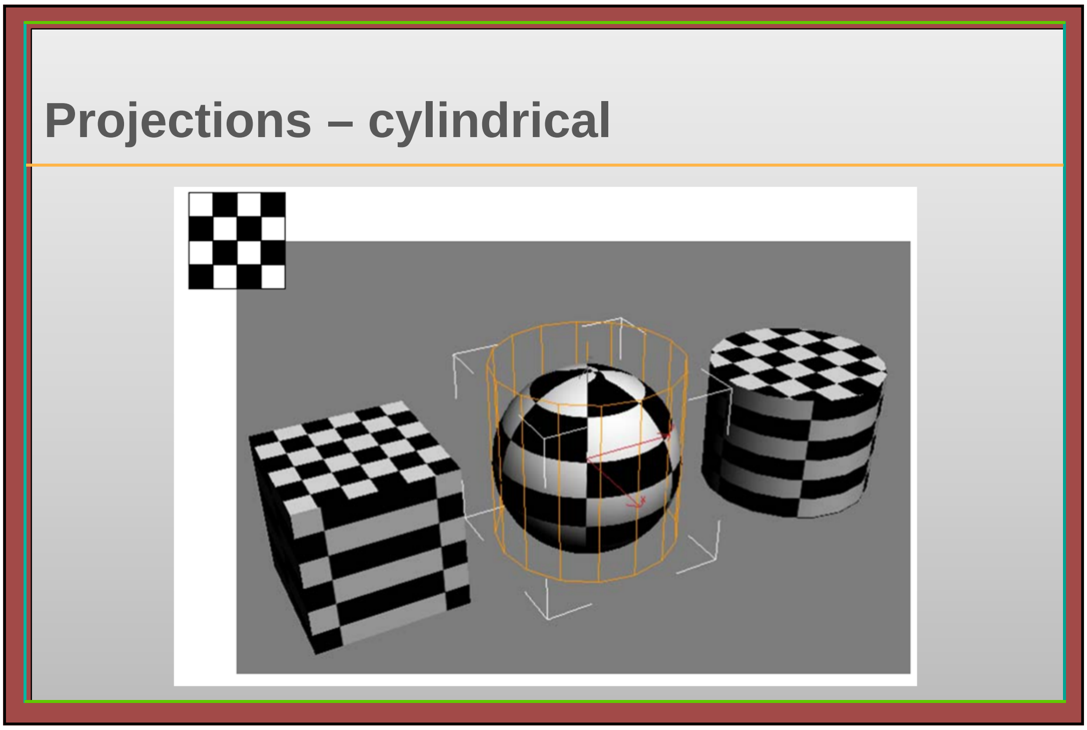
$$f(\mathbf{p}) = (\phi, p_y / h)$$

Projections – planar





Projections – cubical



Projections – cylindrical

Texture Coordinates

Every polygon can have object coordinates and texture coordinates

- Object coordinates describe where polygon vertices are on the screen
- Texture coordinates describe texel coordinates of each vertex
- Texture coordinates are interpolated along vertex-vertex

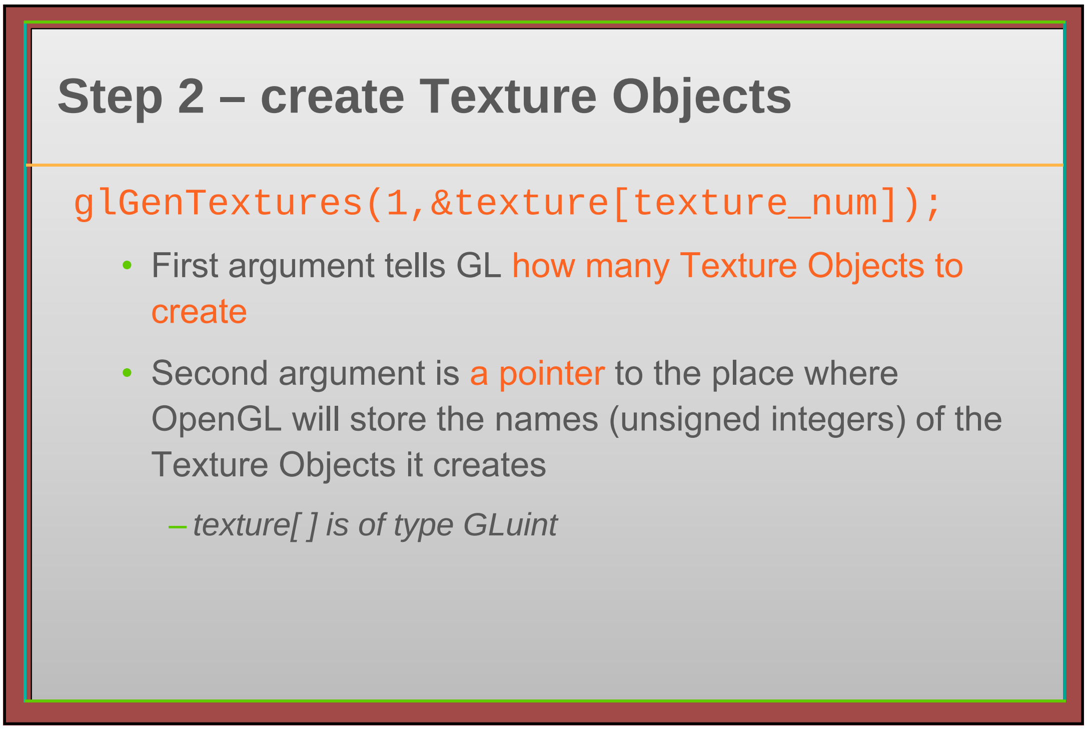
glTexCoord{1234}{sifd}(TYPE coords)

Why 1→4 coords? s, t, r, and q (for homogeneous coordinates)

Example use of Texture

Read .bmp from file

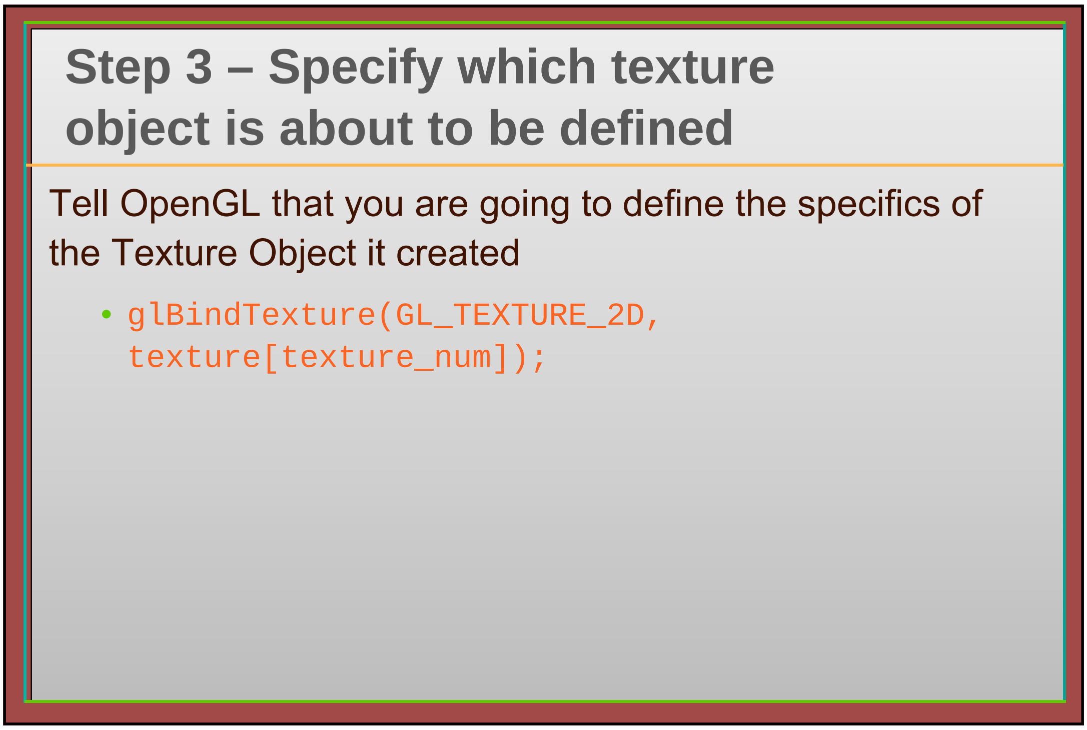
- Use Image data type
  - `getc()` and `fseek()` *to read image x & y size*
  - `fread()` *fills the Image→data memory with actual red/green/blue values from .bmp*
- Note
  - `malloc()` *Image->data to appropriate size*
  - *.bmp stores color in bgr order and we convert to rgb order*



Step 2 – create Texture Objects

```
glGenTextures(1,&texture[texture_num]);
```

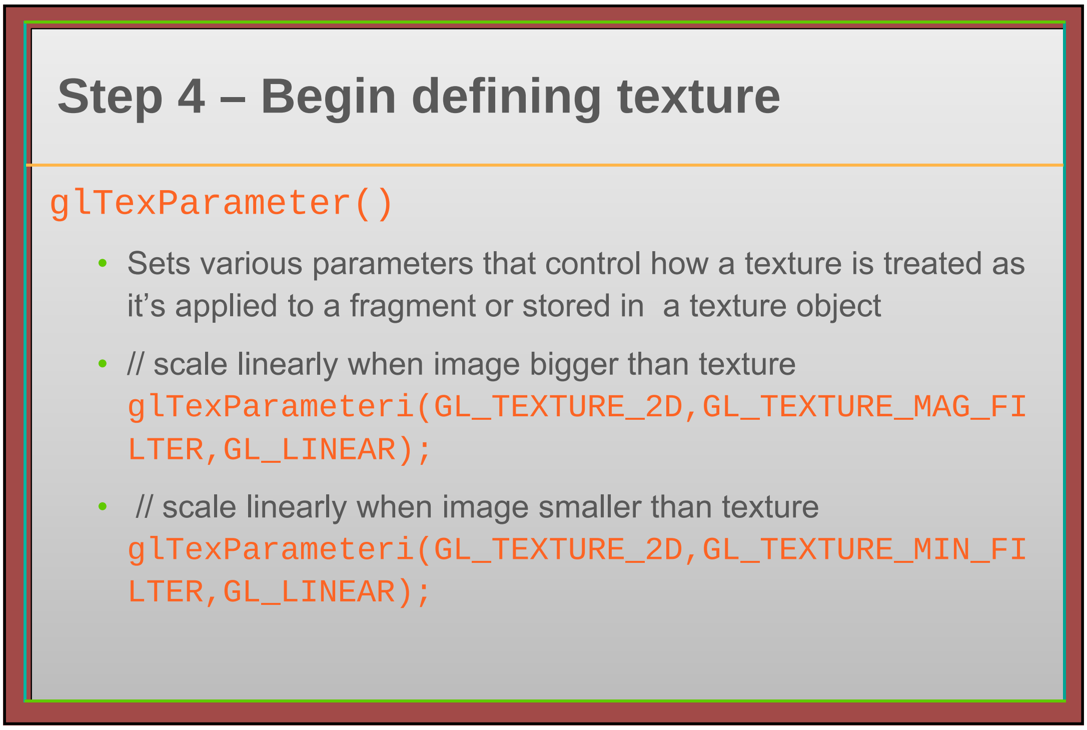
- First argument tells GL how many Texture Objects to create
- Second argument is a pointer to the place where OpenGL will store the names (unsigned integers) of the Texture Objects it creates
  - *texture[ ] is of type GLuint*



Step 3 – Specify which texture object is about to be defined

Tell OpenGL that you are going to define the specifics of the Texture Object it created

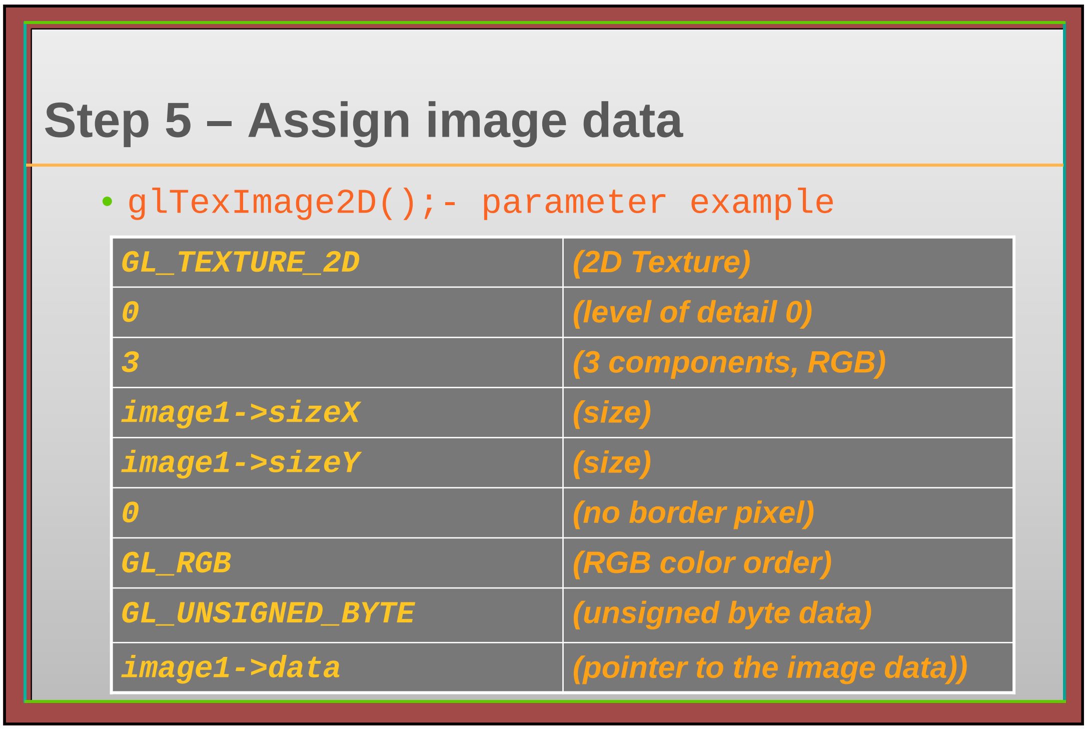
- `glBindTexture(GL_TEXTURE_2D, texture[texture_num]);`



Step 4 – Begin defining texture

`glTexParameter()`

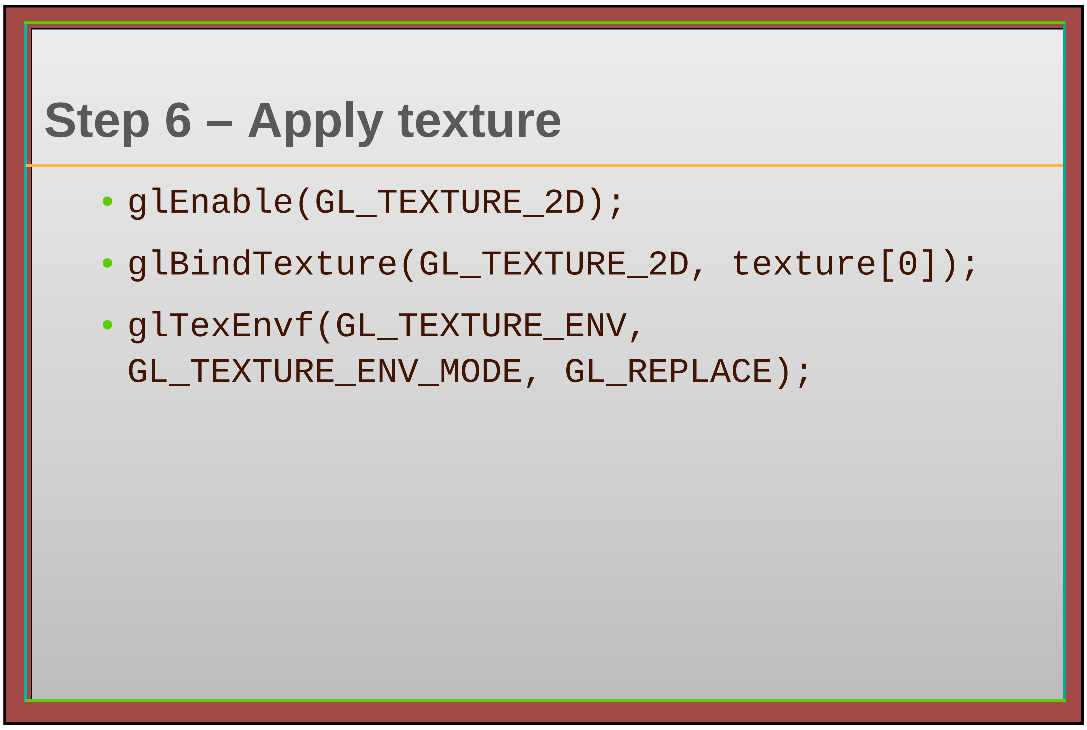
- Sets various parameters that control how a texture is treated as it’s applied to a fragment or stored in a texture object
- // scale linearly when image bigger than texture `glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);`
- // scale linearly when image smaller than texture `glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);`



Step 5 – Assign image data

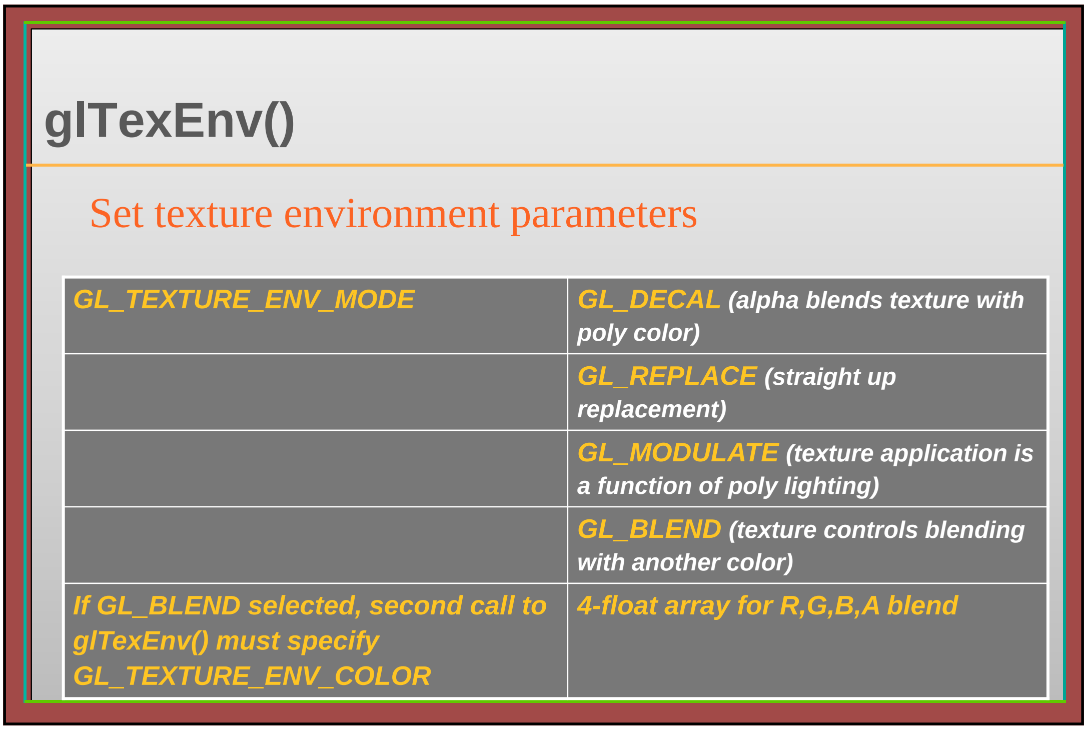
- `glTexImage2D();- parameter example`

GL_TEXTURE_2D	*(2D Texture)*
0	*(level of detail 0)*
3	*(3 components, RGB)*
image1->sizeX	*(size)*
image1->sizeY	*(size)*
0	*(no border pixel)*
GL_RGB	*(RGB color order)*
GL_UNSIGNED_BYTE	*(unsigned byte data)*
image1->data	*(pointer to the image data))*



Step 6 – Apply texture

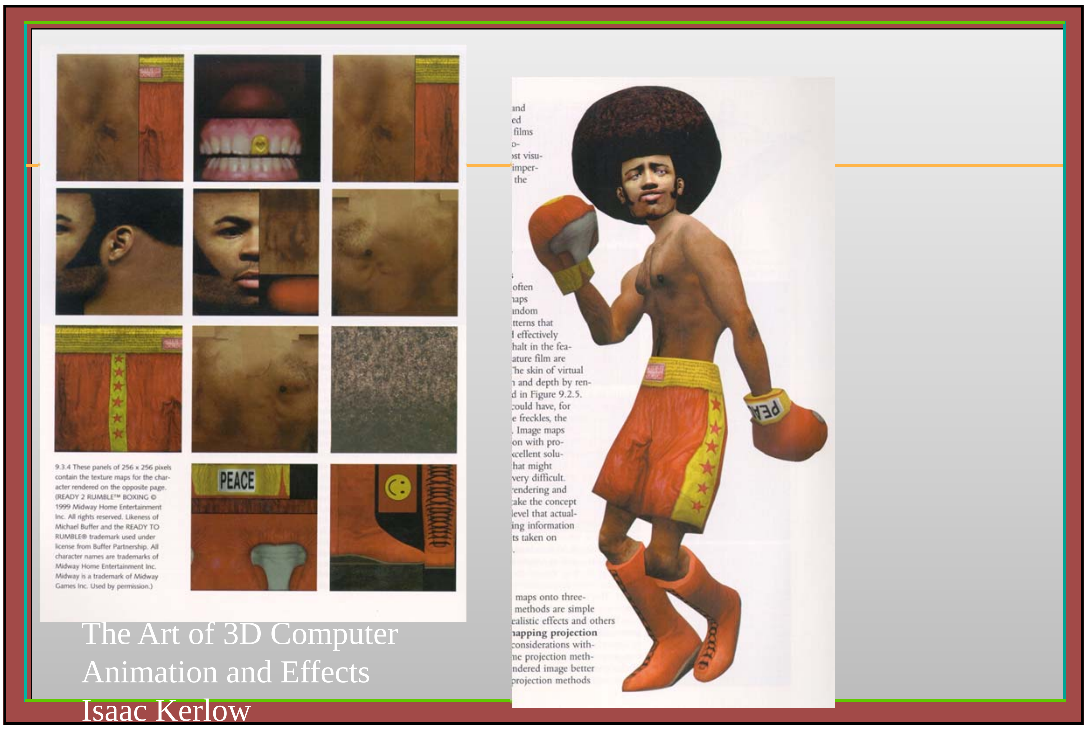
- `glEnable(GL_TEXTURE_2D);`
- `glBindTexture(GL_TEXTURE_2D, texture[0]);`
- `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`



glTexEnv()

Set texture environment parameters

GL_TEXTURE_ENV_MODE	***GL_DECAL*** ***(alpha blends texture with poly color)***
	GL_REPLACE ***(straight up replacement)***
	GL_MODULATE ***(texture application is a function of poly lighting)***
	GL_BLEND ***(texture controls blending with another color)***
If GL_BLEND selected, second call to glTexEnv() must specify GL_TEXTURE_ENV_COLOR	***4-float array for R,G,B,A blend***



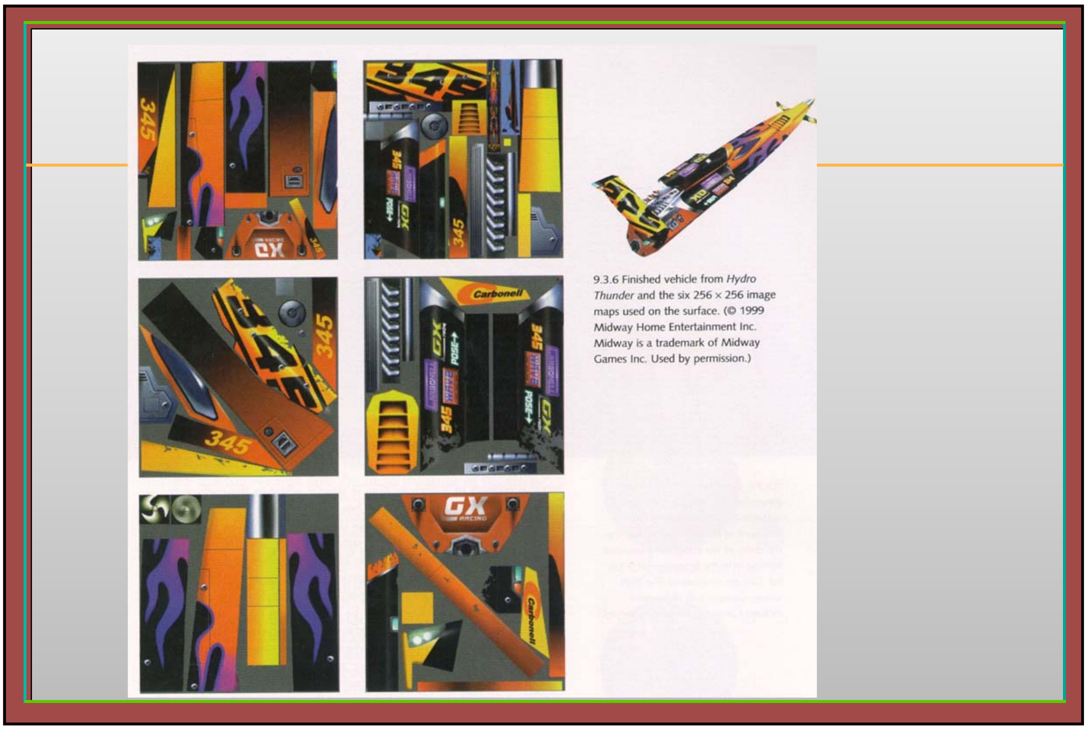
9.3.4 These panels of 256 x 256 pixels contain the texture maps for the character rendered on the opposite page. (READY 2 RUMBLE™ BOXING © 1999 Midway Home Entertainment Inc. All rights reserved. Likeness of Michael Buffer and the READY TO RUMBLE® trademark used under license from Buffer Partnership. All character names are trademarks of Midway Home Entertainment Inc. Midway is a trademark of Midway Games Inc. Used by permission.)

The Art of 3D Computer Animation and Effects
Isaac Kerlow

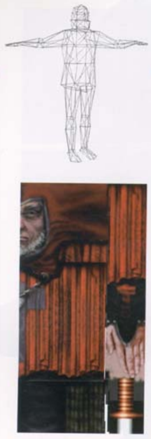
ınd
ed
films
o-
ıst visu-
imper-
the

often
ıaps
ındom
tterns that
l effectively
halt in the fea-
ature film are
'he skin of virtual
ı and depth by ren-
d in Figure 9.2.5.
could have, for
e freckles, the
. Image maps
on with pro-
xcellent solu-
hat might
very difficult.
rendering and
ake the concept
level that actual-
ing information
ts taken on

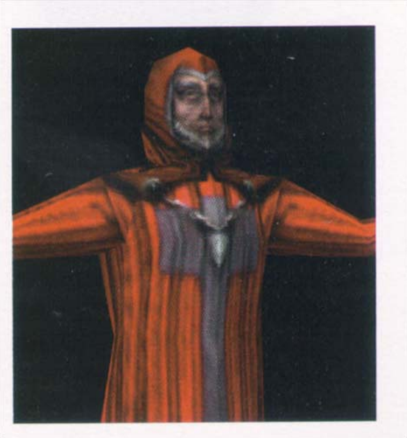
maps onto three-
methods are simple
ealistic effects and others
ıapping projection
considerations with-
ne projection meth-
ndered image better
projection methods

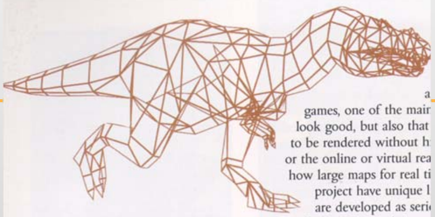


9.3.6 Finished vehicle from *Hydro Thunder* and the six 256 × 256 image maps used on the surface. (© 1999 Midway Home Entertainment Inc. Midway is a trademark of Midway Games Inc. Used by permission.)



9.3.5 The rectangular mosaic of textures contains the color maps for the three-dimensional low resolution model shown in wireframe (top) and shaded modes (opposite page). Notice the economy and effectiveness of the maps once applied to the character. The mosaic is 128 × 256 pixels, and at 72 ppi is roughly 1.8 × 3.5 in. (Created by Mark Lizotte. © 1998 Looking Glass Studios.)



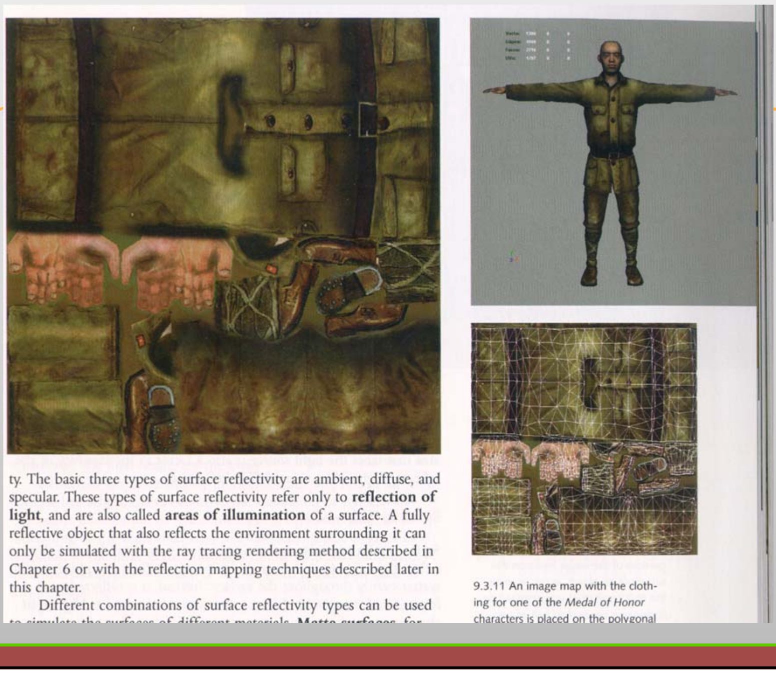


a
games, one of the mair
look good, but also that
to be rendered without h
or the online or virtual rea
how large maps for real ti
project have unique l
are developed as seri
256 × 256 pixels is a c
contain different images that a
the model in real time. Someti
tiled—to fill a large surface. T
ple, uses almost a dozen 256 ×
Figure 9.3.5 uses a single 128
9.3.4, the hair panel for examp
middle left section in the recta
rored and also stretched along
image maps is to make sure th
match, so that the resulting te
map as opposed to many smal

The maps used in the real
have a surface of only 32 × 3
file size of 3.7 Kb. But after t
Playstation .sif texture format
main character in the Spyro t
these texture maps, while mos
maps each. In fact, the majori
games have texture maps that
surfaces, while the rest of thei
shading only in order to cons
ticular—during gameplay.

Generally speaking the p
maps for real time should be k
ally within a fairly limited col

9.3.2 Different color maps were painted for different sections of a T-Rex low resolution polygonal model. The game engine takes care of mapping the different sections of the image onto the right area of the geometry. See Fig. 10.5.5 for the inverse kinematics skeleton for this polygonal model. (Courtesy of Angel Studios.)



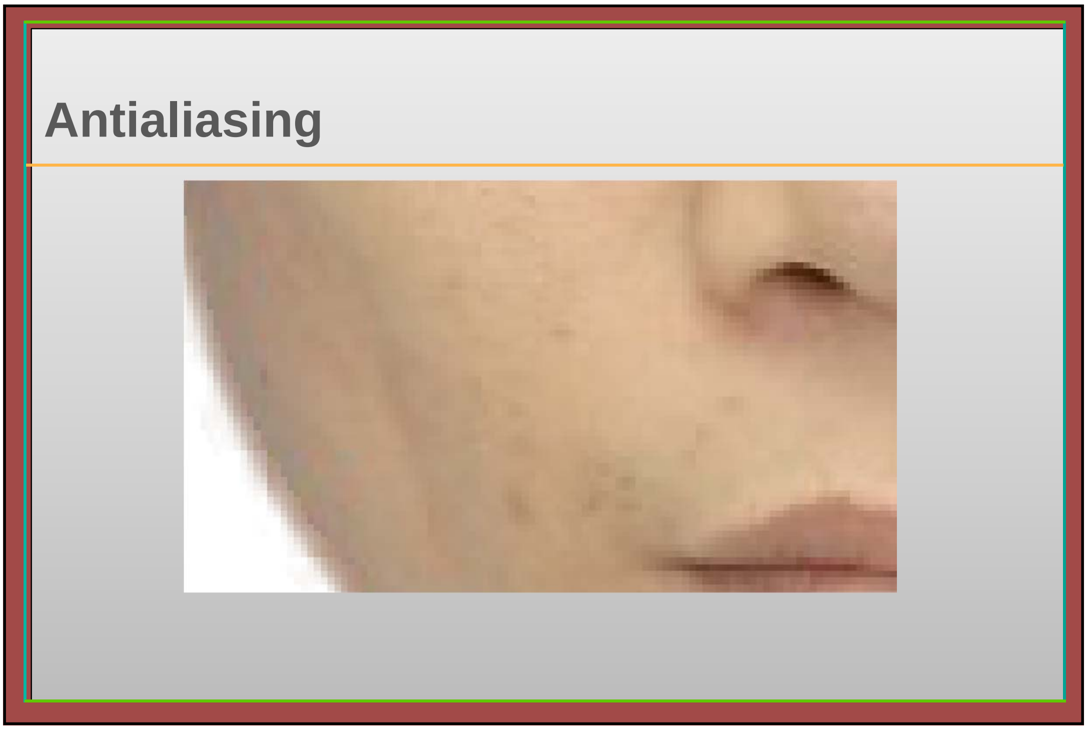
ty. The basic three types of surface reflectivity are ambient, diffuse, and specular. These types of surface reflectivity refer only to **reflection of light**, and are also called **areas of illumination** of a surface. A fully reflective object that also reflects the environment surrounding it can only be simulated with the ray tracing rendering method described in Chapter 6 or with the reflection mapping techniques described later in this chapter.

Different combinations of surface reflectivity types can be used to simulate the surfaces of different materials. **Matte surfaces**, for

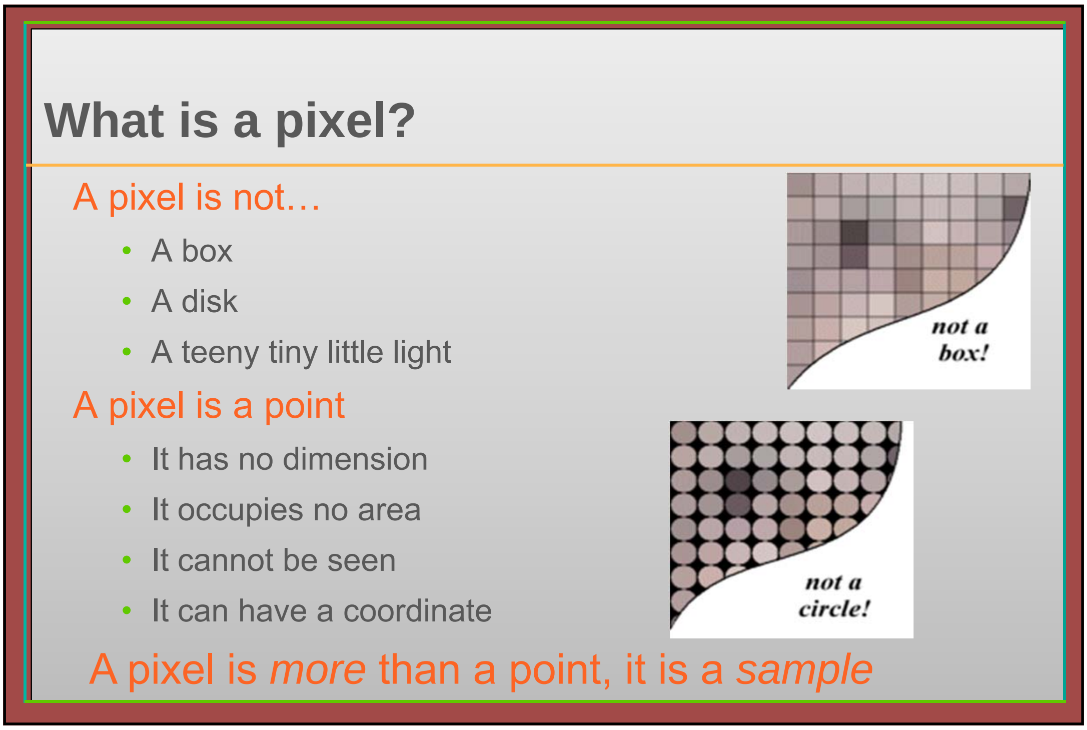
9.3.11 An image map with the clothing for one of the *Medal of Honor* characters is placed on the polygonal

Question 4: Antialiasing

- What is aliasing?
- How to achieve antialiasing?



Antialiasing



What is a pixel?

A pixel is not…

- A box
- A disk
- A teeny tiny little light

A pixel is a point

- It has no dimension
- It occupies no area
- It cannot be seen
- It can have a coordinate

A pixel is *more* than a point, it is a *sample*

Samples

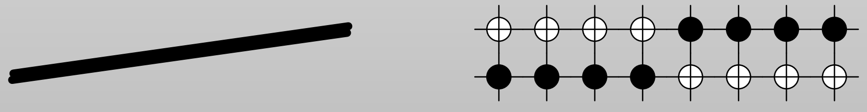
- Most things in the real world are *continuous*
- Everything in a computer is *discrete*
- The process of mapping a continuous function to a discrete one is called *sampling*
- The process of mapping a continuous variable to a discrete one is called *quantization*
- Rendering an image requires *sampling and quantization*

Aliasing

Aliasing is cause by the discrete nature of pixels (Sampling Error).

We tried to sample a line segment so it would map to a 2D raster display

Approximation of lines and circles with discrete points often gives a staircase appearance or "Jaggies”



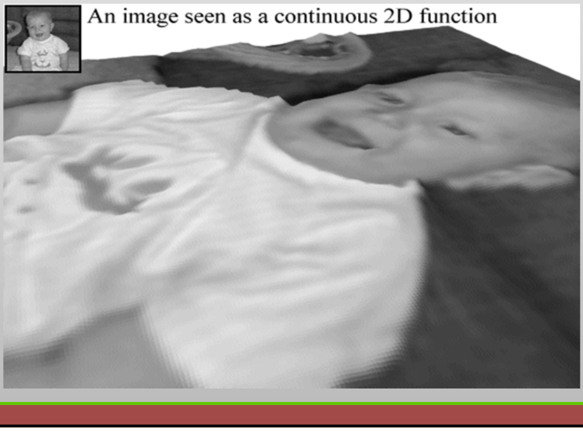
Desired line

Aliased rendering of the line

Samples

Images

An image is a 2D function I(x, y) that specifies intensity for each point (x, y)


An image seen as a continuous 2D function

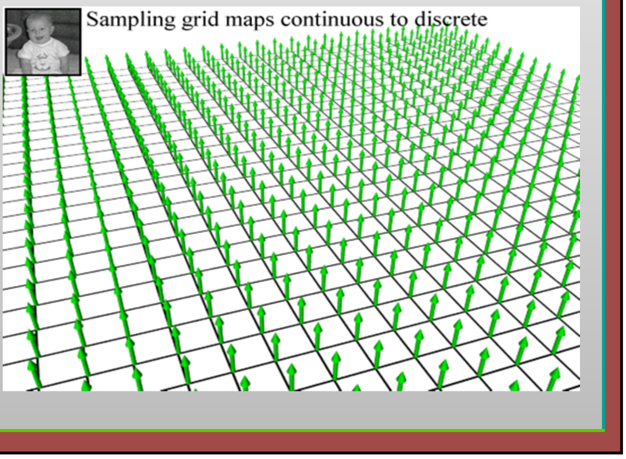
Sampling and Image

- Our goal is to convert the continuous image to a discrete set of samples
- The graphics system's display hardware will attempt to reconvert the samples into a continuous image: reconstruction

Point Sampling an Image

Simplest sampling is on a grid

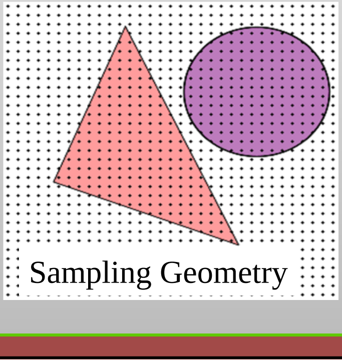
Sample depends solely on value at grid points

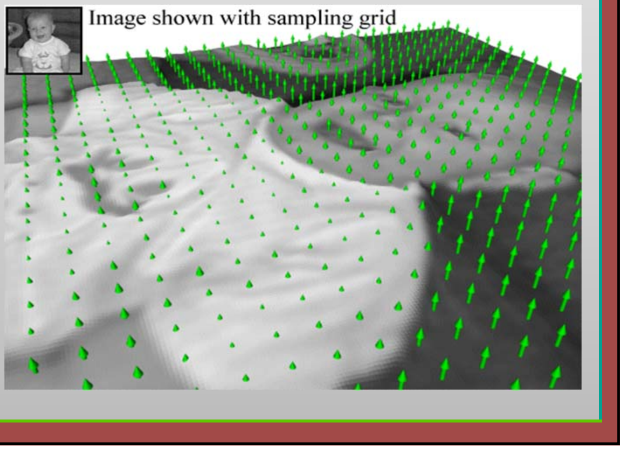


Sampling grid maps continuous to discrete

Point Sampling

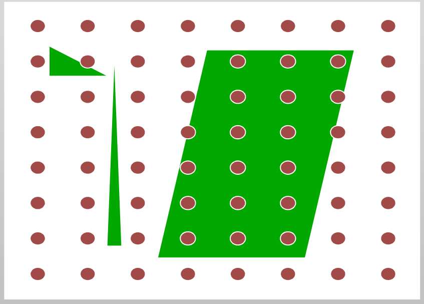
Multiply sample grid by image intensity to obtain a discrete set of points, or samples.





Sampling Errors

Some objects missed entirely, others poorly sampled



Anti-Aliasing

Two general approaches: Area sampling and super-sampling

Area sampling：sample primitives with a box (or Gaussian, or whatever) rather than spikes

- Requires primitives that have area (lines with width)
- Sometimes referred to as pre-filtering

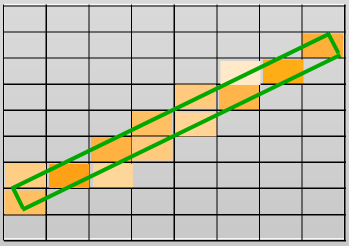
Super-sampling：samples at higher resolution, then filters down the resulting image

- Sometimes called post-filtering
- The prevalent form of anti-aliasing in hardware

Area Sampling

shade pixels according to the area covered by thickened line

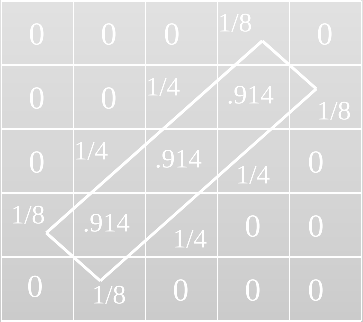
this is unweighted area sampling



a rough approximation formulated by dividing each pixel into a finer grid of pixels

Unweighted Area Sampling

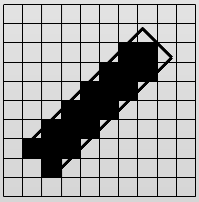
- Consider a line as having thickness
- Consider pixels as little squares
- Fill pixels according to the proportion of their square covered by the line

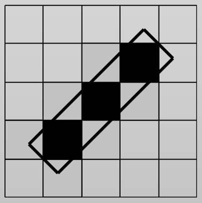


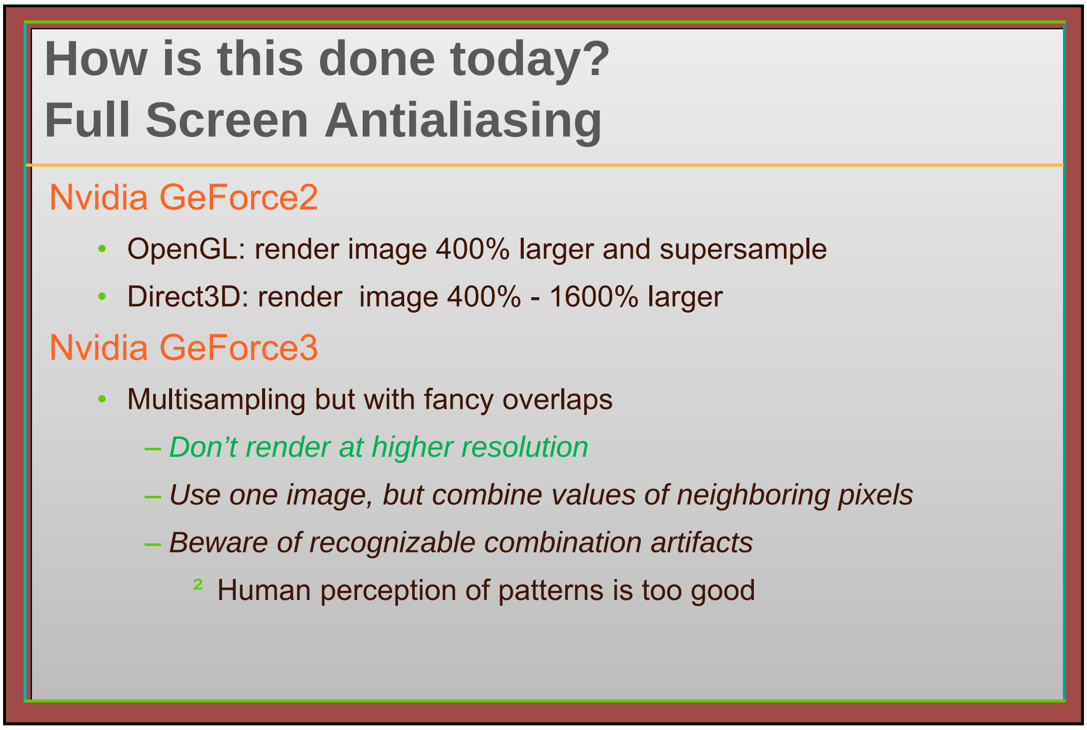
0	0	0	1/8	0
0	0	1/4	.914	1/8
0	1/4	.914	1/4	0
1/8	.914	1/4	0	0
0	1/8	0	0	0

Super-sampling

- Sample at a higher resolution than required for display, and filter image down
- 4 to 16 samples per pixel is typical
- Samples might be on a uniform grid, or randomly positioned, or other variants
- Number of samples can be adapted







How is this done today? Full Screen Antialiasing

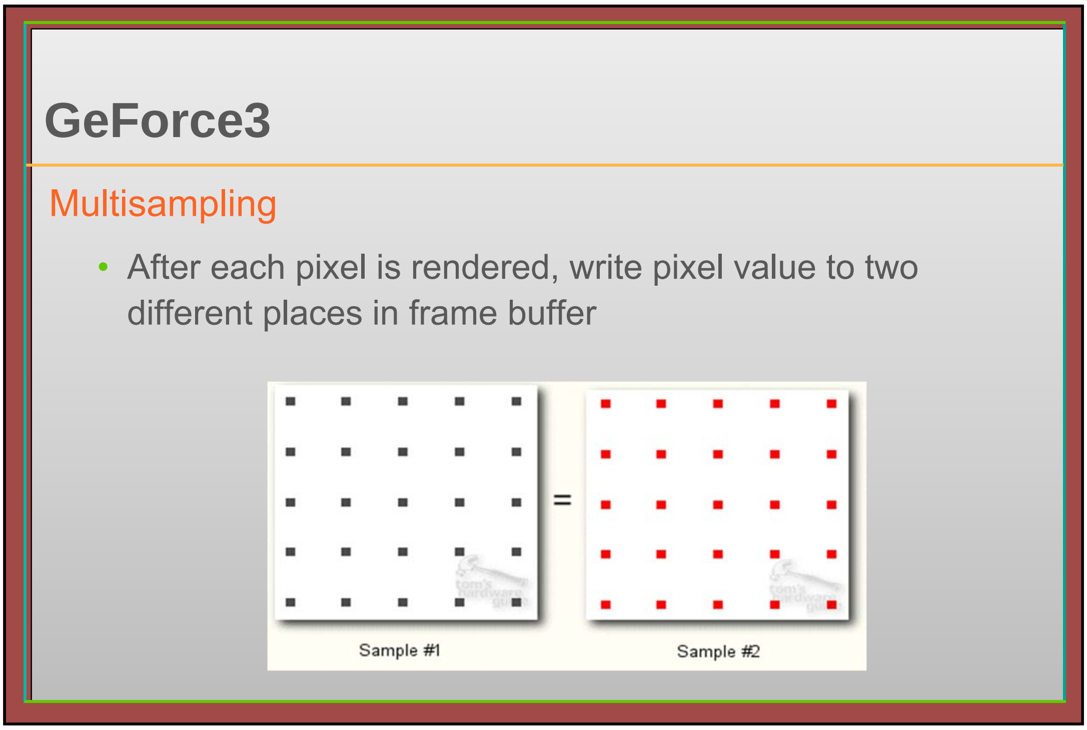
Nvidia GeForce2

- OpenGL: render image 400% larger and supersample
- Direct3D: render image 400% - 1600% larger

Nvidia GeForce3

- Multisampling but with fancy overlaps
  - *Don't render at higher resolution*
  - *Use one image, but combine values of neighboring pixels*
  - *Beware of recognizable combination artifacts*

    ² Human perception of patterns is too good



GeForce3

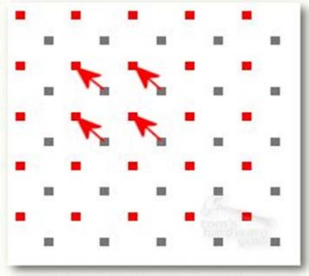
Multisampling

- After each pixel is rendered, write pixel value to two different places in frame buffer

GeForce3 - Multisampling

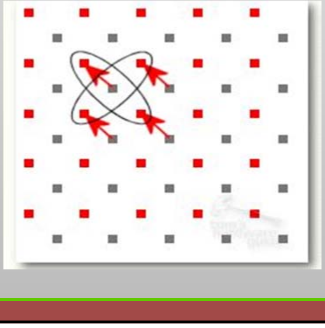
After rendering two copies of entire frame

- Shift pixels of Sample #2 left and up by ½ pixel
- Imagine laying Sample #2 (red) over Sample #1 (black)

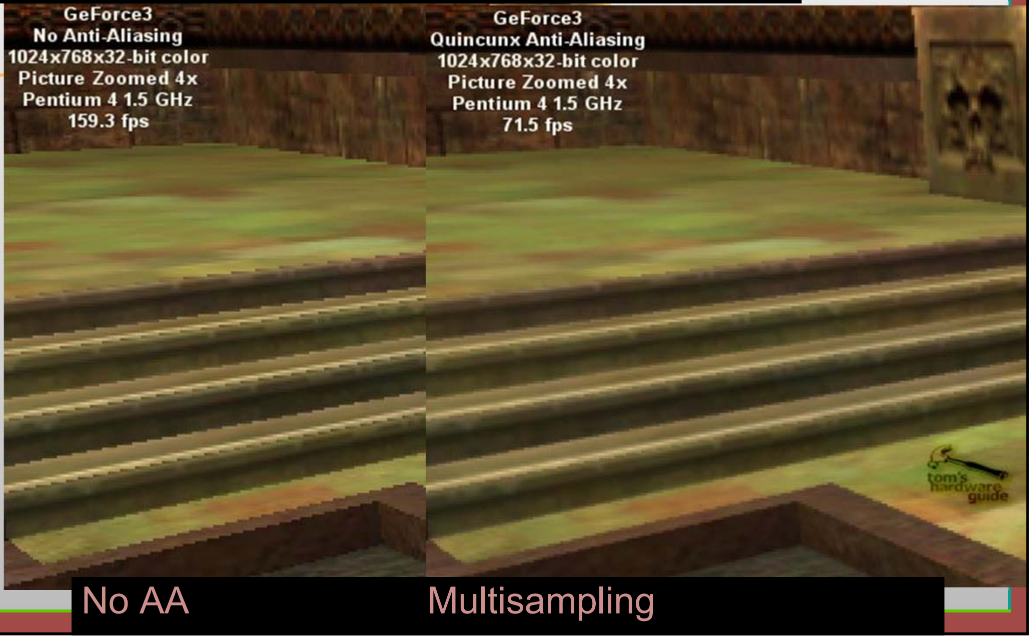


GeForce3 - Multisampling

Resolve the two samples into one image by computing average between each pixel from Sample 1 (black) and the four pixels from
Sample 2 (red) that
are 1/ sqrt(2) pixels
away

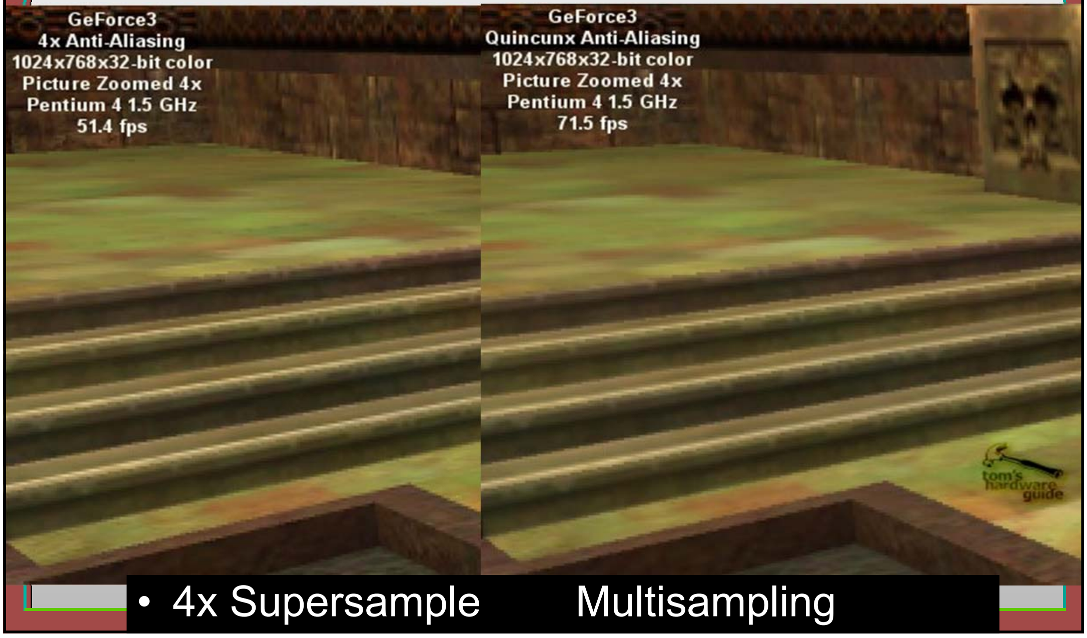


GeForce3 - Multisampling

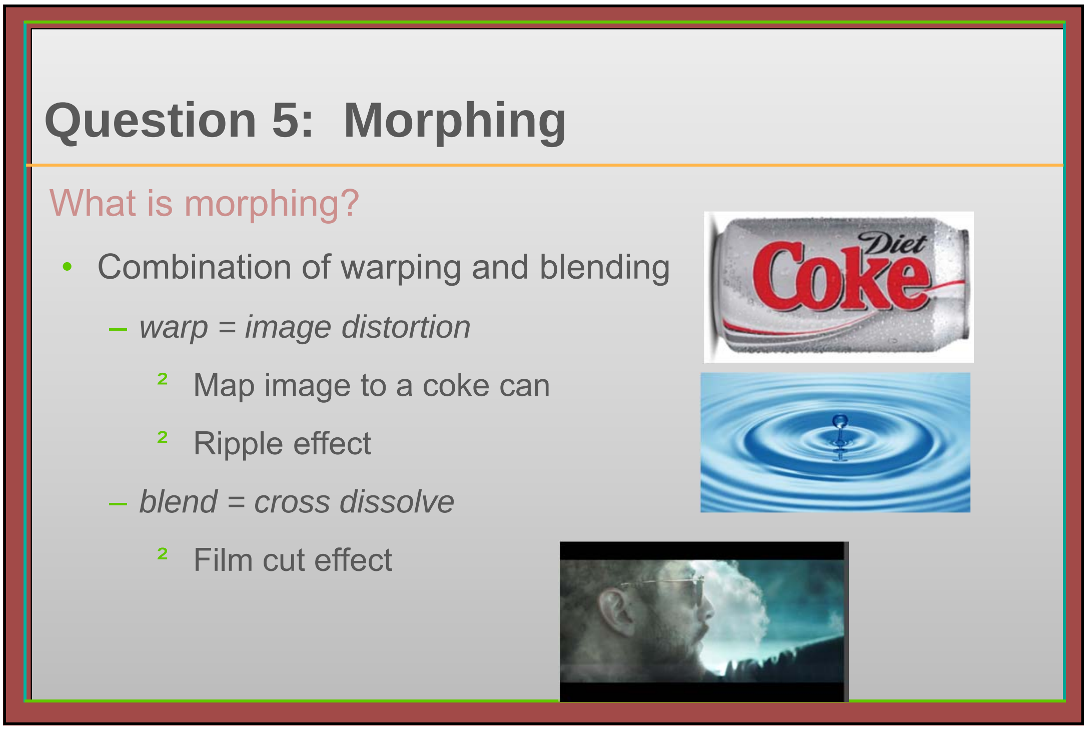


No AA Multisampling

GeForce3 - Multisampling



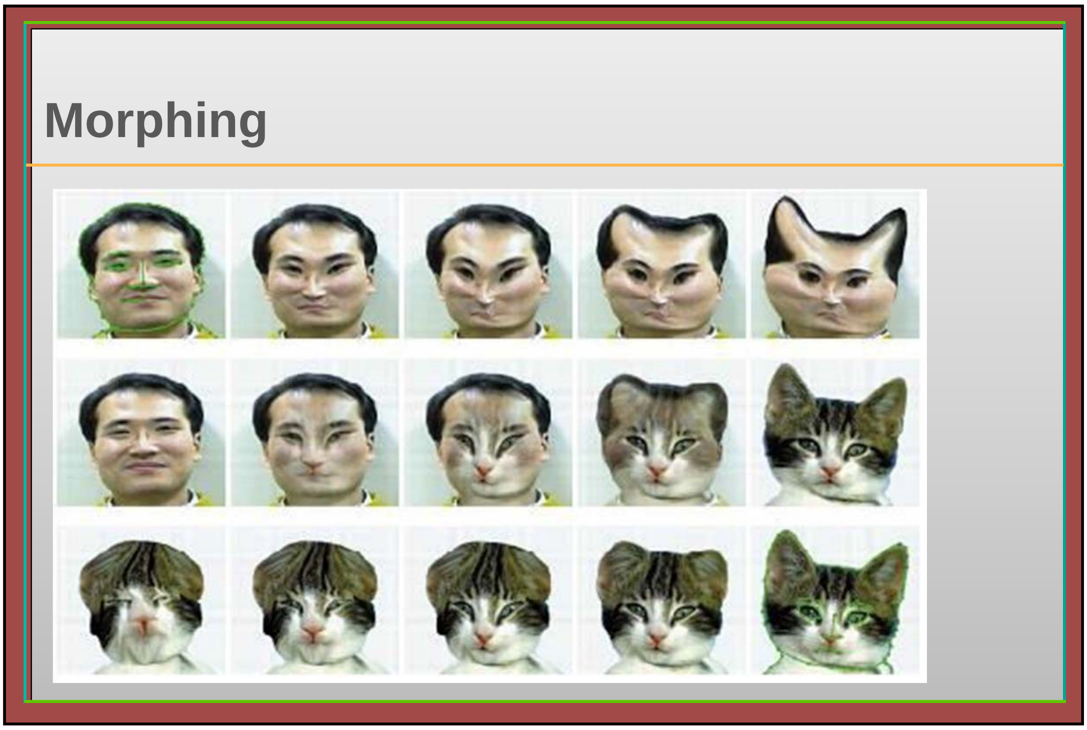
- 4x Supersample Multisampling



Question 5: Morphing

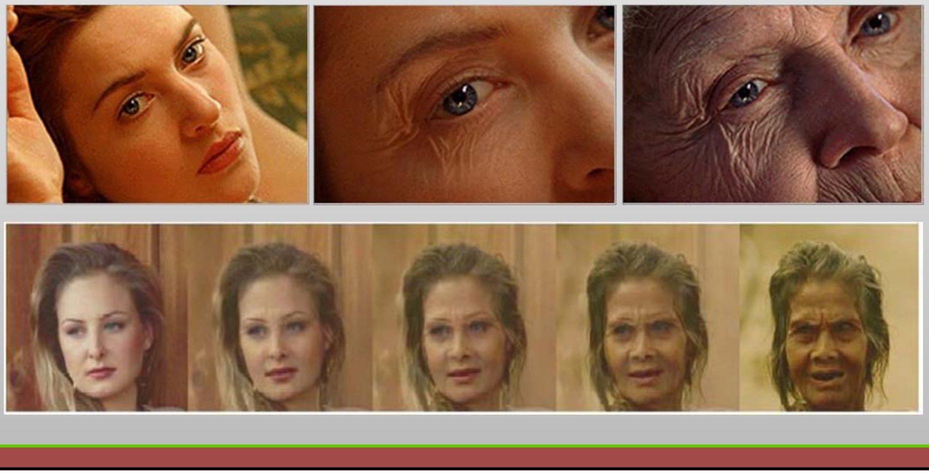
What is morphing?

- Combination of warping and blending
  - – *warp = image distortion*
    - ² Map image to a coke can
    - ² Ripple effect
  - – *blend = cross dissolve*
    - ² Film cut effect



Morphing

Morphing



Ways to morph

3D Techniques

- Interpolate between corresponding vertices
  - *Models must align somehow*
  - *Polygon transformations may be difficult*

2D Techniques

- Cross-dissolve
  - *Difficult to align*
- Pixelize the images and move the “tiles”
  - *Tile paths must be determined somehow*

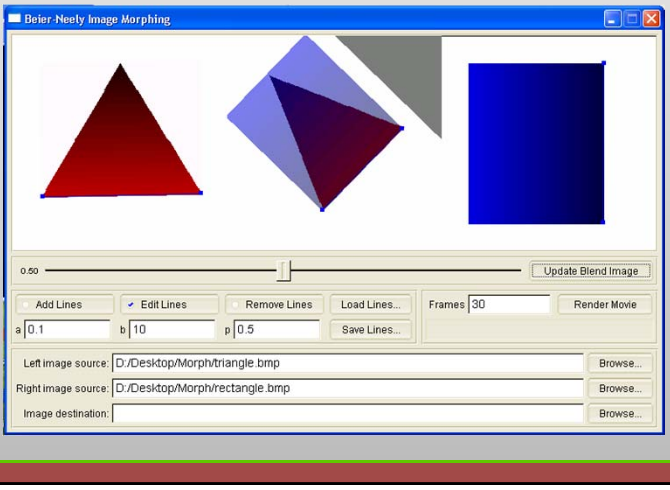
Beier-Neely Morphing

Simple / effective morphing

- User identifies key features with line segments
- Everything else is automatic

Beier-Neely Morphing

How do we explain this?



Beier-Neely Image Morphing

0.50

☐ Add Lines ☑ Edit Lines ☐ Remove Lines

a 0.1 b 10 p 0.5

Frames 30

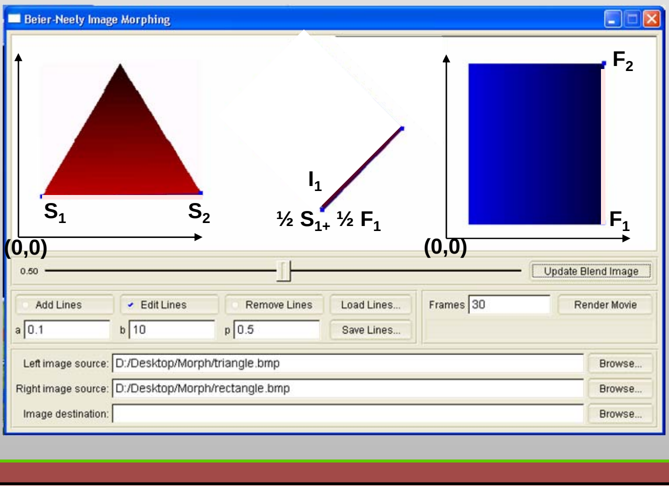
Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

Transformation with One Pair of Lines

Interpolate between lines $S_1 \rightarrow S_2$ and $F_1 \rightarrow F_2$



0.50

a 0.1 b 10 p 0.5

Frames 30

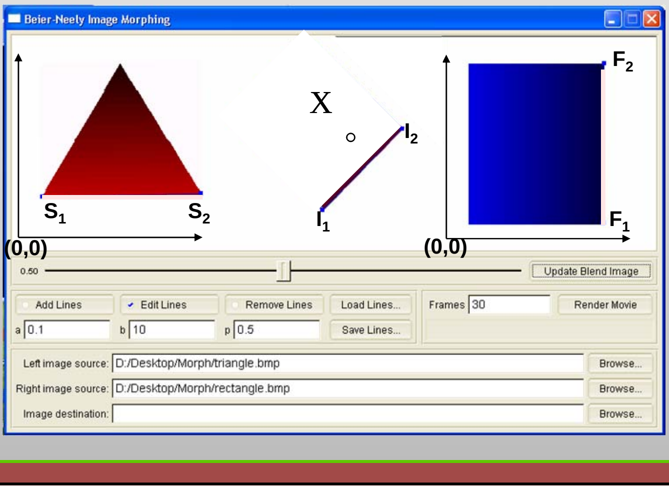
Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

How would start image morph?

What is color from source image at X in intermediate image?



0.50

a 0.1 b 10 p 0.5

Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

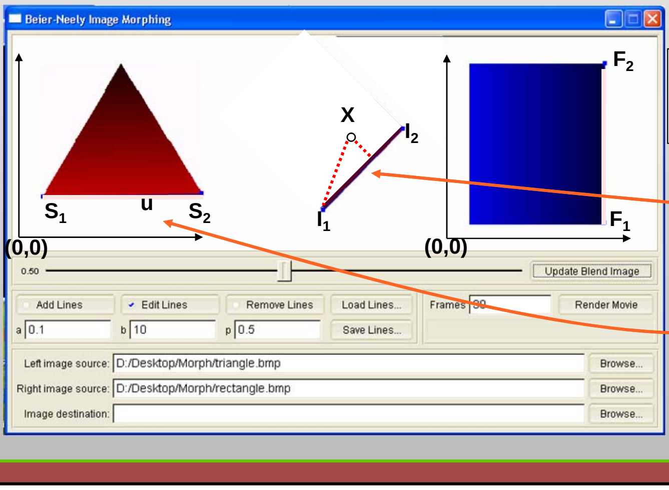
Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

Build Common Coordinate System

Find projection

the position along the line



a 0.1

b 10

p 0.5

Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

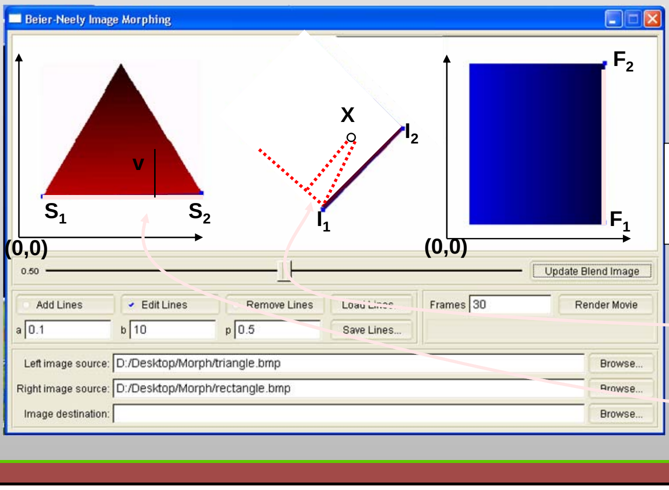
Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

$$u = \frac{(X - I_1)\bullet(I_2 - I_1)}{\|(I_2 - I_1)\|}$$

Build Common Coordinate System

Find projection on perpendicular



a 0.1

b 10

p 0.5

Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

distance from the line

$$v = \frac{(X - I_1)\bullet(I_2 - I_1)^{\perp}}{\|(I_2 - I_1)\|}$$

Find Projection

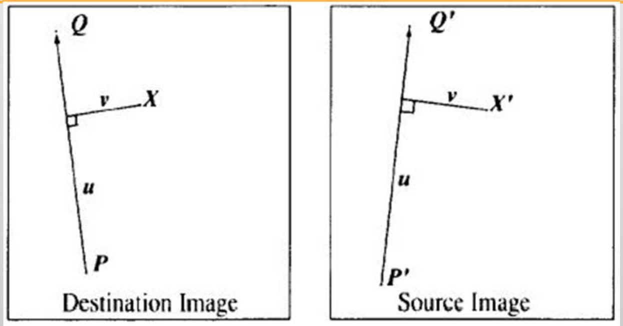
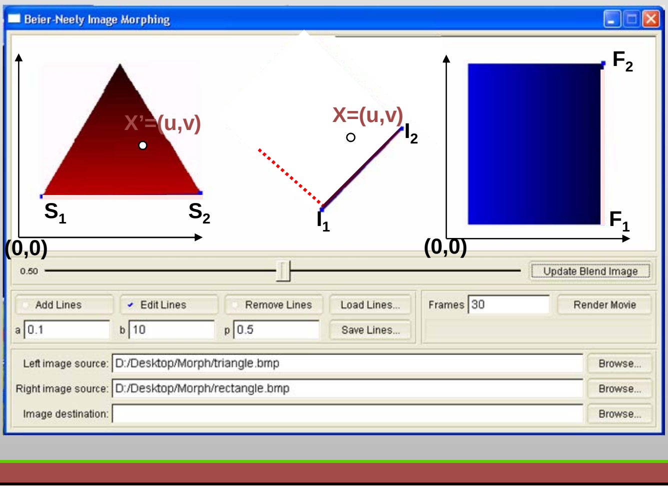


Figure 1: Single line pair

```
For each pixel $X$ in the destination image
    find the corresponding $u,v$
    find the $X'$ in the source image for that $u,v$
    destinationImage($X$) = sourceImage($X'$)
```

Sample color from initial image



0.50

☑ Edit Lines

a 0.1 b 10 p 0.5

Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

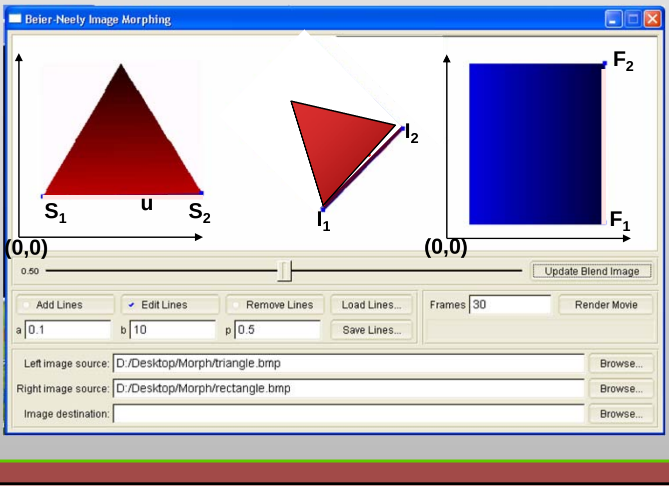
Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

$$X' = S_1 + u(S_2 - S_1) + \frac{v(S_2 - S_1)^{\perp}}{\|(S_2 - S_1)\|}$$

$$I_D(X) \leftarrow I_S(X')$$

Repeat for all pixels in image



0.50

a 0.1 b 10 p 0.5

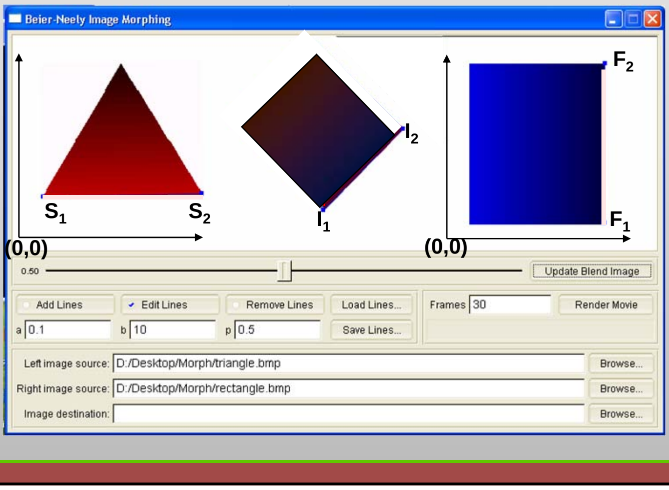
Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

Now do the same thing for the final image



0.50

a 0.1 b 10 p 0.5

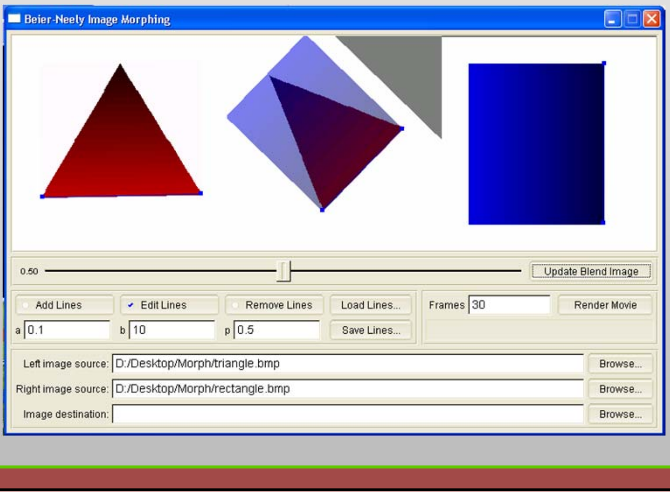
Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

Now do a 50/50 blend of two warped images



0.50

a 0.1 b 10 p 0.5

Frames 30

Left image source: D:/Desktop/Morph/triangle.bmp

Right image source: D:/Desktop/Morph/rectangle.bmp

Image destination:

Blends between Two Images

- We define corresponding lines in I_0 and I_1.
- Each intermediate frame I of the metamorphosis is defined by creating a new set of line segments by interpolating the lines from their positions in I_0 to the positions in I_1.
- Both images I_0 and I_1 are distorted toward the position of the lines in I. These two resulting images are crossdissolved throughout the metamorphosis