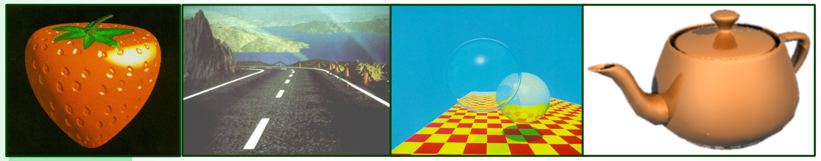
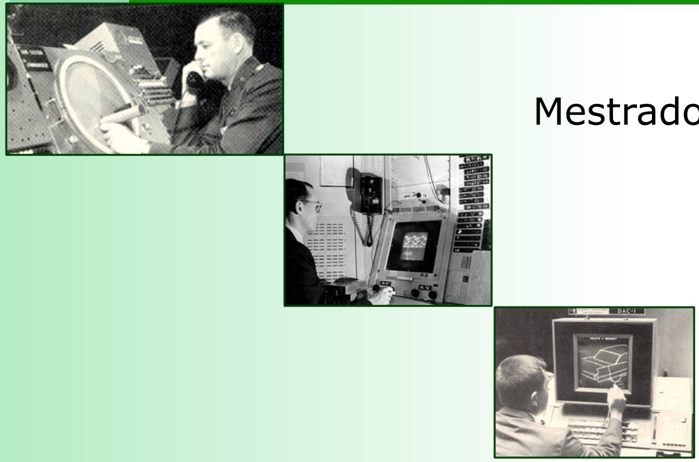


Chap. 5 Textures



Mestrado em Engenharia Informática - 1º ano, 2º semestre

Overview

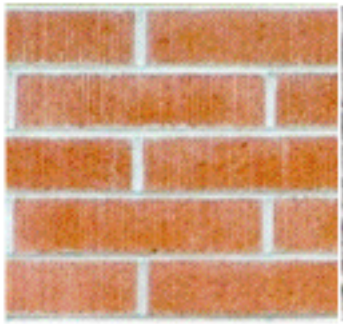
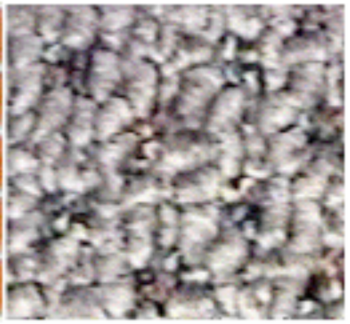
- Objectives
- Notion of texture
- Motivation
- Texture mapping, texture patterns, and texels
- Mapping textures to polygons, texture interpolation
- Rasterization: texture application modes
- Mapping textures to geometric objects
  - planar mapping
  - cylindrical mapping
  - spherical mapping
  - box mapping
- Wrapping modes
- ...

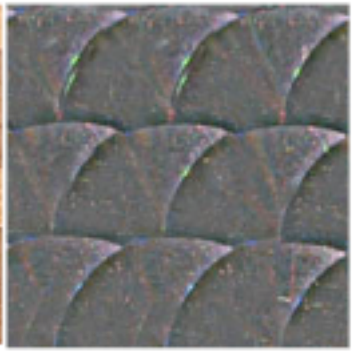
Objectives

- Introduce Mapping Methods
  - Texture Mapping
  - Environmental Mapping
  - Bump Mapping
  - Light Mapping
- Consider two basic strategies
  - Manual coordinate specification
  - Two-stage automated mapping

Notion of Texture

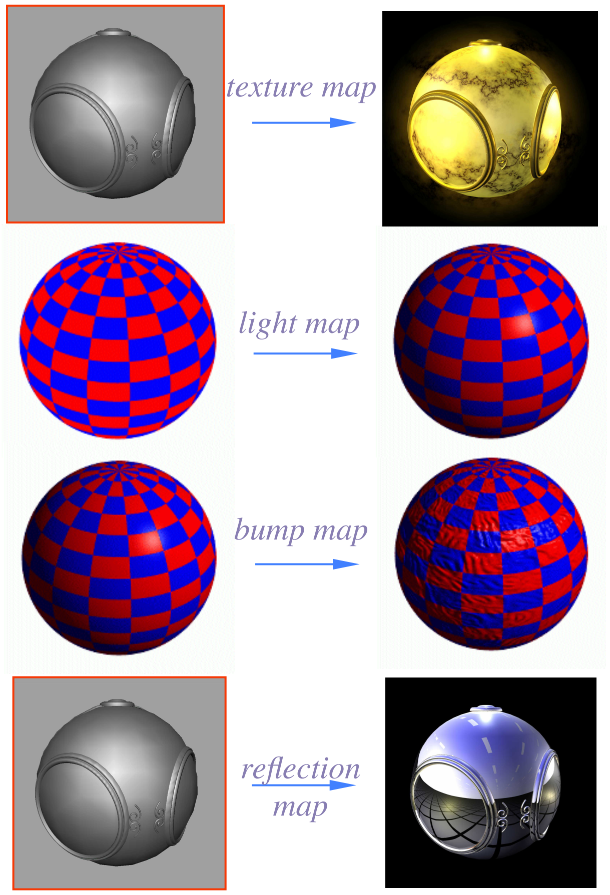
- A **texture** is an image with red, green, blue and alpha components...
- **Texture Mapping**: a method to create complexity in an image without the overhead of building large geometric models.



Motivation: (1) adding realism

- Objects rendered using Phong reflection model and Gouraud or Phong interpolated shading often appear rather ‘plastic’ and ‘floating in air’
- Texture effects can be added to give more realistic looking surface appearance
  - Texture mapping
    - Texture mapping uses pattern to be put on a surface of an object.
  - Light maps
    - Light maps combine texture and lighting through a modulation process
  - Bump mapping
    - Smooth surface is distorted to get variation of the surface
  - Environmental mapping
    - Environmental mapping (reflection maps) – enables ray-tracing like output



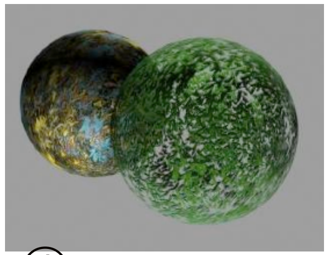
Motivation: (2) adding surface detail

- The most obvious solution is not the best
  - breaking the scene into smaller and smaller polygons increases the detail
  - But it is very hard to model and very time-consuming to render
- Preferred solution is texture mapping
  - typically a 2D image ‘painted’ onto objects
- *Examples*:
  - Model t-shirt with logo ①
    - no need to model the letters and engine with triangles
    - use large base polygon
    - color it with the photograph
  - Subtle wall lighting ②
    - No need to compute it at every frame
    - No need to model it with a lot of constant color triangle
    - Past photograph on large polygon
  - Non-planar surfaces also work ③
    - subdivide surface into planar patches
    - assign photograph subregions to each individual patch
  - Examples of modulating color, bumpiness, shininess, transparency with identical sphere geometry ④

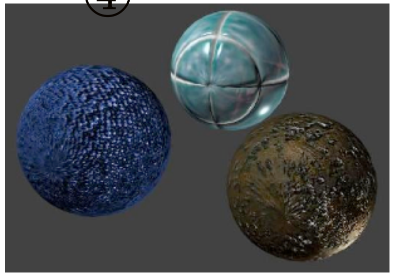
②

①

③

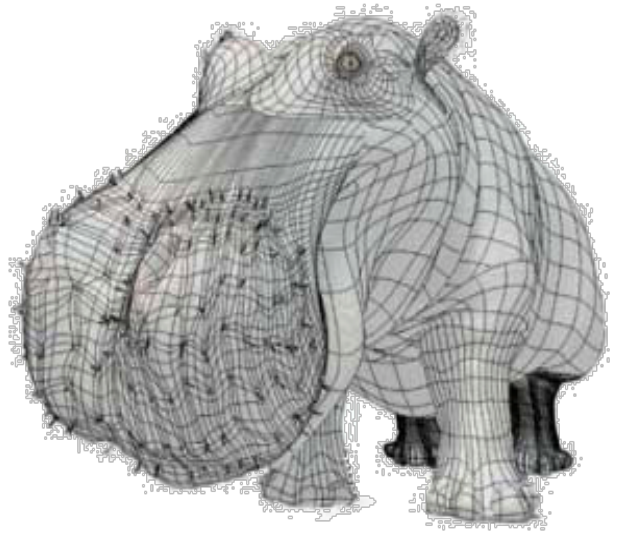


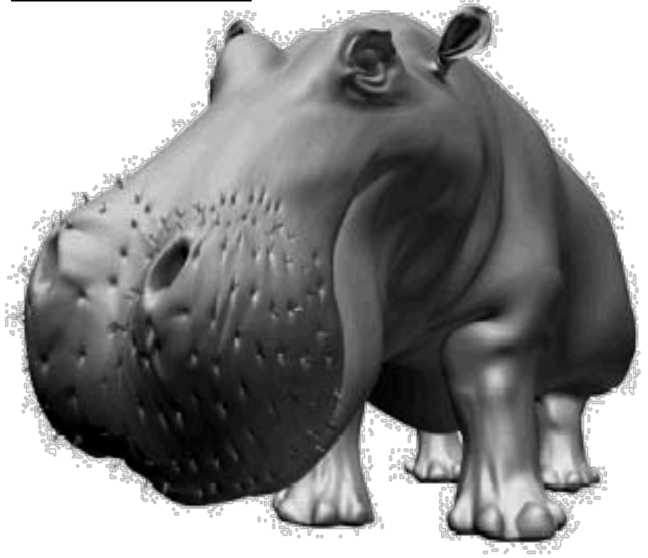
④

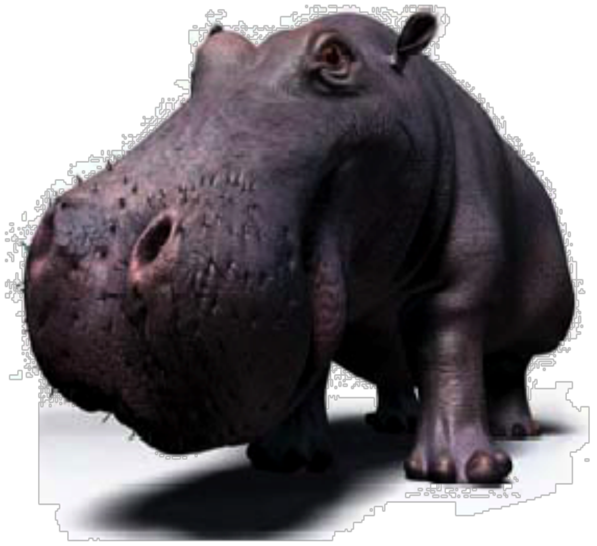


Textures: at what point do things start to looking real?

- Surfaces “in the wild” are very complex
- Cannot model all the fine variations
- We need to find ways to add **surface detail**. How?


geometric model

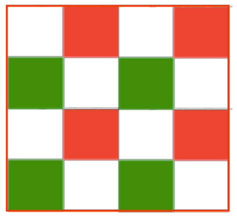

geometric model
+
shading

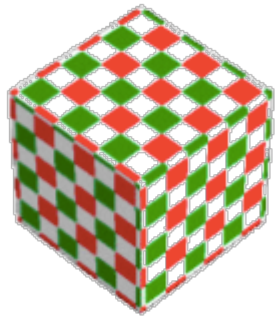

geometric model
+
shading
+
textures

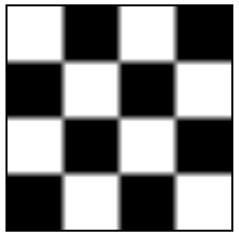
Texture mapping, texture pattern, and texels

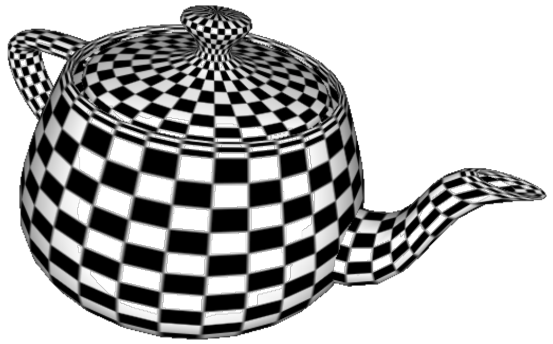
- Developed by Catmull (1974), Blinn and Newell (1976), and others.
- **Texture mapping**: adds surface detail by mapping texture patterns onto the surface.
- Pattern is repeated. For example, the **texture pattern** for the cube aside is the following:

- Texel: short for “texture element”.
- A **texel** is a pixel on a texture. For example, an 128x128 texture has 128x128 texels. On screen this may result in more or fewer pixels depending on how far away the object is on which the texture is used and also on how the texture is scaled on the object.







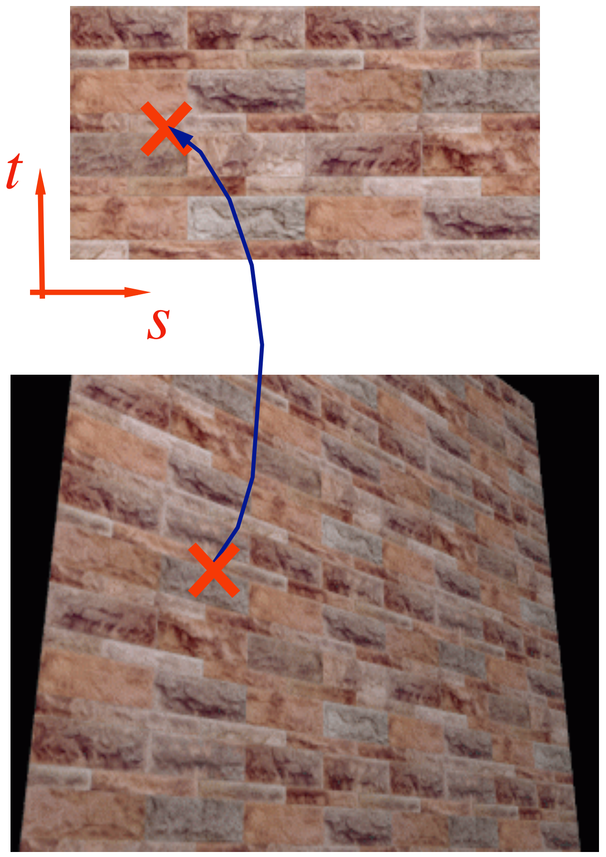


Mapping Techniques

- Texture Mapping
- Environmental Mapping
- Bump Mapping
- Light Mapping

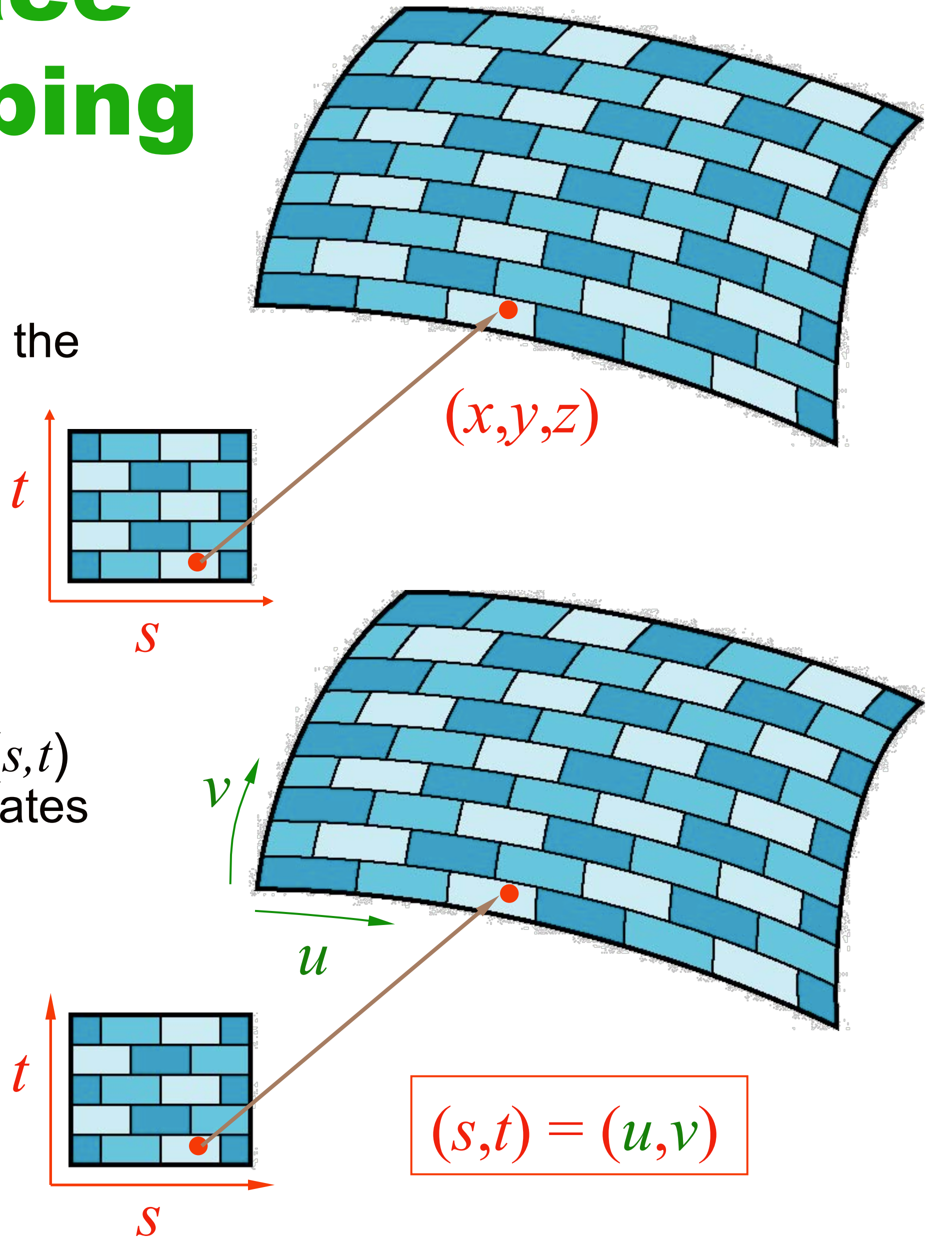
Texture Mapping

- Question to address: Which point of the texture do we use for a given point on the surface?
- The texture is simply an image, with a 2D coordinate system (s,t)
  - Parameterize points in the texture with 2 coordinates: (s,t)
- Define the mapping from (x,y,z) in world space to (s,t) in texture space
  - To find the color in the texture, take an (x,y,z) point on the surface, map it into texture space, and use it to look up the color of the texture
- With polygons:
  - Specify (s,t) coordinates at vertices
  - Interpolate (s,t) for other points based on given vertices



Texture to Surface Coordinate Mapping

- The basic problem is how to find the texture to surface mapping
- Given a texture position (s,t), what is the position (x,y,z) on the surface?
- Appear to need three functions:
  - $x = \mathbf{X}(s, t)$
  - $y = \mathbf{Y}(s, t)$
  - $z = \mathbf{Z}(s, t)$
- So, there are 2 coordinate systems involved, the 2D image coordinates (s,t) and the 2D parameterization coordinates (u,v) that we assign to the 3D object
- Texture coordinates (s,t)
  - Used to identify points in the texture image
- Parametric coordinates (u,v)
  - Used to map the 3D surface with 2D parameters

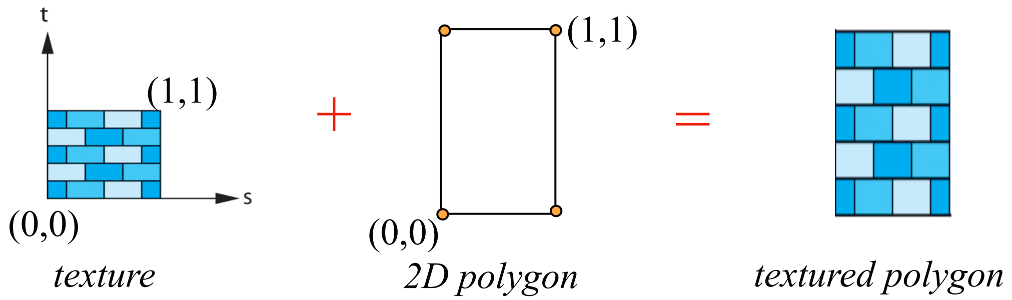


How to set (u,v) parametric coordinates?

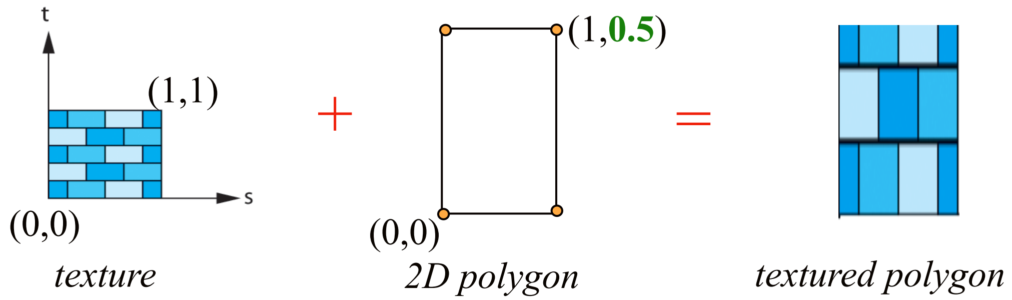
- Set the coordinates manually
  - Set the texture coordinates for each vertex ourselves
- Automatically compute the coordinates
  - Use an algorithm that sets the texture coordinates for us

Manually specifying the coordinates

- We can manually specify the texture coordinates at each vertex



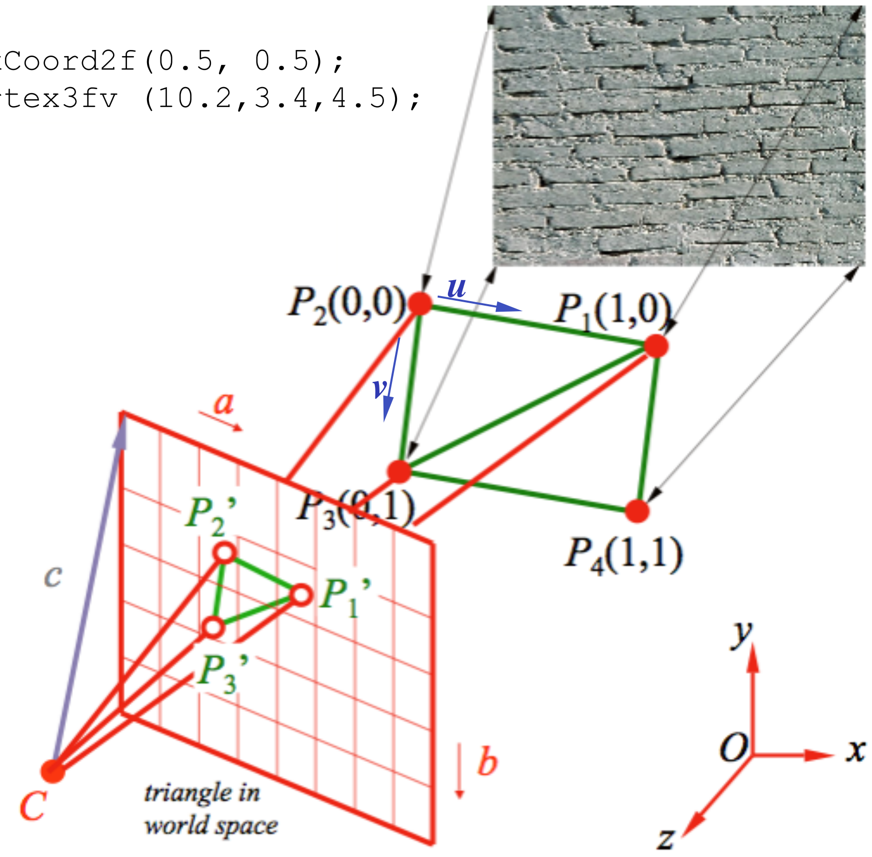
- We can chose alternate texture coordinates



Mapping Texture to Polygons

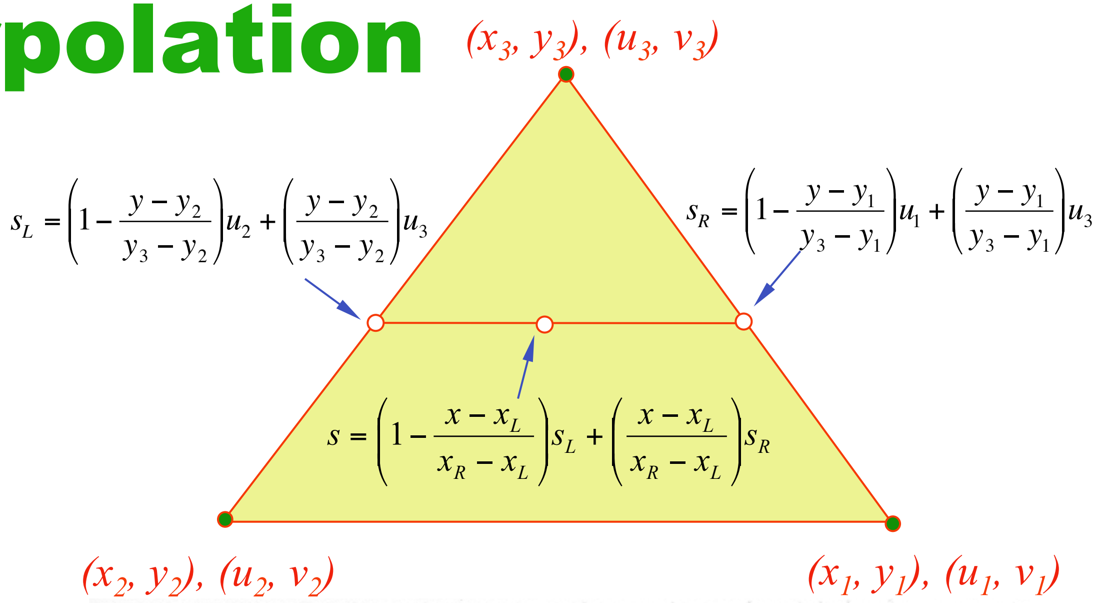
- For polygon texture mapping, we explicitly define the (*u*,*v*) coordinates of the polygon vertices
- That is, we pin the texture at the vertices
- We **interpolate** within the triangle <u>at the time of scan converting</u> into screen space

```
…
glTexCoord2f(0.5, 0.5);
glVertex3fv (10.2,3.4,4.5);
…
```



Texture Interpolation

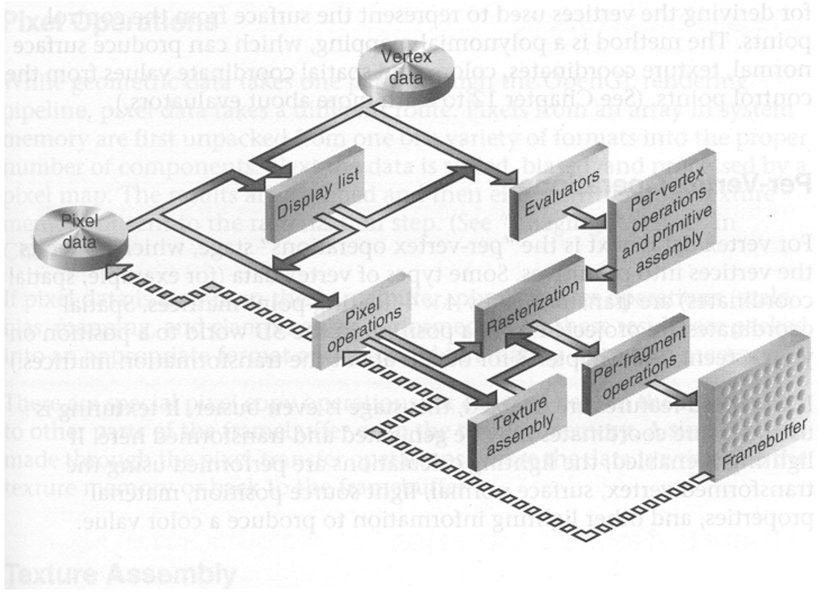
- Interpolation is done during *scan conversion*, similar as is done for Gouraud interpolated shading
- But rather than interpolate to get RGB values, we get (*u*,*v*) values which point to elements of texture map.
- Thus, texture mapping is done in canonical screen space as the polygon is rasterized
- When describing a scene, you assume that texture interpolation will be done in world space



$$s_L = \left(1 - \frac{y - y_2}{y_3 - y_2}\right) u_2 + \left(\frac{y - y_2}{y_3 - y_2}\right) u_3$$

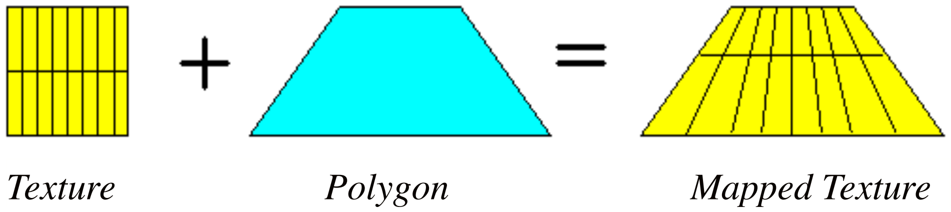
$$s_R = \left(1 - \frac{y - y_1}{y_3 - y_1}\right) u_1 + \left(\frac{y - y_1}{y_3 - y_1}\right) u_3$$

$$s = \left(1 - \frac{x - x_L}{x_R - x_L}\right) s_L + \left(\frac{x - x_L}{x_R - x_L}\right) s_R$$

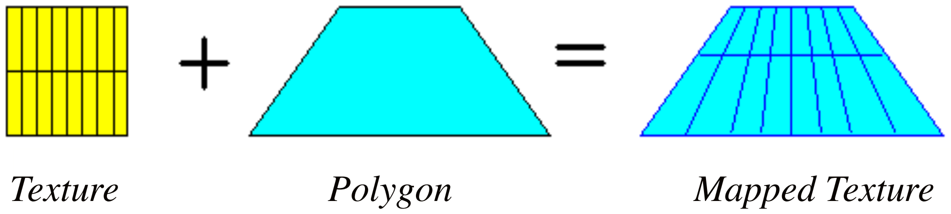


Rasterization: Texture Application Modes

- After a texture value is retrieved (may be further transformed), the resulting values are used to modify one or more polygon/surface attributes
- Called ***combine functions*** or ***texture blending operations***:
  - **replace:** replace surface color with texture color
  - **decal**: replace surface color with texture color, blend the color with underlying color with an alpha texture value, but the alpha component in the framebuffer is not modified
  - **modulate**: multiply the surface color by the texture color (shaded + textured surface). Need this for multitexturing (i.e., lightmaps).
  - **blend**: similar to modulation but add alpha-blending



REPLACE operation



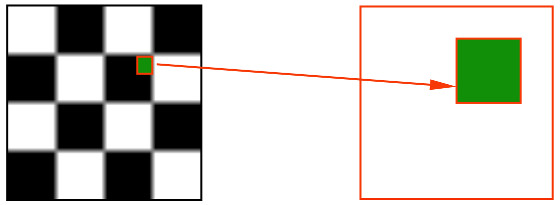
MODULATE operation

Texture Mapping Issues

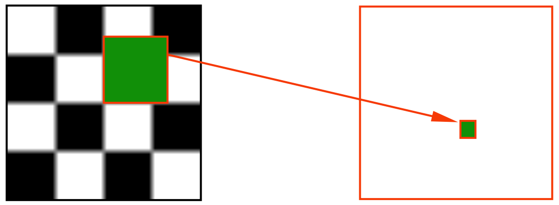
- What should happen when we zoom in close or zoom out far away?
- How do we generate texture coordinates?
- What happens if we use texture coordinates less than zero or greater than one?
- Are texture maps only for putting color on objects?

Texture to Surface Mapping

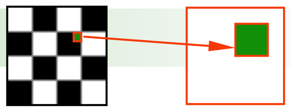
- Texture map to surface takes place during rendering.
- Similar to smooth shading method:
  - Triangle rasterized
  - Each pixel mapped back to the texture
  - Use known values at vertices to interpolation over the texture
- Each pixel is associated with small region of surface *and* to a small area of texture.
- There are 3 possibilites for association:
  1. one texel to one pixel (rare)
  2. Magnification: one texel to many pixels
  3. Minification: many texels to one pixel



Magnification



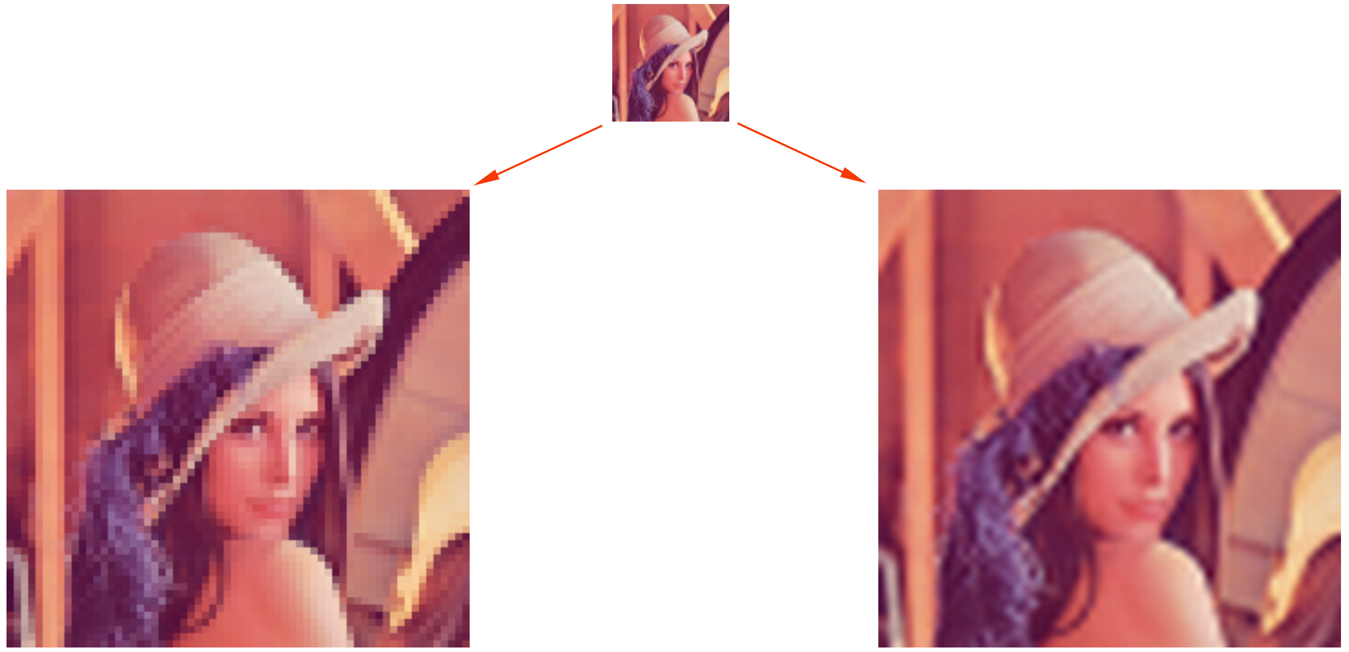
Minification



Texture to Surface Mapping

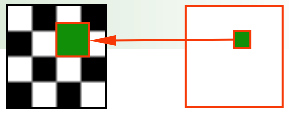
Zoom In: Magnification Filter

- Pixel maps to a small portion of *one* texel
- Results in many pixels mapping to same texel
- Without a filtering method, aliasing is common
- Magnification filter: smooths transition between pixels



many pixels correspond to one texel
→ “blockiness” / jaggies / aliasing

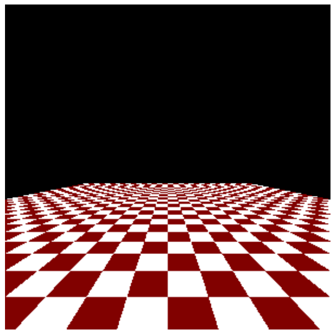
solution: apply averaging
(magnification filter)

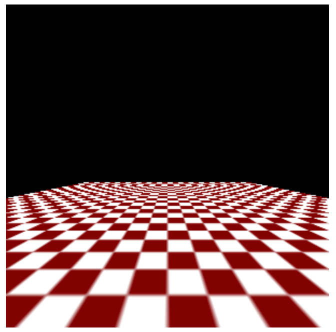


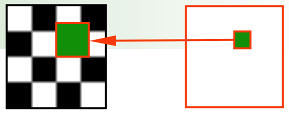
Texture to Surface Mapping

Zoom Out: Minification Filter

- One pixel maps to many texels
- Common with perspective foreshortening

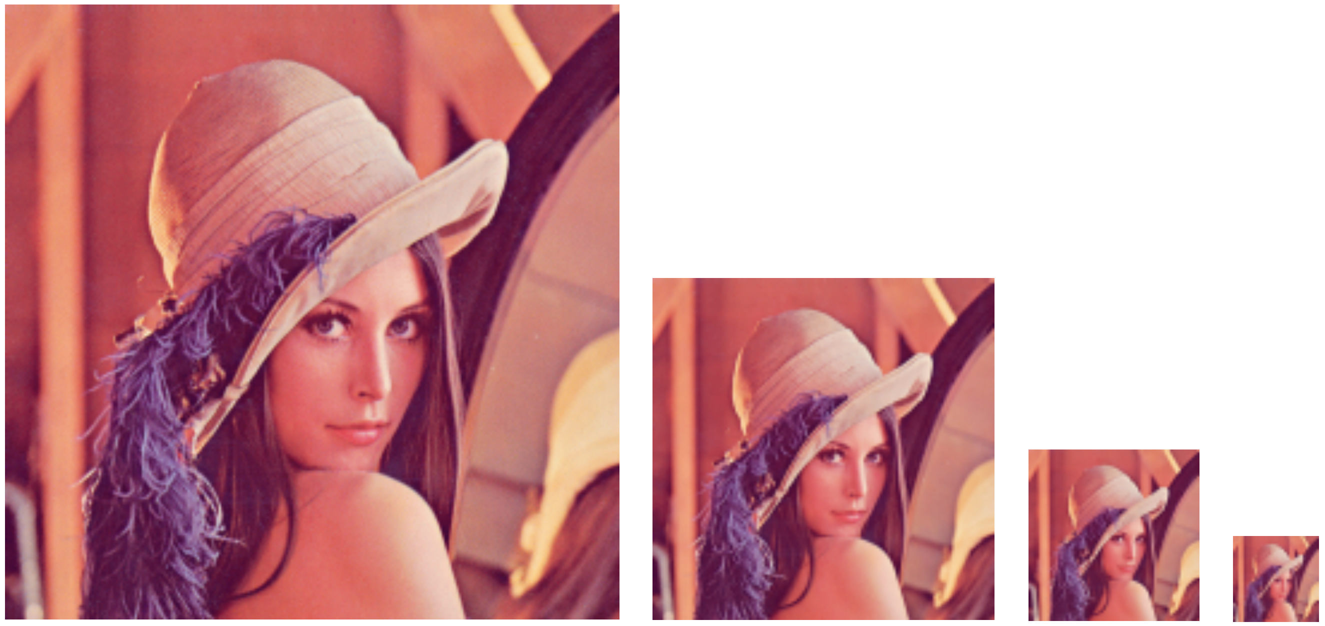

Perspective foreshortening and poor texture mapping causes checkerboard to deform


Mipmaps improve the mapping, returning more form to the checkerboard



Texture to Surface Mapping
Better Min Filter: Mipmaps

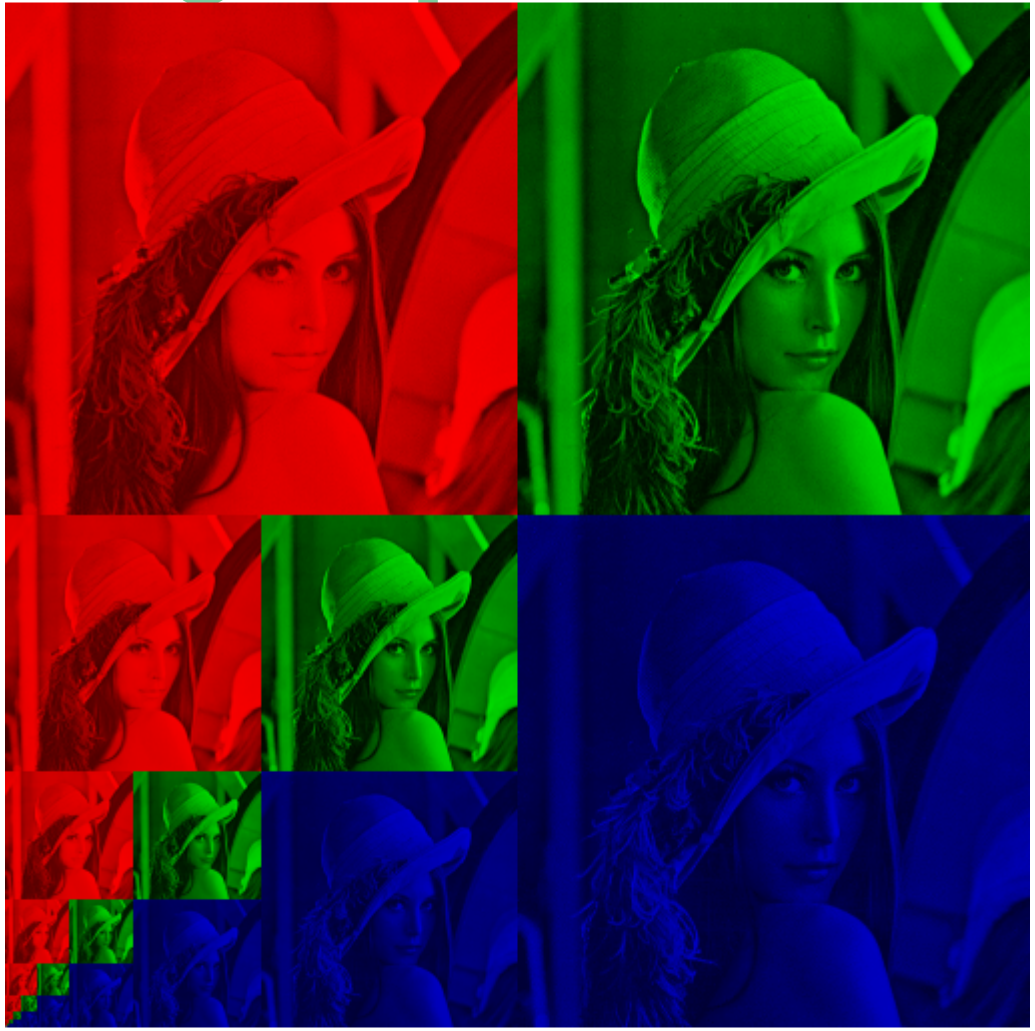
- "mip" stands for *multum in parvo*, or "many things in a small place"
- Basic idea: Create many textures of decreasing size and use one of these subtextures when appropriate
- Pre-filter textures = mipmaps



Texture to Surface Mapping

Mipmaps: Storage Optimization

- Must provide all sizes of texture from input to 1x1 in powers of 2

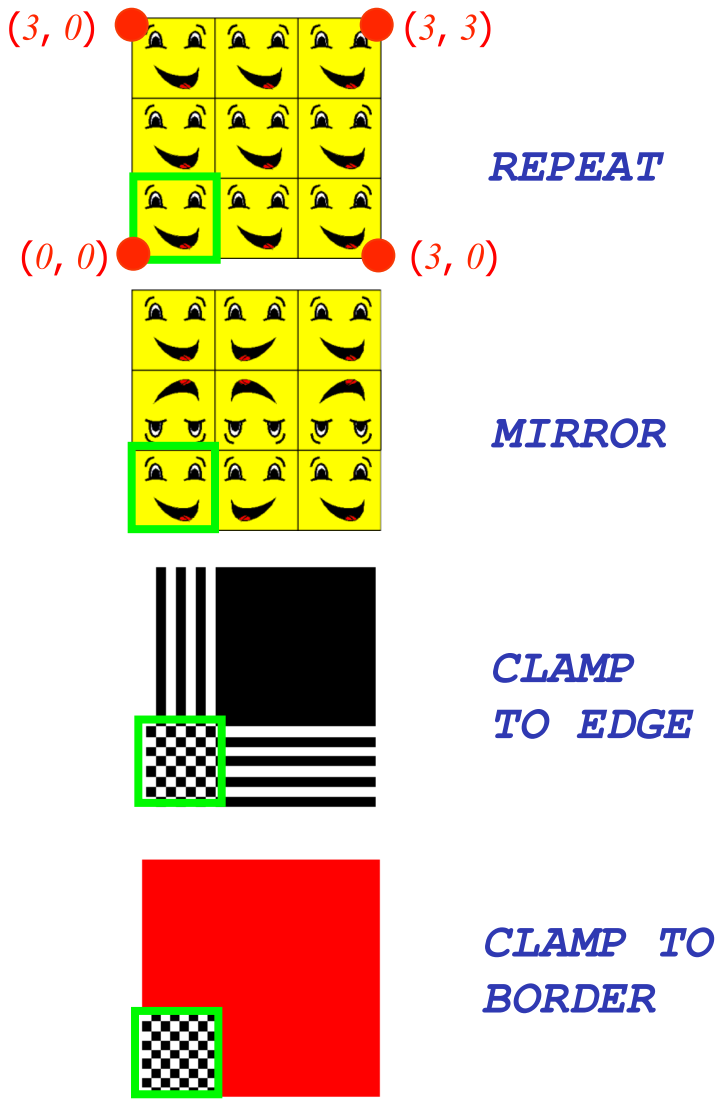


Filtering in Summary

- Zoom-in calls for mag filter
- Zoom-out calls for min filter
- More advanced filters require more time/computation but produce better results
- Mipmapping is an advanced min filter
- Caution: requesting mipmapping without pre-defining mipmaps will turn off texturing; (see Filtering in OpenGL)

Wrapping Modes

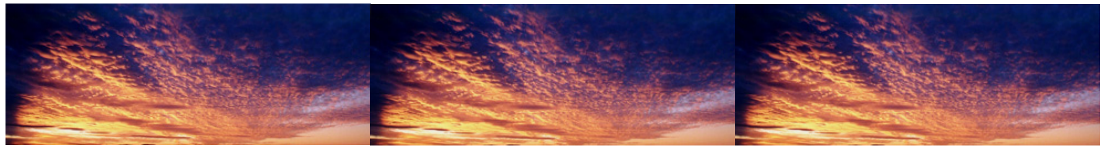
- Can assign texture coords outside of [0,1] and have them either *clamp* or *repeat* the texture map
- **Wrapping modes:**
  - **repeat**: start entire texture over
    - Repeat Issue: making the texture borders match-up
  - **mirror**: flip copy of texture in each direction
    - get continuity of pattern
  - **clamp to edge**: extend texture edge pixels
  - **clamp to border**: surround with border color



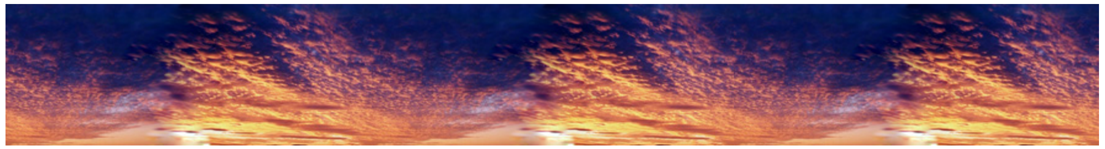
courtesy of Microsoft

Wrapping modes:
Repetitive texture tiling

- A texture can also be repeatedly tiled across the surface by repeating the (s,t) parameterization over the surface
- But, best results are obtained when the texture is seamlessly tilable
  - This means that the right side of the texture joins seamlessly with the left side (same with the bottom and top)
- Seams will appear for most textures when tiled:



- But, we can edit or re-synthesize textures to be seamlessly repeatable (*this is another topic onto itself*):



Texturing in OpenGL

- Procedure
- Example

Texturing in OpenGL

- Procedure: Texture Mapping
  - uploading of the texture to the video memory
  - set up texture parameters
  - enable texturing
  - the application of the texture onto geometry

Texturing in OpenGL: main steps

1. Create texture and load with
   - `glTexImage()`
   - Three methods:
     - read in an image in a jpg, bmp, ... File
     - generate the texture yourself within application
     - copy image from color buffer
2. define texture parameters as to how texture is applied
   - `glTexParameter*()`
   - wrapping, filtering, etc
3. enable texturing
   - `glEnable(GL_TEXTURE_*D)`
4. assign texture coordinates to vertices
   - the mapping function is left up to you
   - `glTexCoord*(s,t);`
   - `glVertex*(x,y,z);`

Step 1: Specifying Texturing Image

- Define a texture image as an array of *texels* (texture elements) in CPU memory:

```
    Glubyte myTexture[width][height][3];
```

- Each RGB value is specified to be an unsigned byte, between 0 and 255

- For example, a blue color would be (0, 0, 255)

Warning: *this [3] seems to be missing in the textbook*

Step 1: Defining an Image as a Texture

- Call `glTexImage2D`. It uploads the texture to the video memory where it will be ready for us to use in our programs.

- `void glTexImage2D(target,level,components,w,h,border,format,type,texture);`
  Parameters:

☐	`target`	: type of texture, e.g. **`GL_TEXTURE_2D`**
☐	`level`	: used for mipmapping = 0 (discussed later)
☐	`components`	: elements per texel (for RGB)
☐	`w, h`	: width and height of texture in pixels
☐	`border`	: used for smoothing = 0 (don't worry about this)
☐	`format`	: **`texel`** format e.g. **`GL_RGB`**
☐	`type`	: rgb component format e.g. **`GL_UNSIGNED_BYTE`**
☐	`texture`	: pointer to the texture array

- Example, set the current texture:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, texture);
```

Step 1 (*alternative*): Generating a Random Texture

- Here is a function for a random texture:

```
GLubyte texture[64][64][3];
int u, v;

for(u=0; u<64; u++)
{
    for(v=0; v<64; v++)
    {
      texture[u][v][0] = (GLubyte)(255 * rand()/RAND_MAX);
      texture[u][v][1] = (GLubyte)(255 * rand()/RAND_MAX);
      texture[u][v][2] = (GLubyte)(255 * rand()/RAND_MAX);
    }
}
```

Step 2: Specifying Texture Parameters

- OpenGL has a variety of parameters that determine how textures are applied:
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
- The `glTexParameter()` function is a crucial part of OpenGL texture mapping, this function determines the behavior and appearance of textures when they are rendered.
- Take note that each texture uploaded can have its own separate properties, texture properties are not global.

Target	Specifies the target texture
GL_TEXTURE_1D	One dimensional texturing.
GL_TEXTURE_2D	Two dimensional texturing.

Step 2: glTexParameter()

Texture Parameter	Accepted values	Description
GL_TEXTURE_MIN_FILTER	GL_NEAREST , GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR and GL_LINEAR_MIPMAP_LINEAR	The texture minification function is used when a single screen pixel maps to more than one texel, this means the texture must be shrunk in size. Default setting is GL_NEAREST_MIPMAP_LINEAR.
GL_TEXTURE_MAG_FILTER	GL_NEAREST or GL_LINEAR	The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texel, this means the texture must be magnified. Default setting is GL_LINEAR.
GL_TEXTURE_WRAP_S	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the s texture coordinate. Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_WRAP_T	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the t texture coordinate. Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_BORDER_COLOR	Any four values in the [0, 1] range	Sets the border color for the texture, if border is present. Default setting is (0, 0, 0, 0).
GL_TEXTURE_PRIORITY	[0, 1]	Specifies the residence priority of the texture, use to prevent OpenGL from swapping textures out of video memory. Can be set to values in the [0, 1] range. See *glPrioritizeTextures()* for more information or this article on Gamasutra.

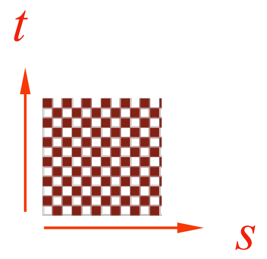
Step 2: glTexParameter()

Parameter Value	Description
GL_CLAMP	Clamps the texture coordinate in the [0,1] range.
GL_REPEAT	Ignores the integer portion of the texture coordinate, only the fractional part is used, which creates a repeating pattern. A texture coordinate of 3.0 would cause the texture to tile 3 times when rendered .
GL_NEAREST	Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured. Use this parameter if you would like your texture to appear sharp when rendered .
GL_LINEAR	Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of GL_TEXTURE_WRAP_S and GL_TEXTURE_WRAP_T, and on the exact mapping. Use this parameter if you would like your texture to appear blurred when rendered .
GL_NEAREST_MIPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value.
GL_LINEAR_MIPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.
GL_NEAREST_MIPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.
GL_LINEAR_MIPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

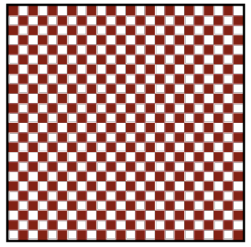
Step 2: Wrapping Modes

- Clamping : if $s,t > 1$ use color at 1, if $s,t < 0$ use color at 0
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);`

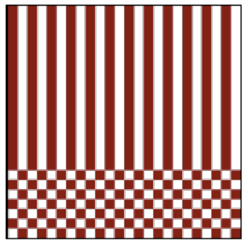
- Repeating : use s,t modulo 1
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`



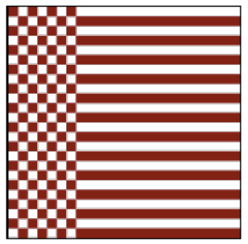
texture



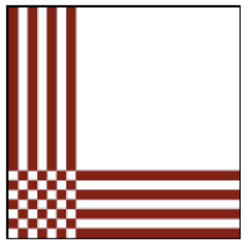
Wrap S : GL_REPEAT
Wrap T : GL_REPEAT



Wrap S : GL_REPEAT
Wrap T : GL_CLAMP



Wrap S : GL_CLAMP
Wrap T : GL_REPEAT



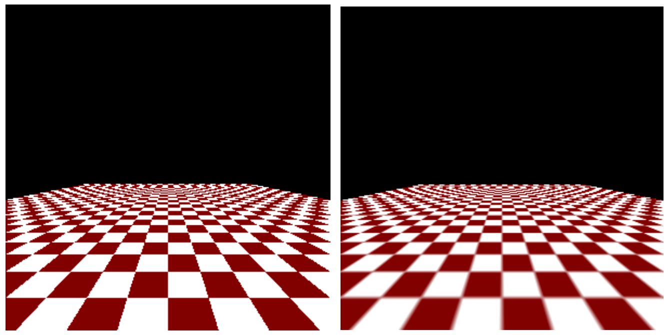
Wrap S : GL_CLAMP
Wrap T : GL_CLAMP

Step 2:
Filter Modes

- Minification and magnification

Step 2: Mipmapping

- …



Step 3: Enable Texturing

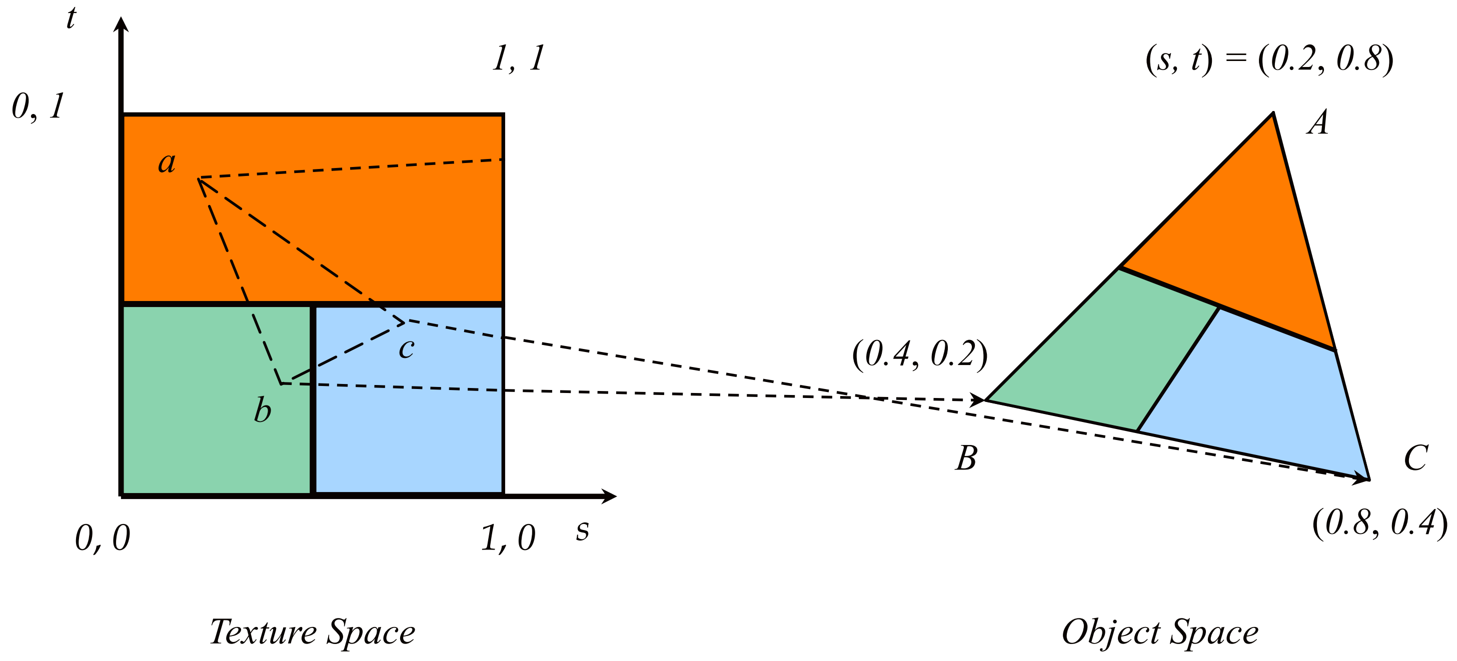
- To enable/disable just call:
  - `glEnable(GL_TEXTURE_2D)`
  - `glDisable(GL_TEXTURE_2D);`

- What does texture mapping affect?
  - the current shading color of a pixel (after lighting) is multiplied by the corresponding texture color

- So, if the object is a near white color (0.8, 0.8, 0.8) at some point and the current texture color at that point is red (1, 0, 0), then when multiplied, it produces (0.8, 0, 0)

Step 4: Mapping a Texture

- Assign the texture coordinates
- `glTexCoord*()` is specified at each vertex

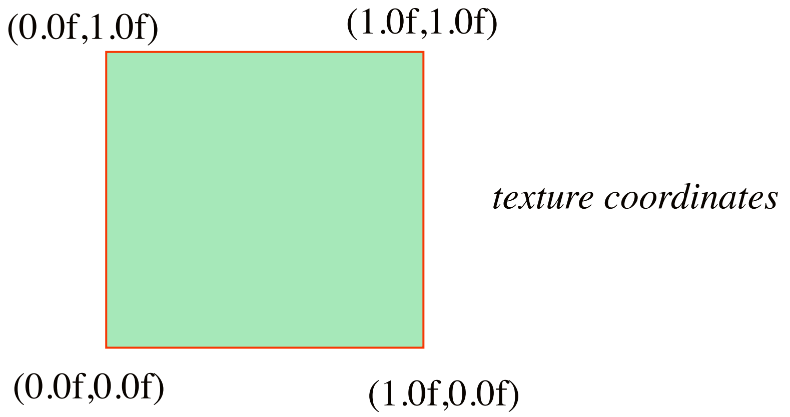


Step 4: Mapping a Texture: *example*

- How to texture a quad? The following code assumes that texturing has been enabled and that there has been a texture uploaded with the id of 13.

```
glBindTexture (GL_TEXTURE_2D, 13);

glBegin (GL_QUADS);
    glTexCoord2f (0.0, 0.0); glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (1.0, 0.0); glVertex3f (10.0, 0.0, 0.0);
    glTexCoord2f (1.0, 1.0); glVertex3f (10.0, 10.0, 0.0);
    glTexCoord2f (0.0, 1.0); glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```



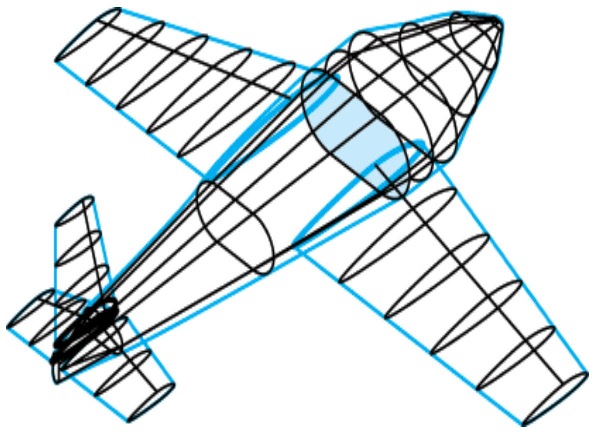
Texturing Mapping in OpenGL: summing up

- We have seen how a 2D texture image can be mapped to an object, at the rendering stage
  - for a polygon, we pin texture to vertices and interpolate (correctly!) at scan conversion time

- The texture value is used to modify the colour that would otherwise be drawn
  - options include replacing completely, or modulating (e.g. by multiplying shaded value with texture value)

What about complex 3D objects?

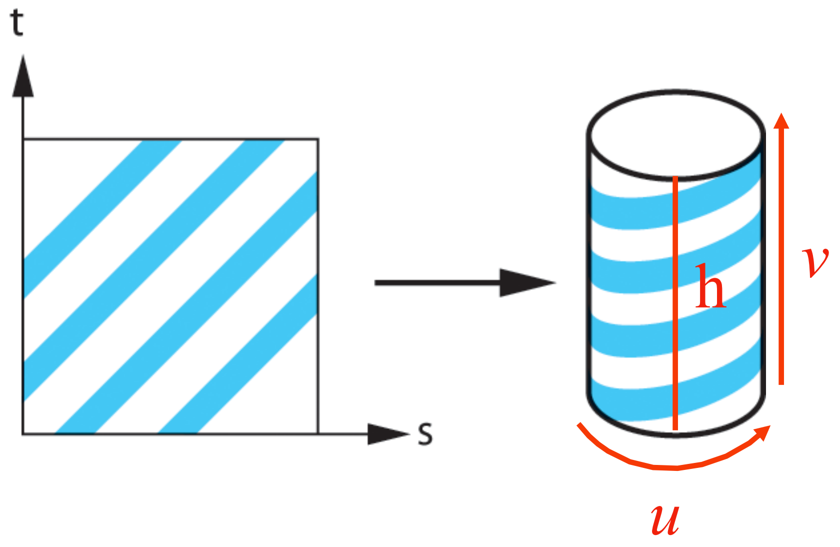
- It is easy to set texture coordinate for a single 2D polygon, but can be difficult to set the texture coordinates for complex 3D regions or objects.
- Besides:
  - in rendering based on pixel-to-pixel approach, the inverse mapping from screen coordinates to texture coordinates is needed
  - because of shading, mapping areas-to-areas and not point-to-point is required, which causes antialising problems, moire patterns etc.
- ***Two-Stage Mapping***: An automatic solution to the mapping problem is to first map the texture to a simple intermediate surface **then** map the simple intermediate surface to the target surface



Automatic: cylindrical mapping

- Example: first map to a cylinder
- Like wrapping a label around a can of soup

- Convert rectangular coordinates (x, y, z) to cylindrical (r, , h), use only (h, μ) to index texture image



Automatic: cylindrical mapping

- Example: first map to a cylinder
- Like wrapping a label around a can of soup

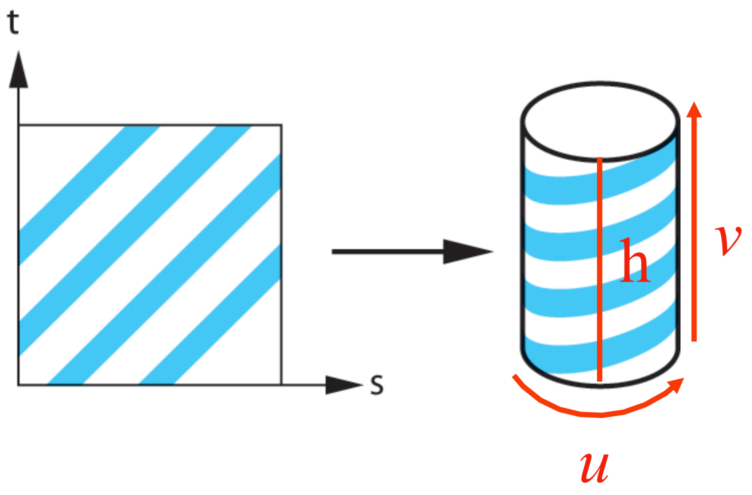
- Convert rectangular coordinates (x, y, z) to cylindrical (r, θ, h), use only (h, θ) to index texture image

- Parametric cylinder:

$$x = r \cos(2\pi\, u)$$
$$y = v/\mathrm{h}$$
$$z = r \sin(2\pi\, u)$$

- Maps rectangle in u, v space to cylinder of radius r and height h in world coordinates:

$$s = u$$
$$t = v$$

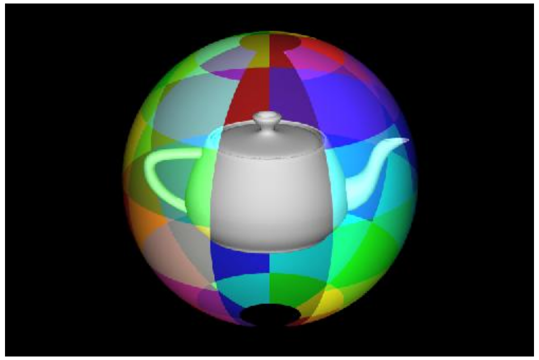


Automatic: spherical mapping

- Example: first map to a sphere
- Convert rectangular coordinates (x,y,z) to spherical (θ,ϕ)
- Parametric sphere:

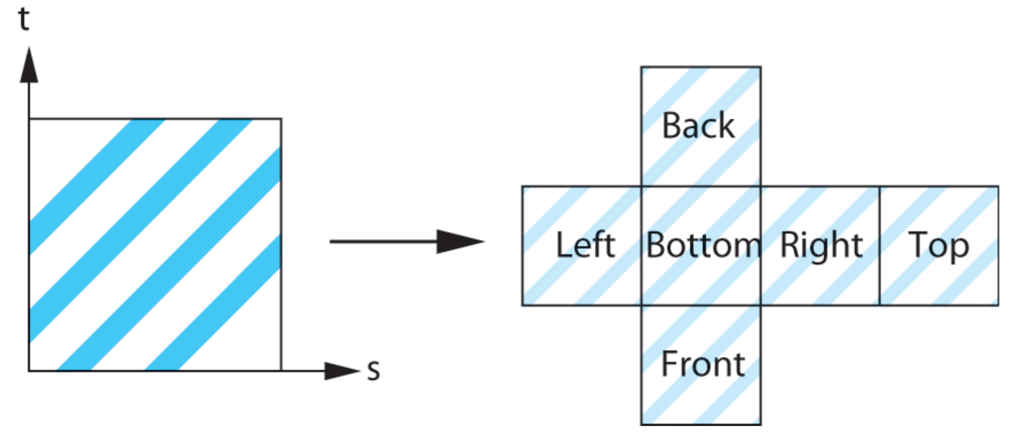
$$x = r \cos(2\pi u)$$
$$y = r \sin(2\pi u) \cos(2\pi v)$$
$$z = r \sin(2\pi u) \sin(2\pi v)$$

- For example: paste a world map onto a sphere to model the earth. But in the case of the sphere there is distortion at the poles (north and south)



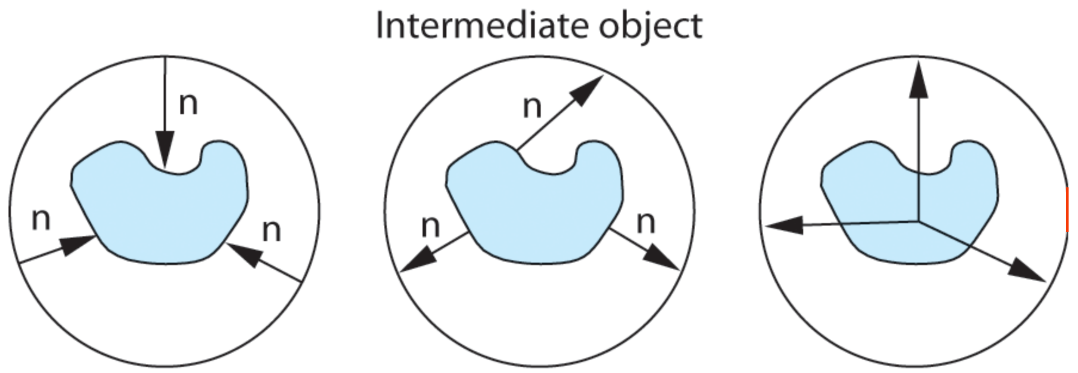
Automatic: box mapping

- Example: first map to a box



Automatic: stage-two mapping

- Now, we still need to map from an intermediate object (sphere, cylinder, or box) to the target object
    1. Intersect the normals from intermediate surface to target surface
    2. Intersect the normals from target surface to intermediate surface
    3. Intersect vectors from center of target surface to intermediate

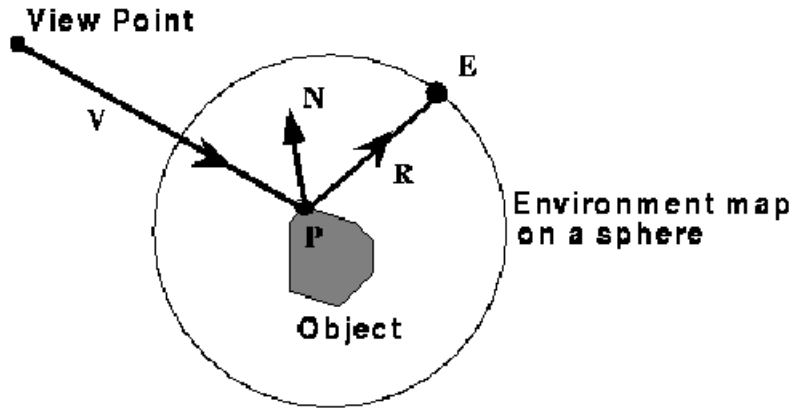


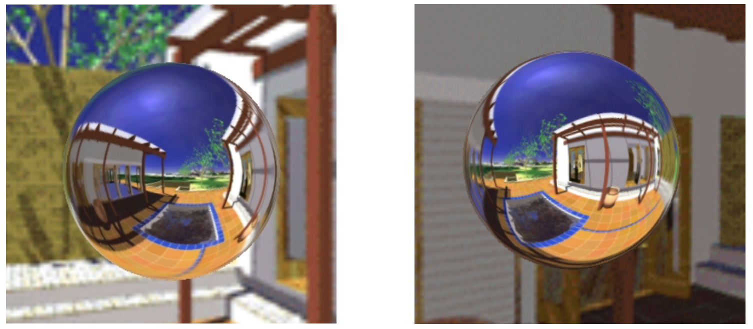
Mapping Techniques

- Texture Mapping
- Environmental Mapping
- Bump Mapping
- Light Mapping

Environmental Maps

- Use texture to represent reflected color
  - Texture indexed by reflection vector
  - Approximation works when objects are far away from the reflective object
- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: $\boldsymbol{R}=2(\boldsymbol{N}\cdot\boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$
- What is in the map?

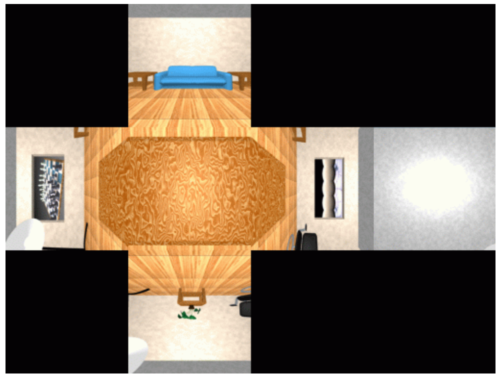




spherical map

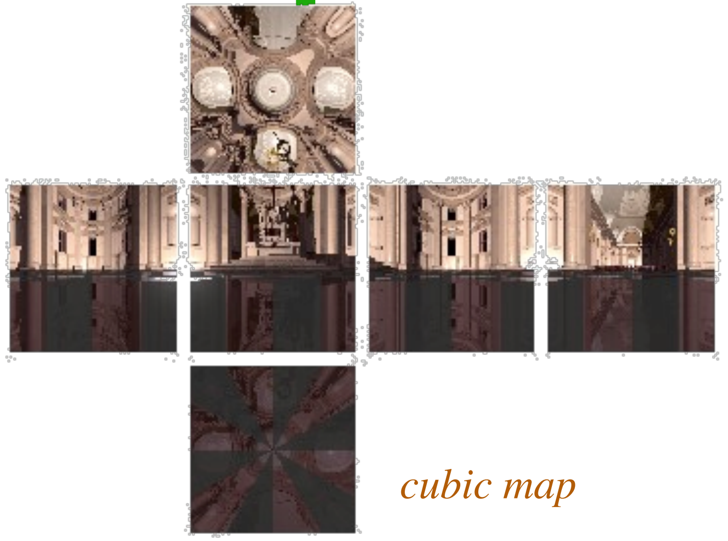
Approximations Made

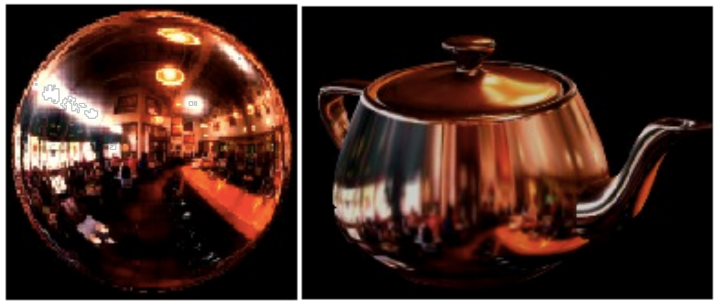
- The map should contain a view of the world with the point of interest on the object as the eye
  - We can’t store a separate map for each point, so one map is used with the eye at the center of the object
  - Introduces distortions in the reflection, but the eye doesn’t notice
  - Distortions are minimized for a small object in a large room
- The object will not reflect itself
- The mapping can be computed at each pixel, or only at the vertices


cubic map

Types of Environment Maps

- The environment map may take one of several forms:
  - Cubic mapping
    - Easy to produce with rendering system
    - Possible to produce from phtographs
    - “uniform” resolution
    - Simple texture coordinates calculation
  - Spherical mapping (two variants)
    - Spatially variant resolution
  - Parabolic mapping
- Describes the shape of the surface on which the map “resides”
- Determines how the map is generated and how it is indexed
- What are some of the issues in choosing the map?


cubic map

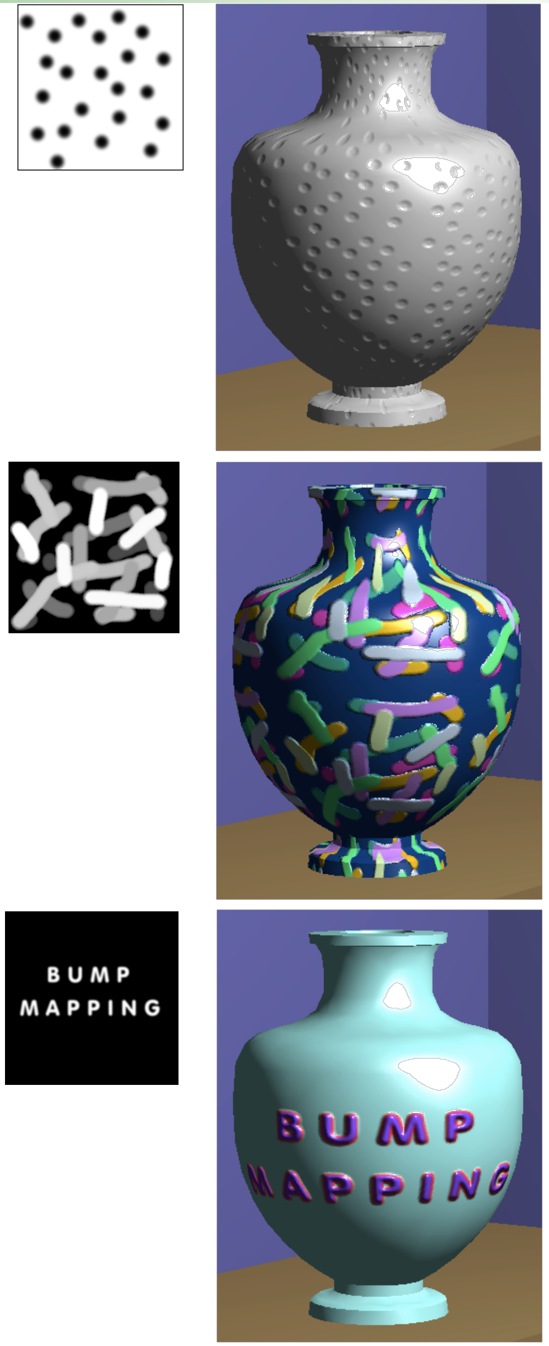

spherical maps

Mapping Techniques

- Texture Mapping
- Environmental Mapping
- Bump Mapping
- Light Mapping

Bump Mapping

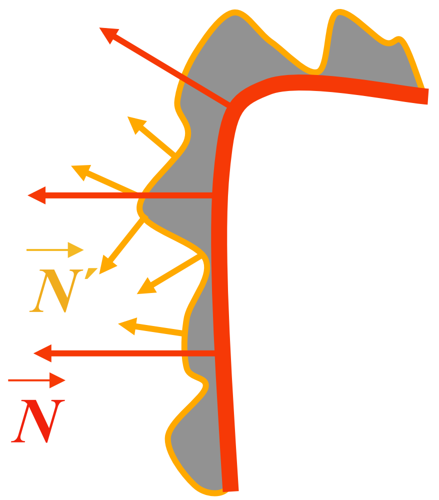
- This is another texturing technique
- Aims to simulate a dimpled or wrinkled surface
    - for example, surface of an orange
- Like Gouraud and Phong shading, it is a trick
    - surface stays the same
    - but the true normal is perturbed, or jittered, to give the illusion of surface ‘bumps’



Bump Mapping: how does it work?

- To create a <u>bump-like effect</u>, we use texture to perturb normals
- Many textures are the result of small perturbations in the surface geometry
- Modeling these changes would result in an explosion in the number of geometric primitives.
- Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.
- We can model this as deviations from some base surface.
- The question is then how these deviations change the lighting.

<u>Idea:</u> small normal deviations



$S(u,v) + B(u,v) = S'(u,v)$

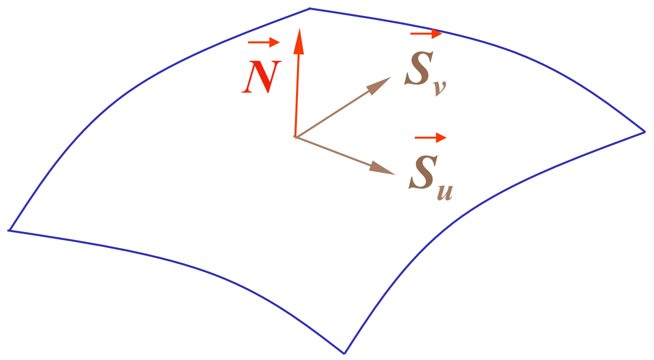
original surface *bump map* *bumped surface*

<u>where B=f(u,v) is a height field defined as a 2D function over S</u>

Bump Mapping: step 1 and 2

- Step 1: Putting everything into the same coordinate frame as B(u,v).
  - x(u,v), y(u,v), z(u,v) – this is given for parametric surfaces, but easy to derive for other analytical surfaces.
  - Or $\boldsymbol{S}(u,v)$
- Step 2: Define the tangent plane to the surface at a point (u,v) by using the two vectors $\vec{S}_u$ and $\vec{S}_v$.
  - The normal is then given by:

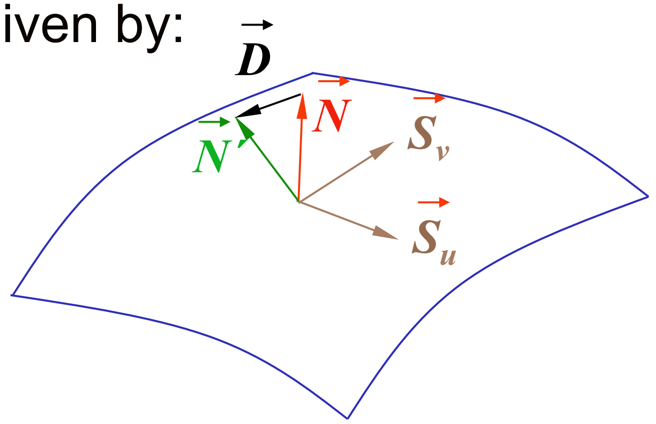
$$\vec{N} = \vec{S}_u \times \vec{S}_v$$



Bump Mapping: step 3, 4, and 5

- Step 3: The new surface positions are then given by:
  - $S'(u,v) = S(u,v) + B(u,v)\,\vec{N}$
  - where, $\vec{N} = \vec{N} / |\vec{N}|$
- Step 4: Differentiating leads to:
  - $\vec{S}'_{\mathbf{u}} = \vec{S}_{\mathbf{u}} + B_{\mathrm{u}}\,\vec{N} + B(\vec{N})_{\mathrm{u}} \approx \vec{S}'_{\mathbf{u}} = \vec{S}_{\mathbf{u}} + B_{\mathrm{u}}\,\vec{N}$
  - $\vec{S}'_{\mathbf{v}} = \vec{S}_{\mathbf{v}} + B_{\mathrm{v}}\,\vec{N} + B(\vec{N})_{\mathrm{v}} \approx \vec{S}'_{\mathbf{v}} = \vec{S}_{\mathbf{v}} + B_{\mathrm{v}}\,\vec{N}$

  since B is small, as it is the case because it is a small height perturbation.
- Step 5: This leads to a new normal:
  - $\vec{N}'(u,v) = \vec{S}_{\mathbf{u}} \times \vec{S}_{\mathbf{v}} - B_{\mathrm{u}}(\vec{N} \times \vec{S}_{\mathbf{v}}) + B_{\mathrm{v}}(\vec{N} \times \vec{S}_{\mathbf{u}}) + B_{\mathrm{u}}\,B_{\mathrm{v}}(\vec{N} \times \vec{N})$
    - $= \vec{N} - B_{\mathrm{u}}(\vec{N} \times \vec{S}_{\mathbf{v}}) + B_{\mathrm{v}}(\vec{N} \times \vec{S}_{\mathbf{u}})$
    - $= \vec{N} + \vec{D}$
  - For efficiency, can store B_{u} and B_{v} in a 2-component texture map:
    - This is commonly called an offset vector map;
    - It is oriented in tangent space, not object space.
  - The cross products are geometry terms only.
  - $\vec{N}'$ will of course need to be normalized after the calculation and before lighting. This floating point square root and division makes it difficult to embed into hardware.

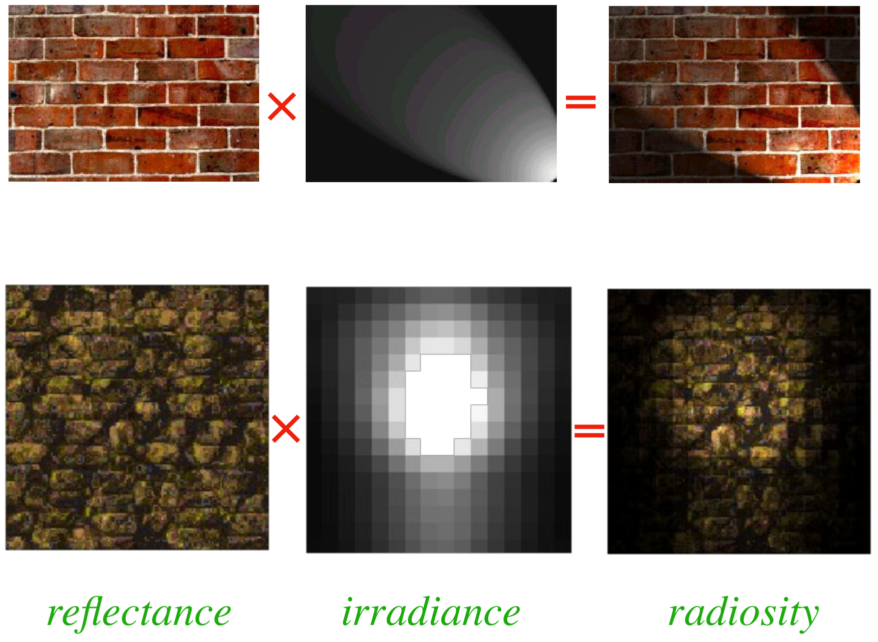


Mapping Techniques

- Texture Mapping
- Environmental Mapping
- Bump Mapping
- Light Mapping

Light Mapping

- Gouraud shading is established technique for rendering but has well known limitations
  - Vertex lighting only works well for small polygons…
  - … but we don't want lots of polygons!
- Solution is to pre-compute some canonical light effects as texture maps
- For example…
- Suppose we want to show effect of a wall light
  - Create wall as a single polygon
  - Apply vertex lighting
  - Apply texture map
  - In a second rendering pass, apply light map to the wall



Light Mapping

- Widely used in games industry
- Latest graphics cards will allow multiple texture maps per pixel