CSE 528: Computer Graphics

Texture Mapping

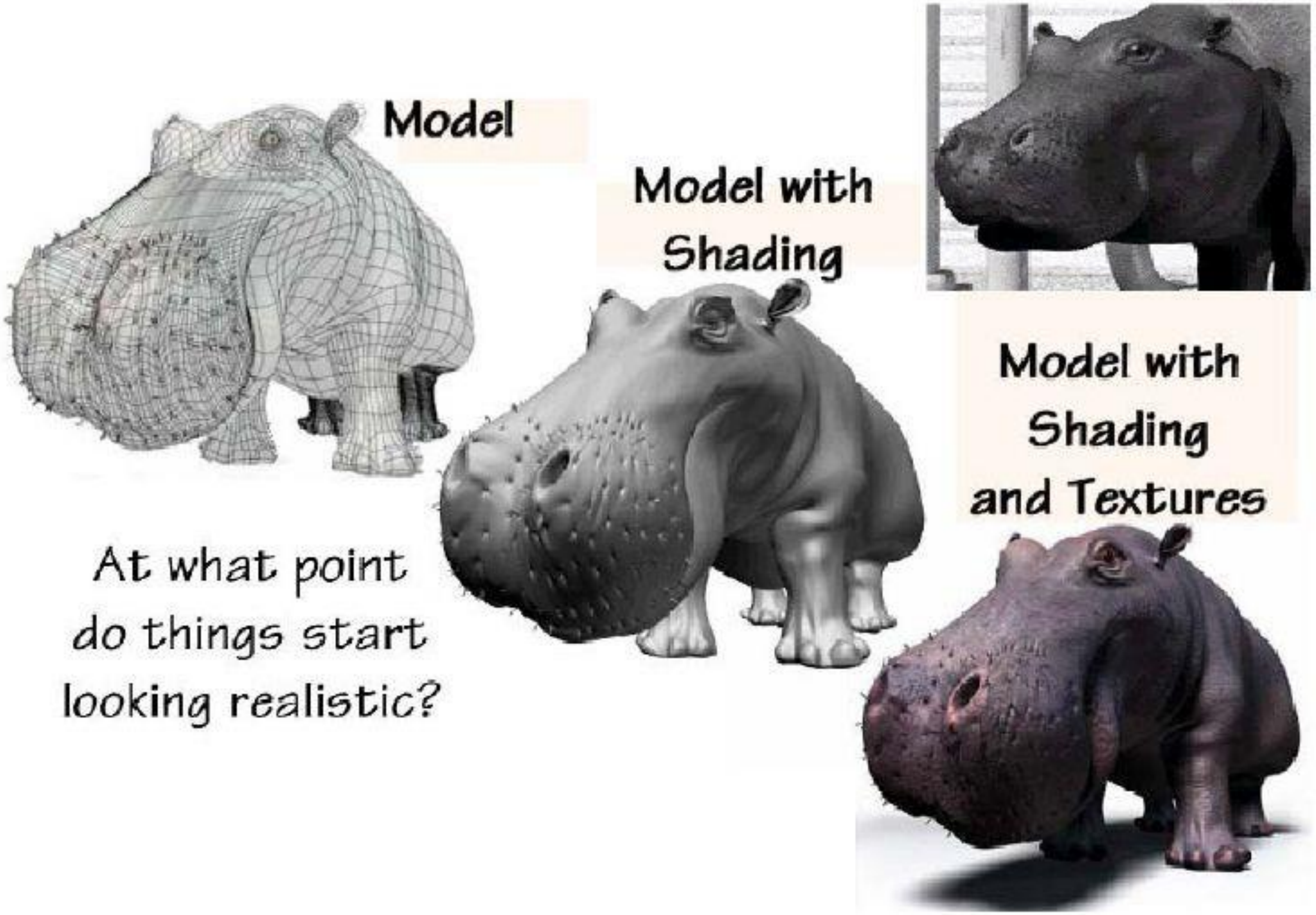
Klaus Mueller

Computer Science Department

Stony Brook University

Some material from courses.csusm.edu/cs535xz/Lectures/texture.ppt

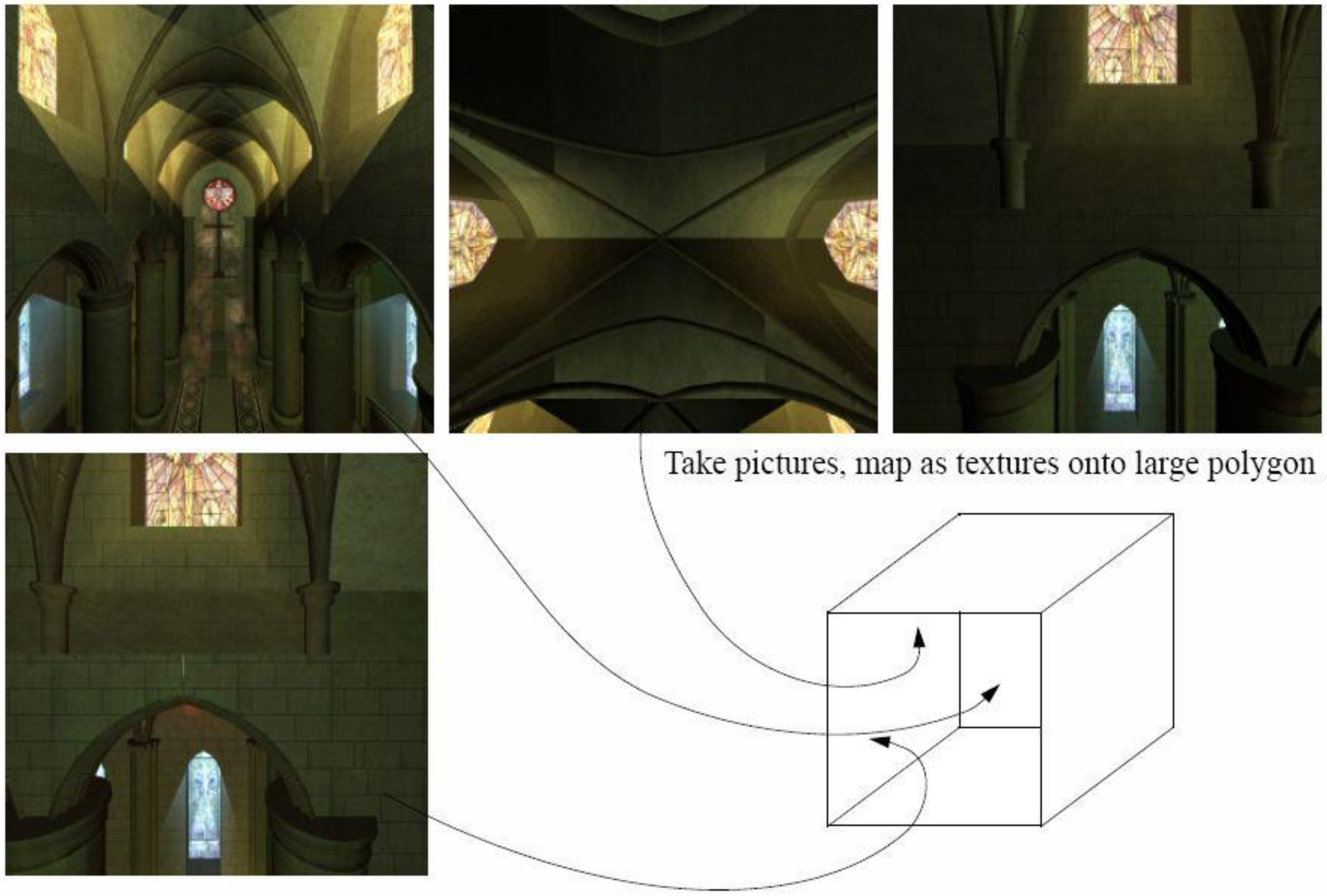
Texture Mapping - Realistic Detail for Boring Polygons



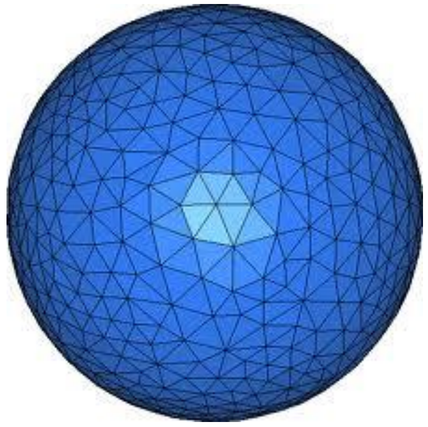
At what point do things start looking realistic?

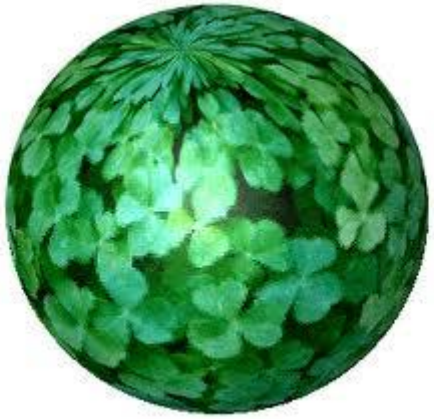
Leonard McMillan

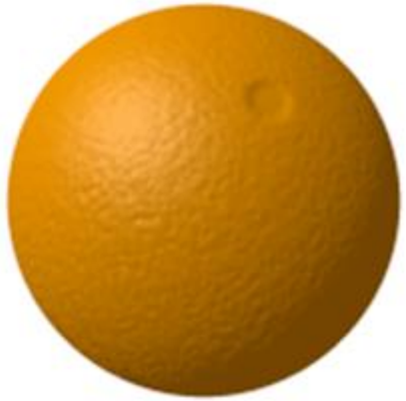
Texture Mapping - Large Walls

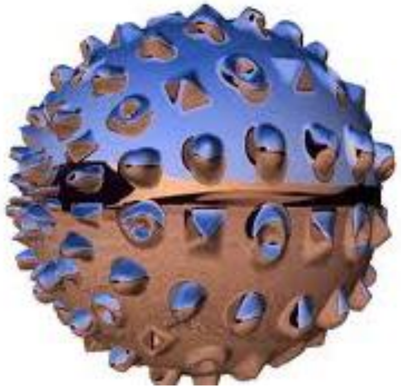


Texture Mapping – Add Detail at Little Cost


shaded polygons


texture mapped


bump mapped


environment mapped

would take lots of polygons to render by shading

Modeling an Orange – by Geometry

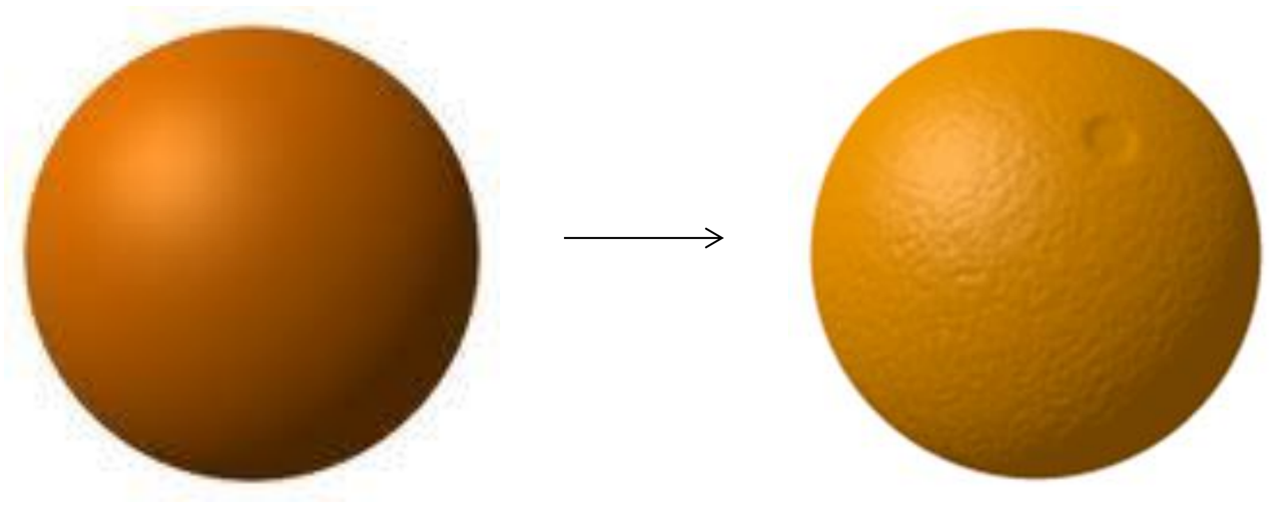
Let’s consider the problem of modeling an orange

Start with an orange-colored sphere

- too simple

Replace sphere with a more complex shape

- does not capture surface characteristics (small dimples)
- takes too many polygons to model all the dimples



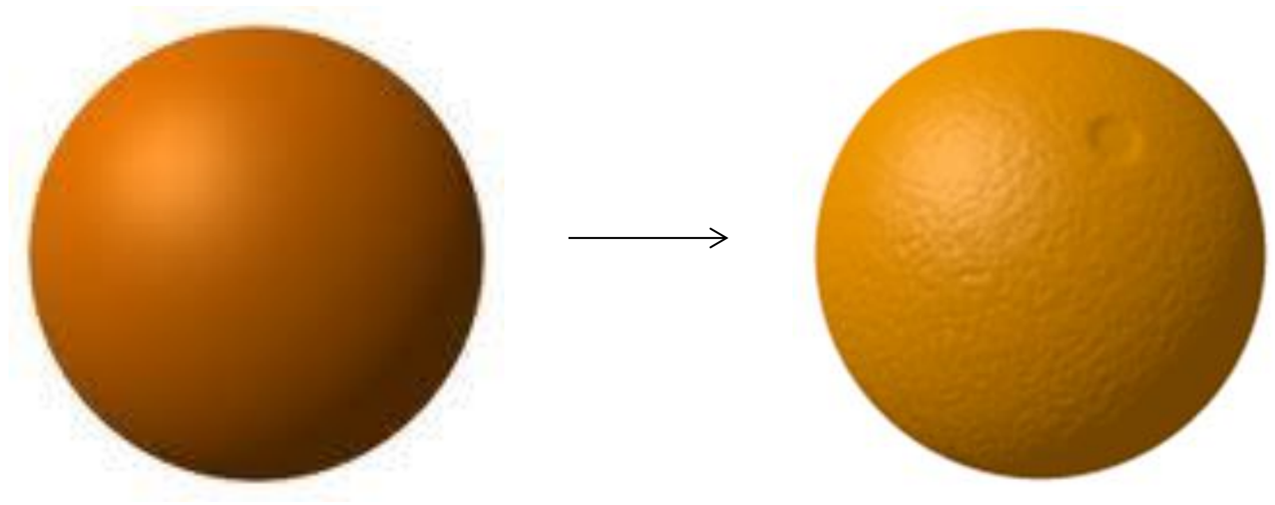
Modeling an Orange – By Image

Take a picture of a real orange, scan it, and “paste” onto simple geometric model

- this process is known as texture mapping

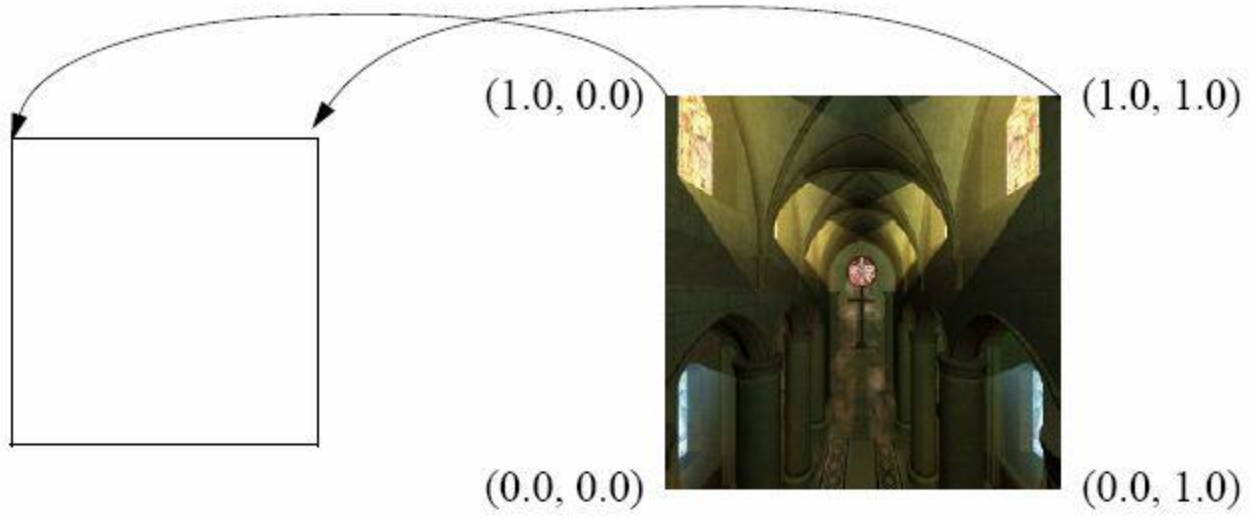
Still might not be sufficient because resulting surface will be smooth

- need to change local shape
- Bump mapping

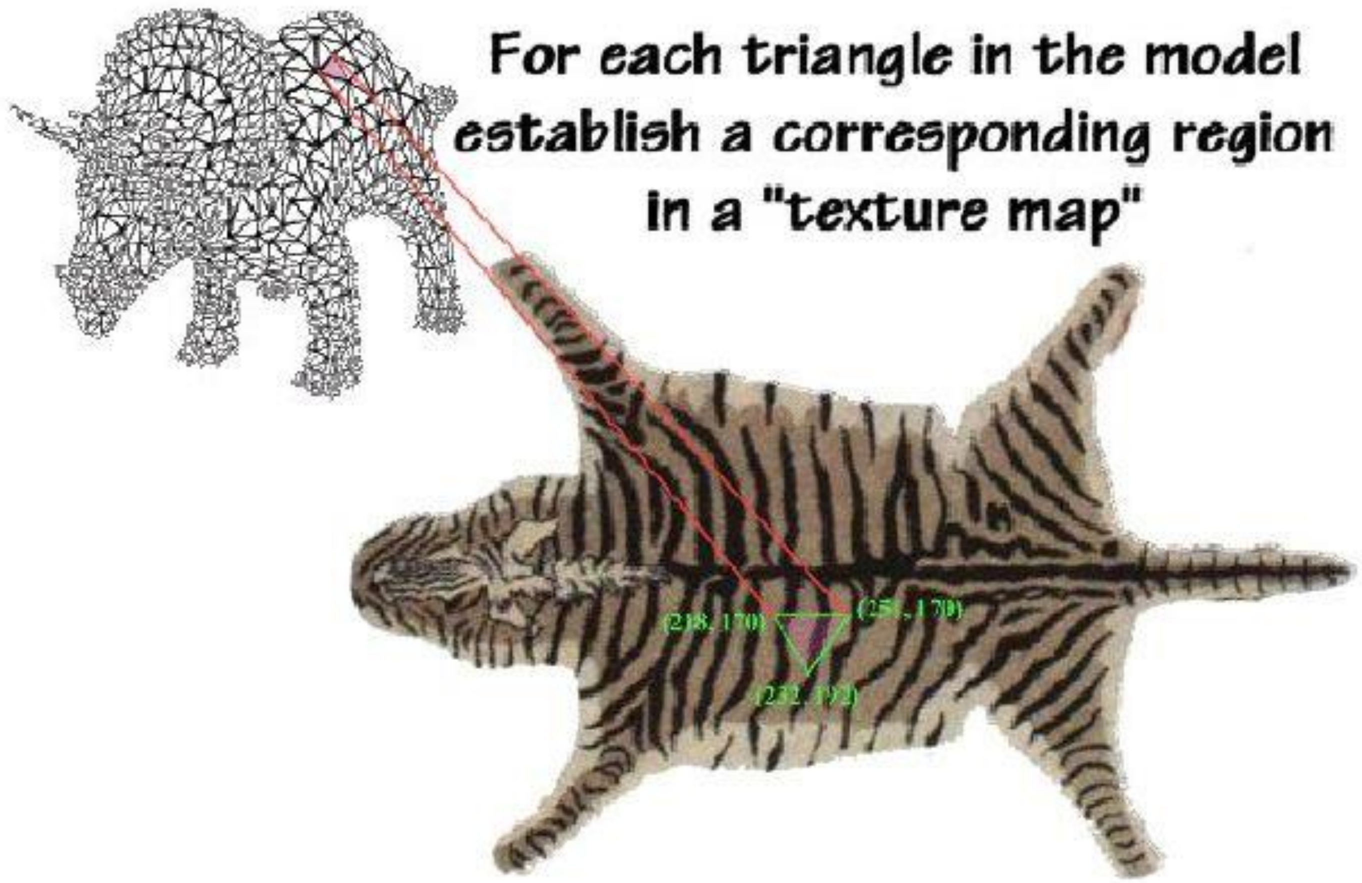


Texture Mapping Large Walls - OpenGL Program

```
glEnable(GL_TEXTURE_2D);
for each polygon
        glBindTexture(textureName);
        glBegin(GL_QUAD);
                glColor3fv(c1);  glVertex3fv(v1);  glTexCoord2D(0.0, 0.0);  /* vertex 1 */
                glColor3fv(c2);  glVertex3fv(v2);  glTexCoord2D(0.0, 1.0); /* vertex 2 */
                glColor3fv(c3);  glVertex3fv(v3);  glTexCoord2D(1.0, 1.0); /* vertex 3 */
                glColor3fv(c4);  glVertex3fv(v4);  glTexCoord2D(1.0, 0.0); /* vertex 4 */
        glEnd();
```



Texture Mapping - Small Facets



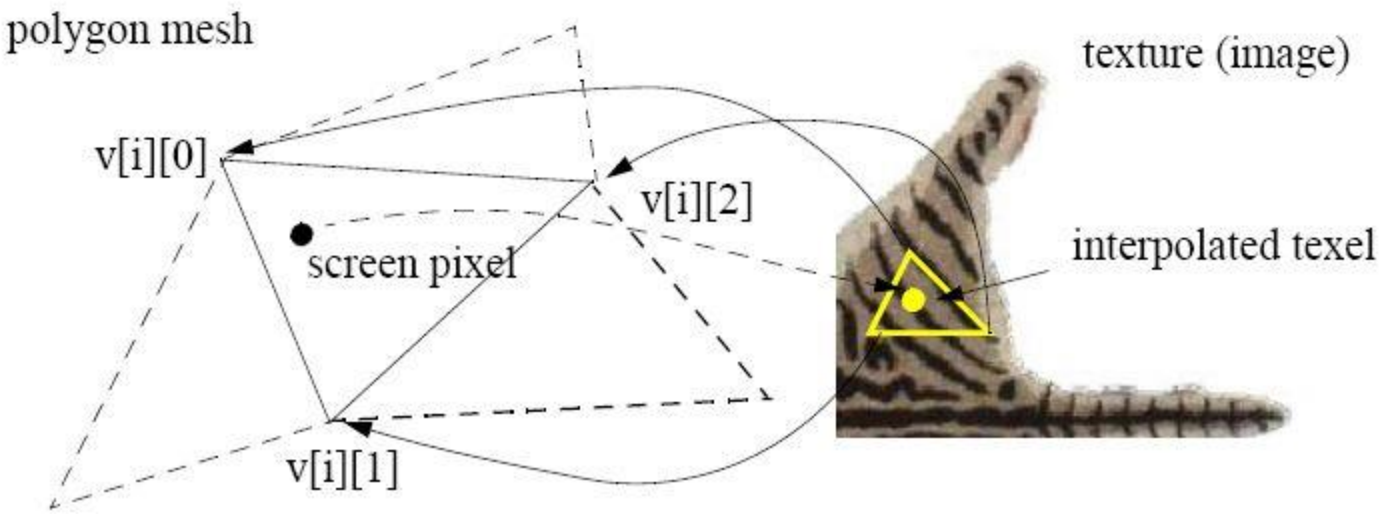
For each triangle in the model establish a corresponding region in a "texture map"

During rasterization interpolate the coordinate indices within the texture map

Leonard McMillan

Texture Mapping Small Facets - OpenGL Program

```
glEnable(GL_TEXTURE_2D);
glBindTexture(textureName);
for each polygon i in the mesh
    glBegin(GL_QUAD);
        glColor3fv(c[i][0]);  glVertex3fv(v[i][0]);  glTexCoord2fv(t[i][0]);  /* vertex 1 */
        glColor3fv(c[i][1]);  glVertex3fv(v[i][1]);  glTexCoord2fv(t[i][1]); /* vertex 2 */
        glColor3fv(c[i][2]);  glVertex3fv(v[i][2]);  glTexCoord2fv(t[i][2]); /* vertex 3 */
    glEnd();
```

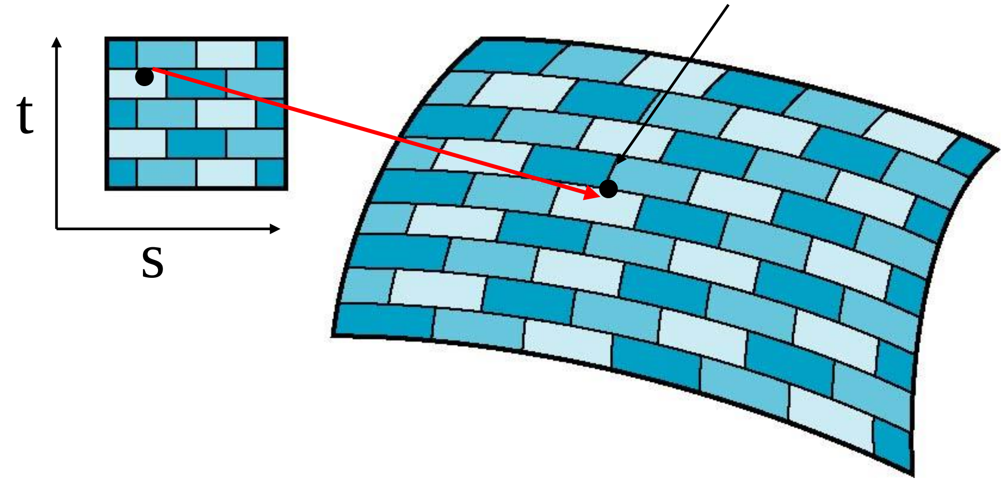


Mapping Function

Need functions

- s = s (x, y, z)
- t = t (x, y, z)

Such functions are difficult to find in general



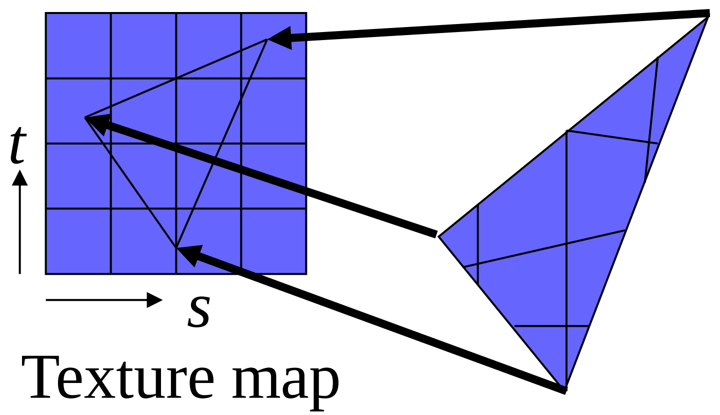
Polygon Texture Interpolation

For polygons, specify *(s,t)* coordinates at vertices

- s = s (x, y, z)
- t = t (x, y, z)

Linearly interpolate the mapping for other points in world space

- straight lines in world space go to straight lines in texture space

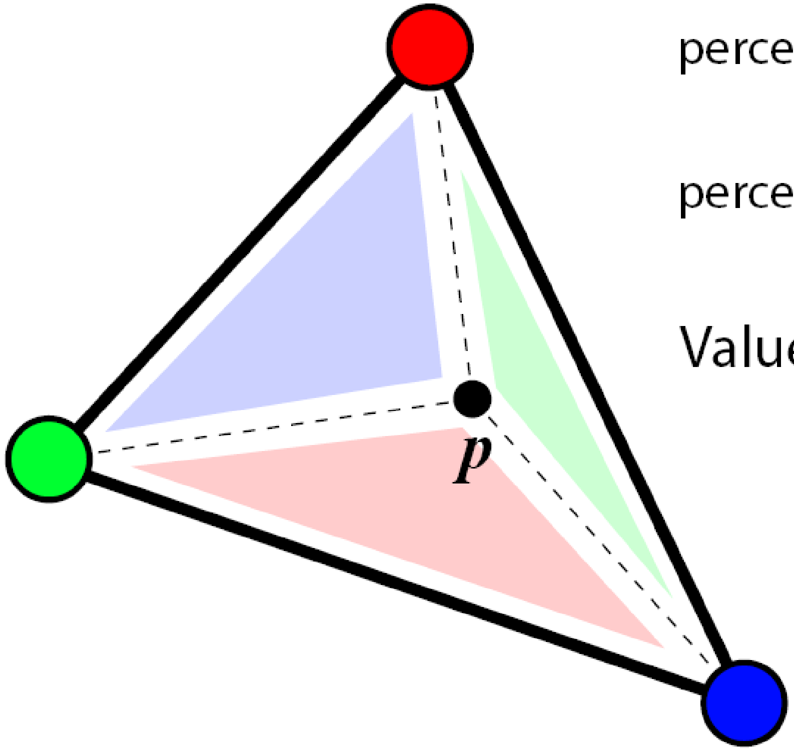


Triangle in world space

Barycentric Interpolation

Similar to linear interpolation

- just now for triangles



percent red = $\dfrac{\text{area of red triangle}}{\text{total area}}$

percent green = $\dfrac{\text{area of green triangle}}{\text{total area}}$

percent blue = $\dfrac{\text{area of blue triangle}}{\text{total area}}$

Value at ***p***:

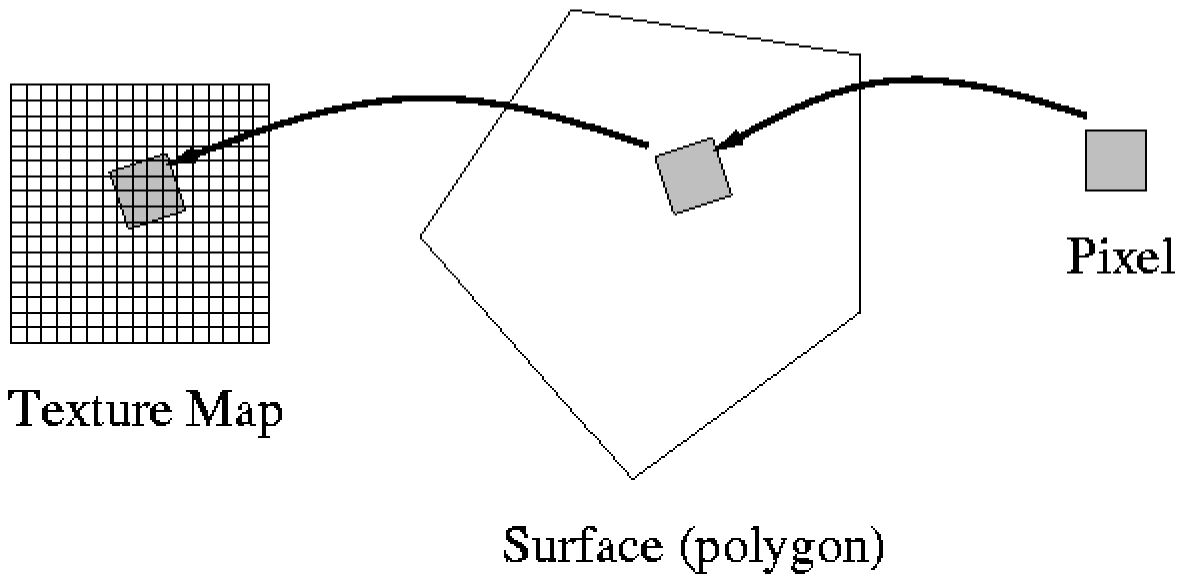
(% red)(value at red) +
(% green)(value at green) +
(% blue)(value at blue)

Computing Pixel Color in Texture Mapping

Associate texture with polygon

Map pixel onto polygon and then into texture map

Use weighted average of covered texture to compute color



2D Texturing

Enable 2D Texturing:

- `glEnable(GL_TEXTURE_2D)`
- texture mapping is disabled by default

Texture objects store texture data. Keep it readily available for usage. Many texture objects can be generated.

Generate identifiers for texture objects first

- `GLuint texids[n];`
- `glGenTextures(n, texids)`
  - `n`: the number of texture objects identifiers to generate
  - `texids`: an array of unsigned integers to hold texture object identifiers

Bind a texture object as the current texture

- `glBindTexture(target, identifier)`
  - `Target`: can be GL_TEXTURE_1D, GL_TEXTURE_2D, or GL_TEXTURE 3D
  - `Identifier`: a texture object identifier
- following texture commands refer to the bound texture, e.g. `glTexImage2D()`

Define Image as a Texture

Load image into a 2D Texture Map

- **`glTexImage2D(GL_TEXTURE_2D, level, internalFormat, width, height, border, format, type, texels)`**
  - **`level`**: level of the texture resolutions. For now = 0
  - **`internalFormat`**: number of color components used for the texture. (integer value from 1 - 4, 3 for RGB)
  - **`width, height`**: size of the texture map (should be power of 2)
  - **`border`**: with (1) or without (0) border
  - **`format`**: format of the texture data, e.g. GL_RGB
  - **`type`**: data type of the texture data, e.g. GL_BYTE
  - **`texels`**: pointer to the actual texture image data

1D and 3D Textures

1D textures can be considered as 2D textures with a height equal to 1. It is often used for drawing color stripes

- `glTexImage1D(GL_TEXTURE_1D, level, components, width, border, format, type, texels)`

3D textures can be considered as a stack of 2D textures. It is often used in visualization applications, e.g. medical imaging.

- `glTexImage3D(GL_TEXTURE_3D, level, components, width, height, depth, border, format, type, texels)`

Created using:
PHOTOSCULPT TEXTURES
See www.photosculpt.net for more

Texture Parameters

OpenGL has a variety of parameters that determine how texture is applied

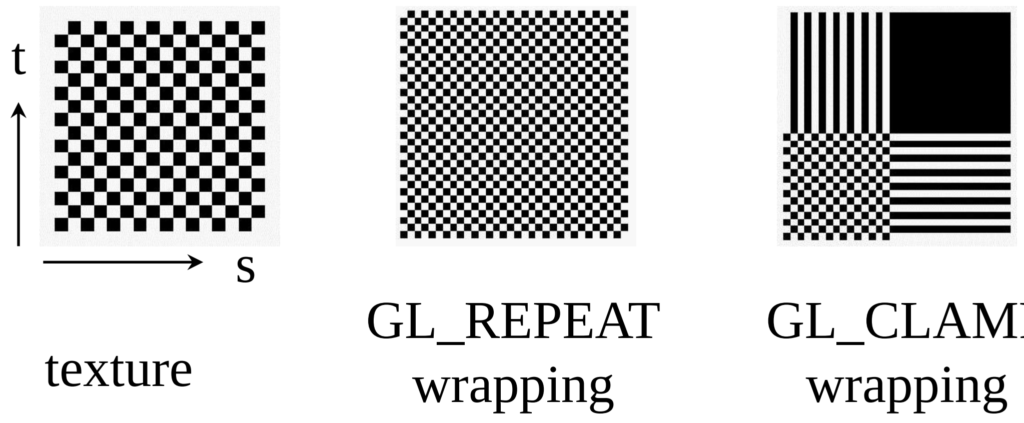
- wrapping parameters determine what happens if s and t are outside the (0,1) range
- environment parameters determine how texture mapping interacts with shading
- filter modes allow us to use area averaging instead of point samples
- Mipmapping allows us to use textures at multiple resolutions

Wrapping Mode

Clamping: if s,t > 1 use 1, if s,t <0 use 0

Wrapping: use s,t modulo 1

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



Texture Functions

You can specify how the texture-map colors are used to modify the pixel colors

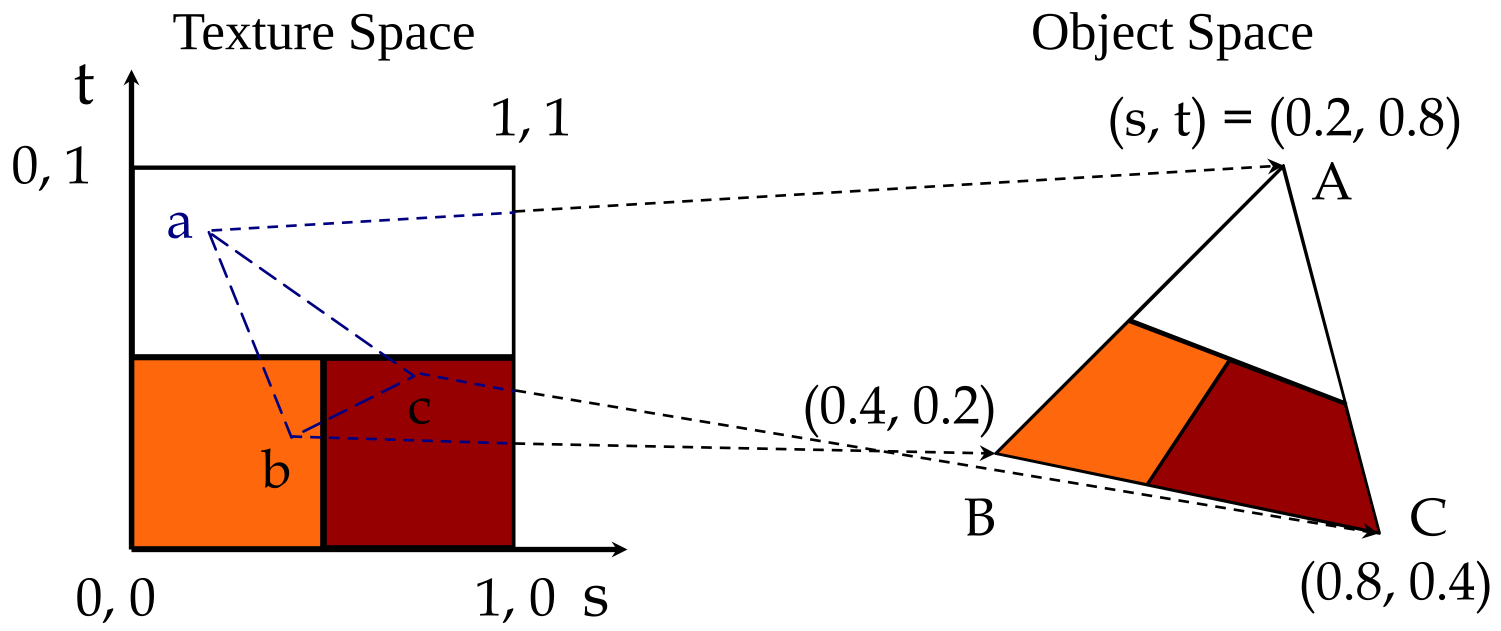
- `glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, mode);`
- **`mode`** values:
  - `GL_REPLACE:` replace pixel color with texture color
  - `GL_DECAL`: replace pixel color with texture color (for GL_RGB texture)
  - `GL_BLEND:` $C = C_f(1-C_t) + C_cC_t$,
    - C_f is the pixel color, C_t is the texture color, and C_c is some constant color
  - `GL_MODULATE:` $C = C_fC_t$
  - More on OpenGL programming guide

Assign Texture Coordinates

Every point on a surface should have a texture coordinate (s,t) in texture mapping

We often specify texture coordinates to polygon vertices and interpolate texture coordinates with the polygon.

- **`glTexCoord*()`** specified at each vertex



Typical Code

```
glBegin(GL_POLYGON);
  glColor3f(r0, g0, b0); //if no shading used
  glNormal3f(u0, v0, w0); // if shading used
  glTexCoord2f(s0, t0);
  glVertex3f(x0, y0, z0);
  glColor3f(r1, g1, b1);
  glNormal3f(u1, v1, w1);
  glTexCoord2f(s1, t1);
  glVertex3f(x1, y1, z1);
        .
        .
glEnd();
```

Note that we can use vertex arrays to increase efficiency

Texture Filtering

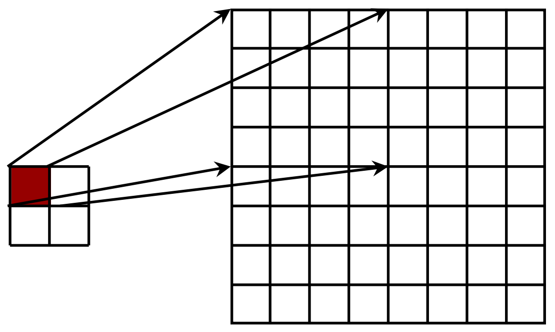
A pixel may be mapped to a small portion of a texel or a collection of texels from the texture map. How to determine the color of the pixel?

Magnification: when a pixel maps to a small portion of a texel

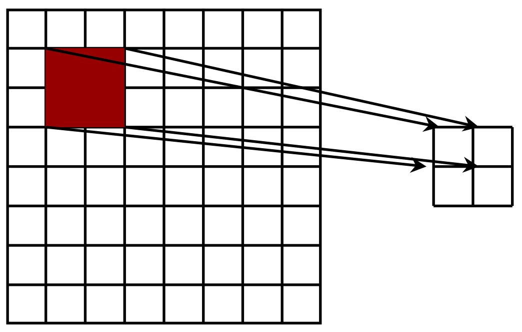
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, type);`
  - `type`: GL_NEAREST or GL_LINEAR

Minification: when a pixel maps to many texels

- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, type);`
  - `type`: GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR, …

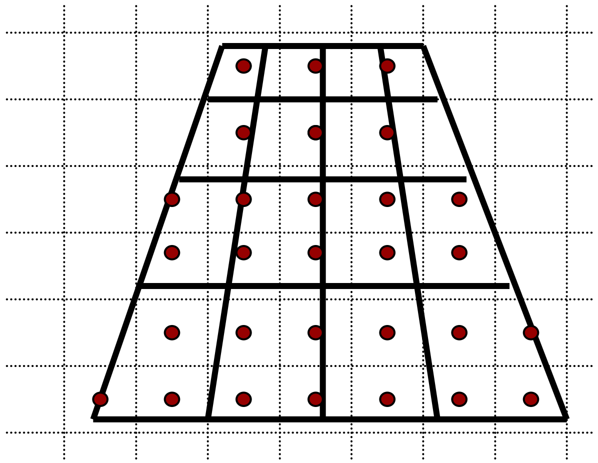


Magnification

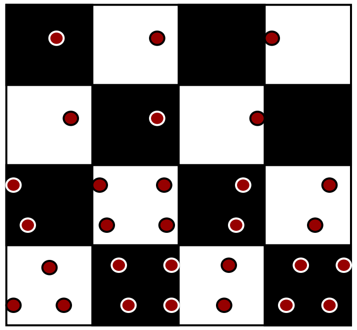


Minification

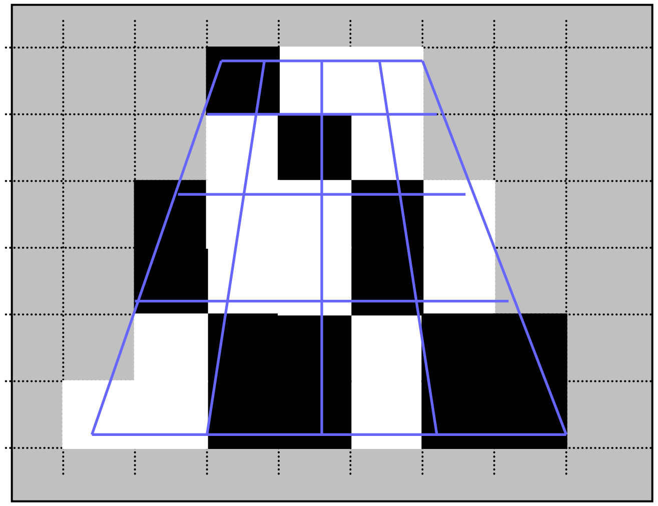
Texture Aliasing



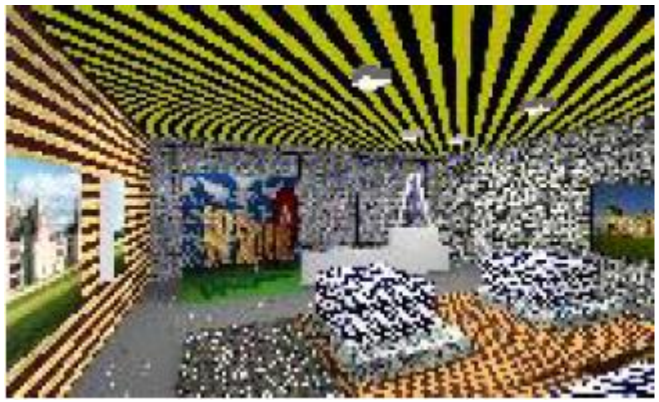
Polygon to pixels



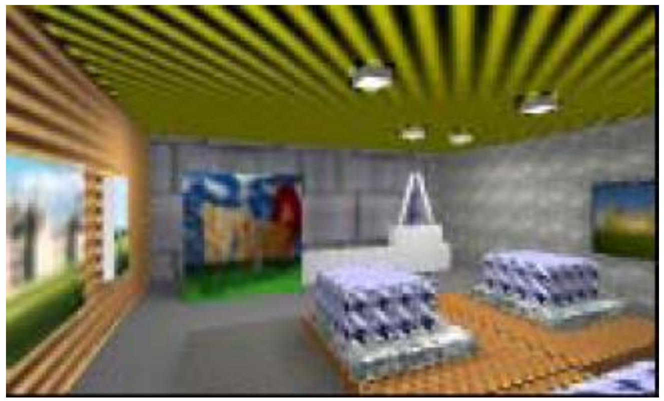
Texture map



Rasterized and textured



Without filtering



Using Mipmaping

Mipmaps

Use different resolution of texture image for rendering different objects

- Level 0: original texture map
- Level 1: half size in width and height

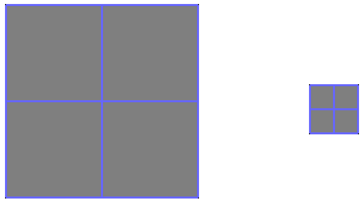
Define mipmaps

- `glTexImage2D( GL_TEXTURE_2D, level, GL_RGB, …);`
  - Where level = 0, 1, 2, ..

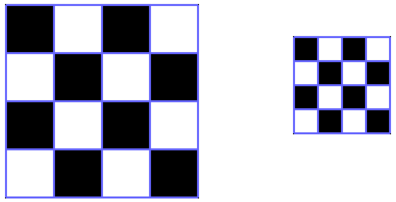
Automatically generate mipmaps

- `gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, format, type, texels);`

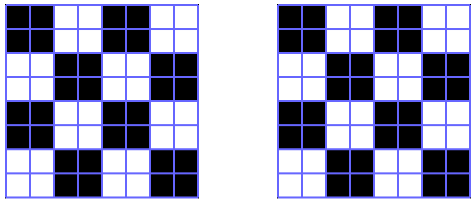
For far objects



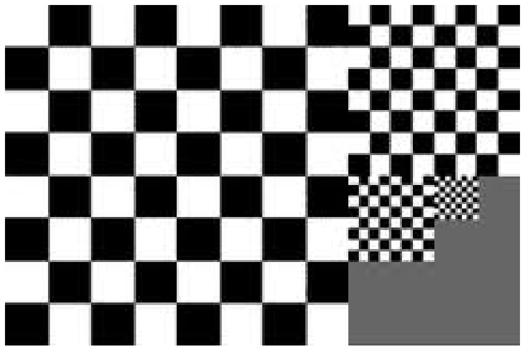
For middle objects



For near objects



mipmap



Mipmap Filters

Mipmap minification filters

- GL_LINEAR_MIPMAP_NEAREST: use the nearest mipmap closest to the polygon resolution, and use linear filtering
- GL_LINEAR_MIPMAP_LINEAR: use linear interpolation between the two mipmaps closest to the polygon resolution, and use GL_LINEAR filtering in each mipmap

Example Code:

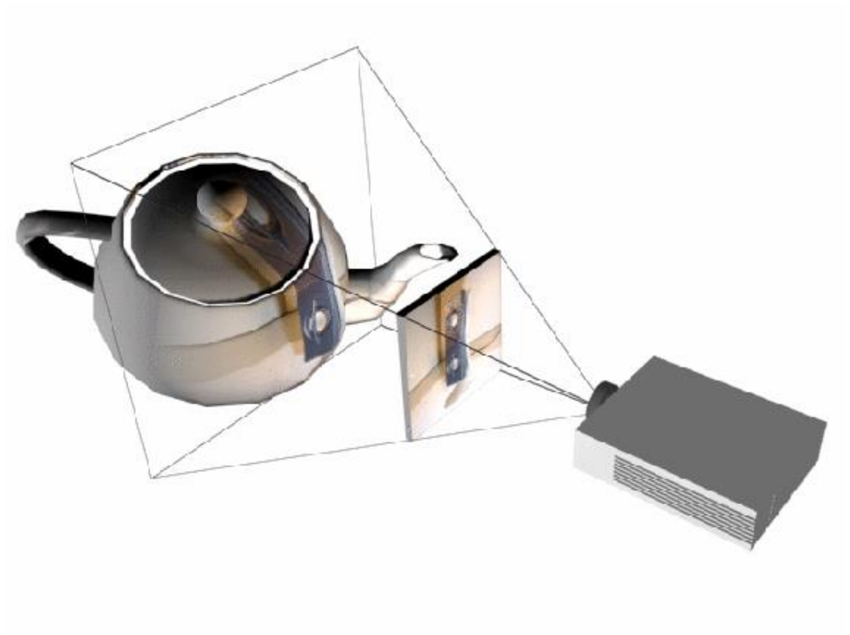
- `gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, 64, 64, GL_RGB, GL_UNSIGNED_BYTE, texImage);`
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);`
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST);`

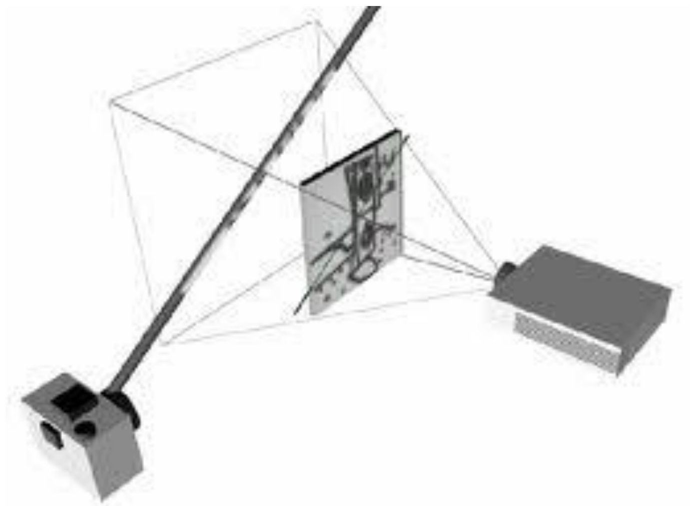
Texture Matrix

Normally, the texture coordinates given at vertices are interpolated and directly used to index the texture

The texture matrix applies a homogeneous transform to the texture coordinates before indexing the texture

Like watching a slide projected onto the mapped polygon





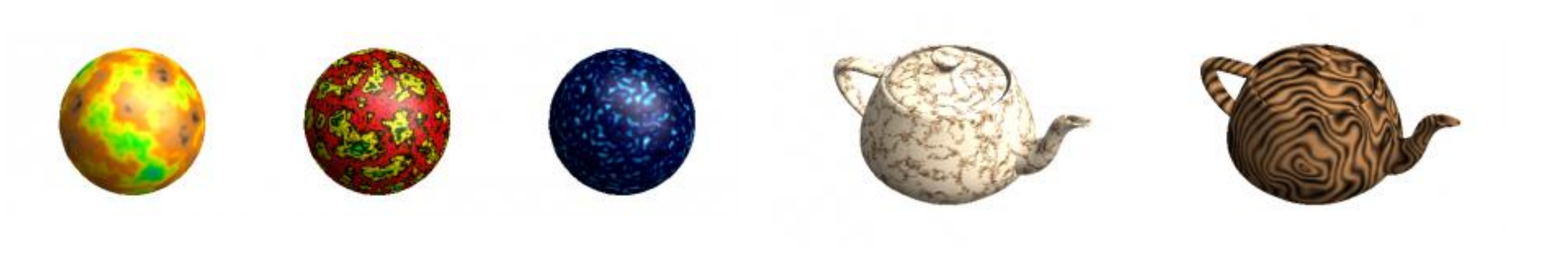
Procedural Texture Mapping

Instead of looking up an image, pass the texture coordinates to a function that computes the texture value on the fly

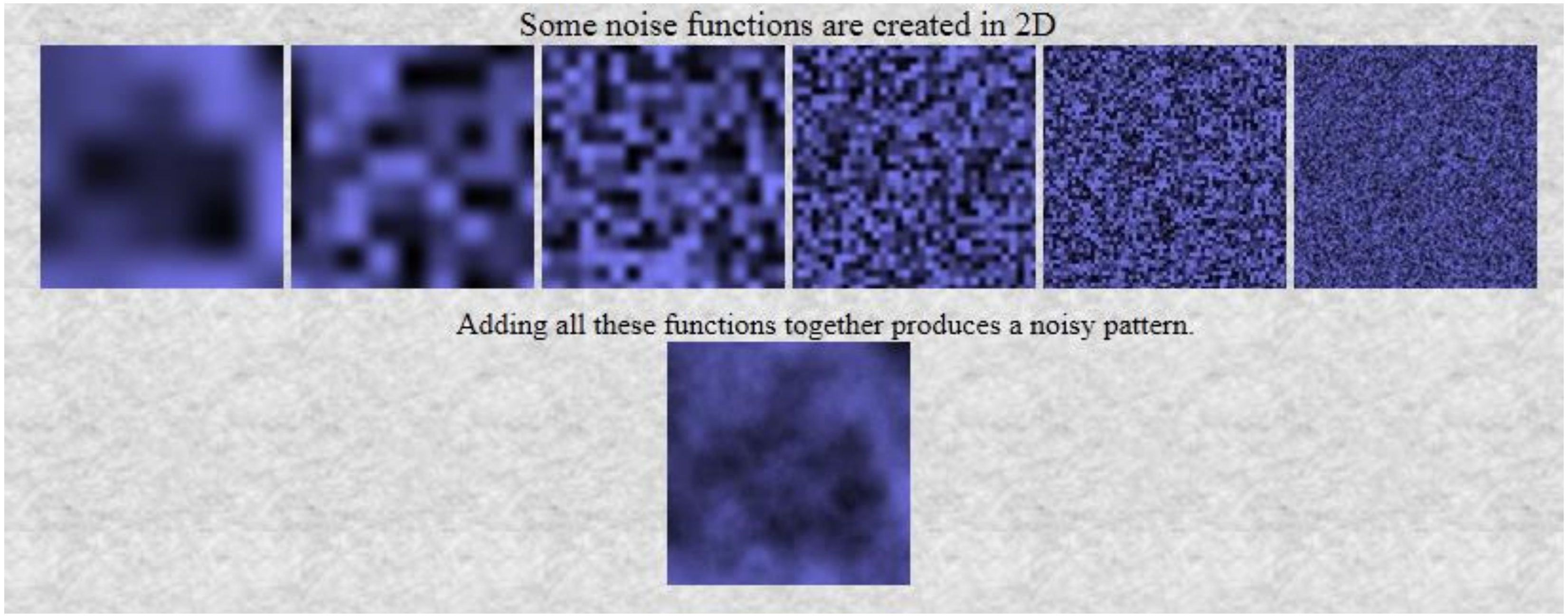
- hyper-textures
- Perlin noise functions

Near-infinite resolution with small storage cost

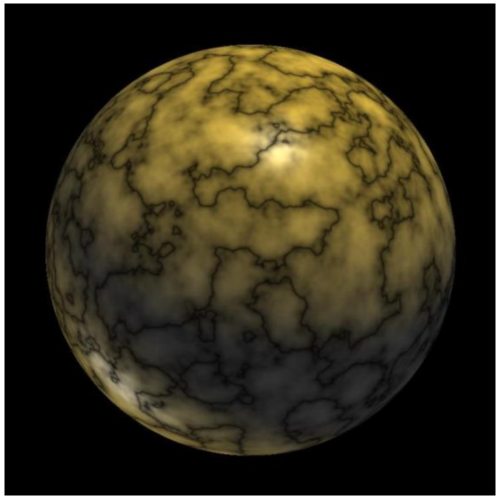
- can be slow to evaluate, but GPUs help



Perlin Noise Functions


Some noise functions are created in 2D
Adding all these functions together produces a noisy pattern.

texturing 3D solid object

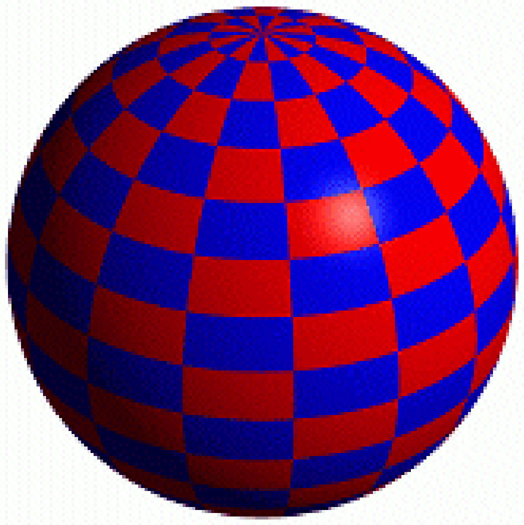


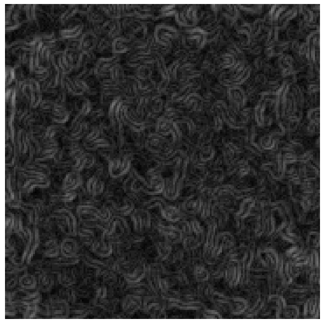
http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

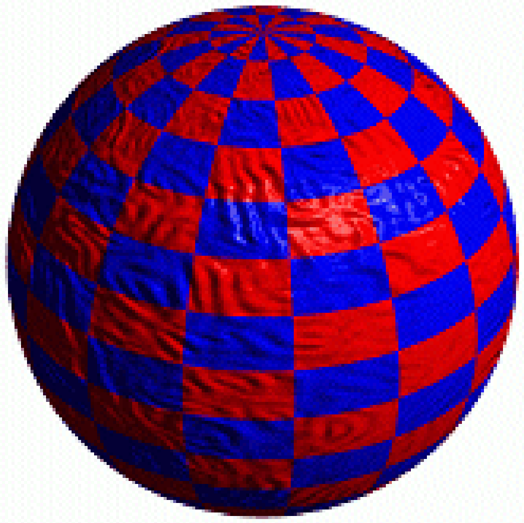
Bump Mapping

Textures can be used to alter the surface normal of an object, but does not change the actual shape of the surface – we are only shading it as if it were a different shape!

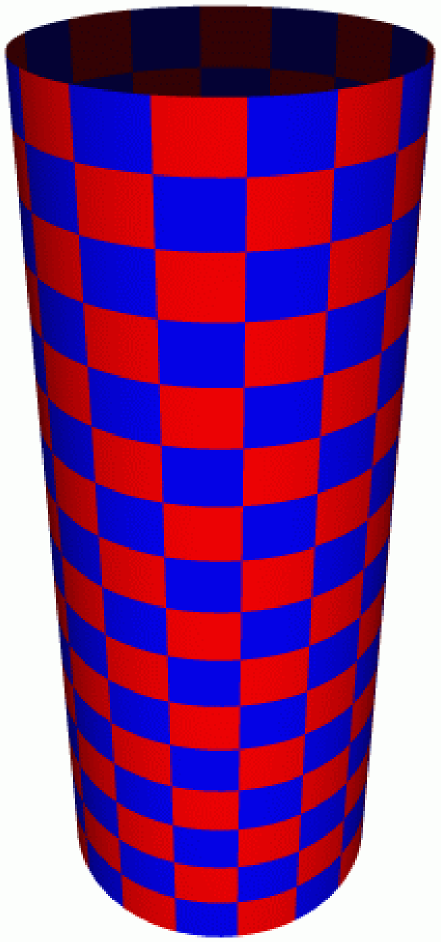
Since the actual shape of the object does not change, the silhouette edge of the object will not change. Bump Mapping also assumes that the illumination model is applied at every pixel (as in Phong Shading).

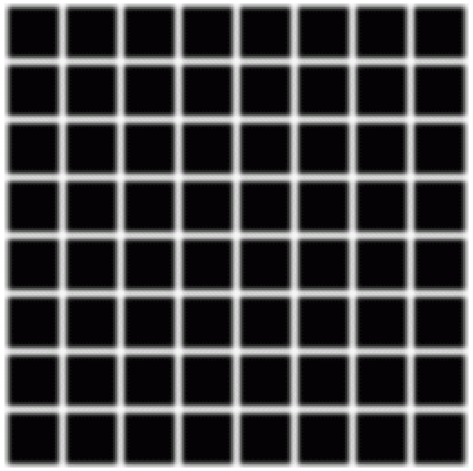

Sphere w/Diffuse Texture

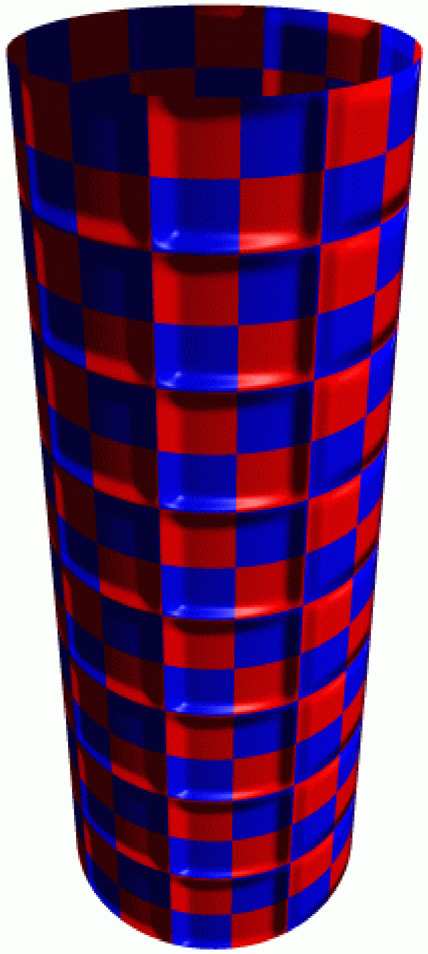

Swirly Bump Map


Sphere w/Diffuse Texture & Bump Map

Bump Map Examples

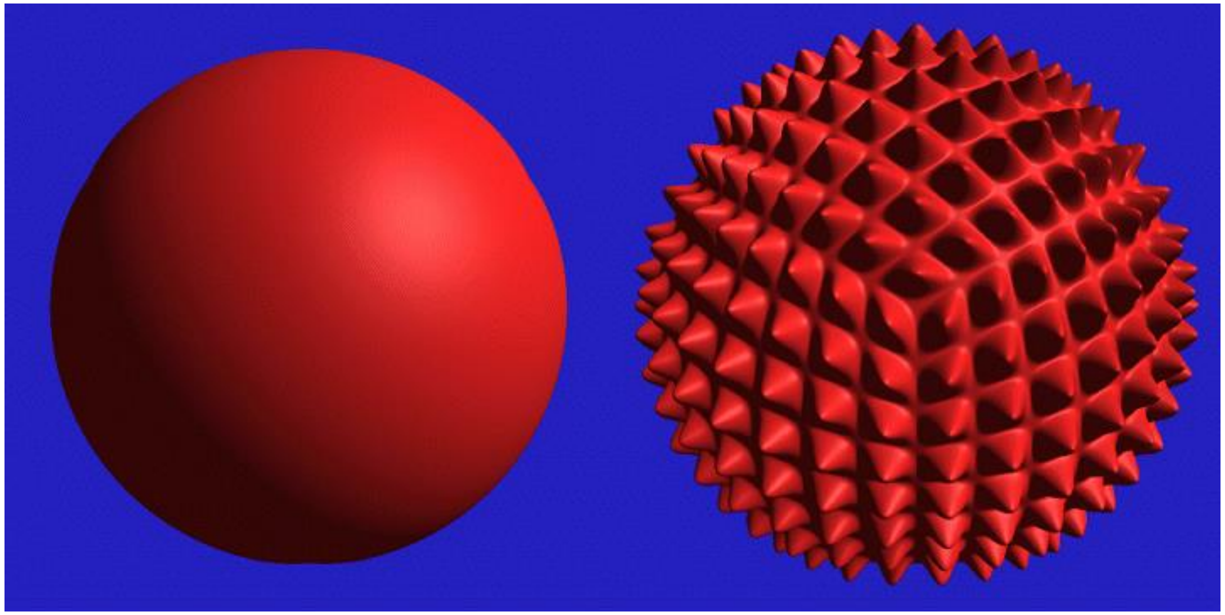

Cylinder w/Diffuse Texture Map


Bump Map


Cylinder w/Texture Map & Bump Map

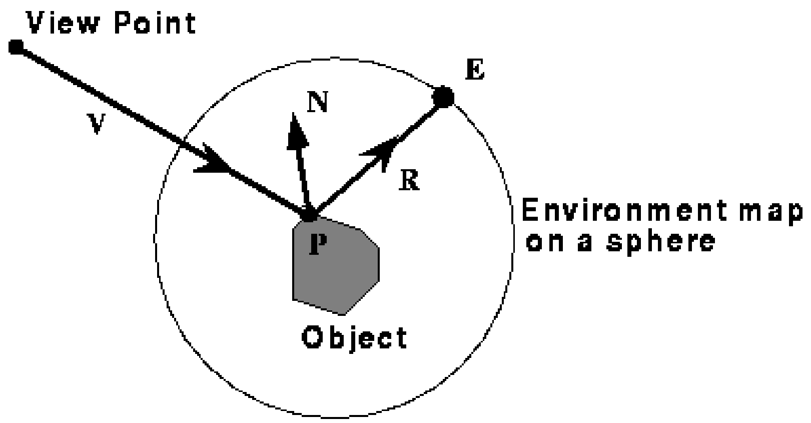
Displacement Mapping

We use the texture map to actually move the surface point. This is called *displacement mapping*. How is this fundamentally different than bump mapping?



The geometry must be displaced before visibility is determined.

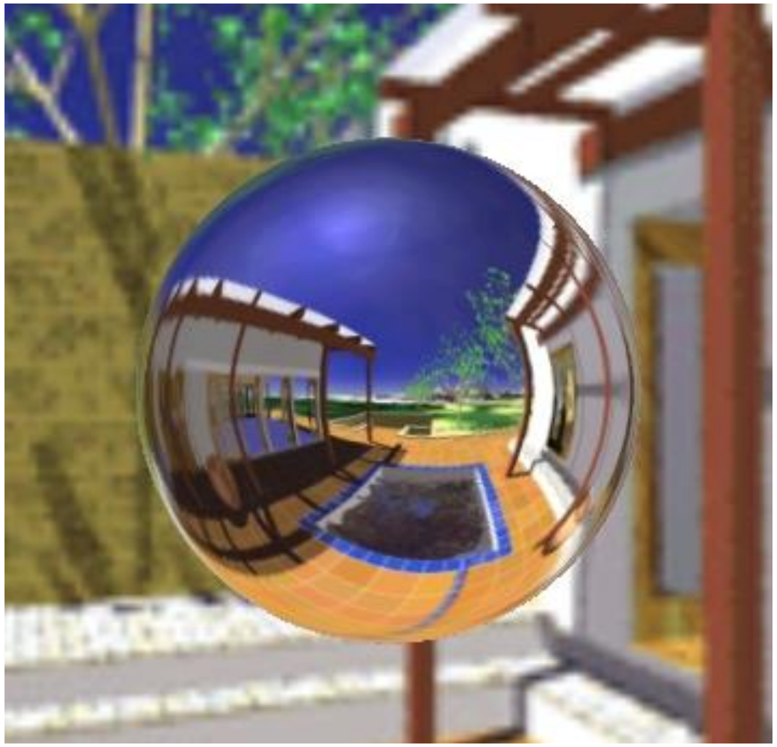
Environment Maps

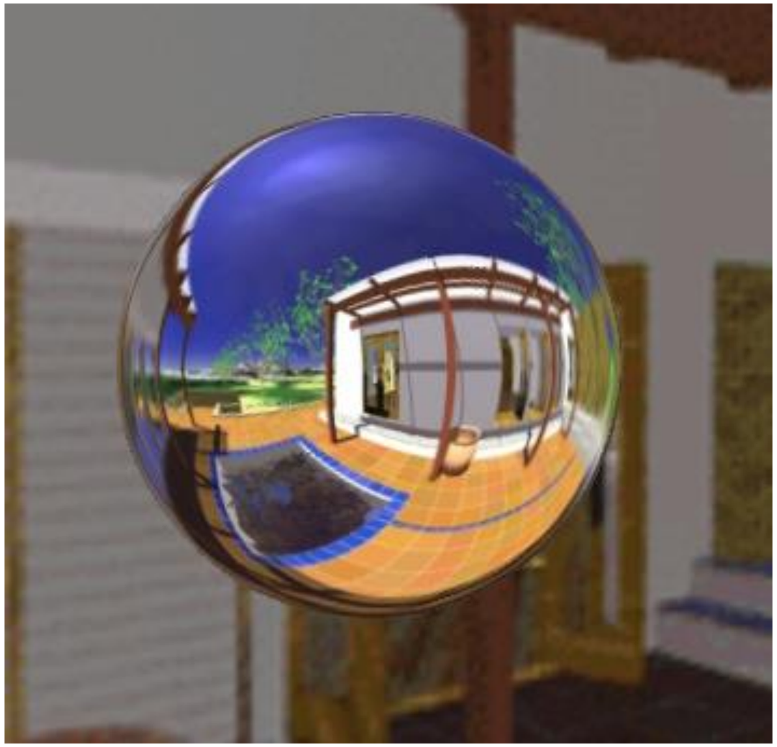


We use the *direction* of the reflected ray to index a texture map.

We can simulate reflections. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.

Environment Mapping





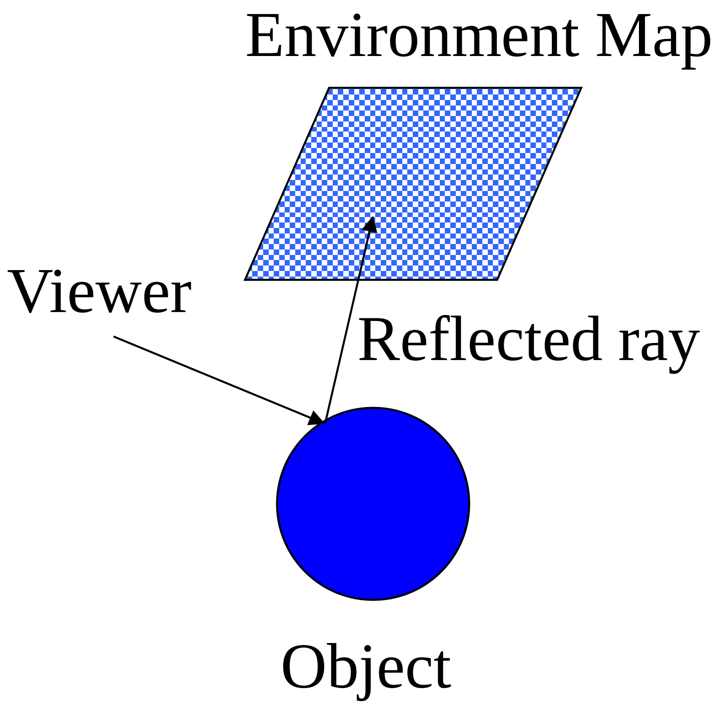
Environment Mapping

Environment mapping produces reflections of its environment on shiny objects

Texture is transferred in the direction of the reflected ray from the environment map onto the object

In principle, trace a ray from eye to a point P on the object surface, determine the reflection ray, and then trace the reflection ray until it hits the surrounding sphere or cube. The color from environment map at this hit-point will be placed at point P

Reflected ray: ***R***=2(***N·V***)***N***-***V***. ***R*** is used to index the environment map.



Environment Mapping

The environment map may take one of several forms:

- Cubic mapping: map resides on 6 faces of a cube
- Spherical mapping: map resides on a sphere surrounding the object

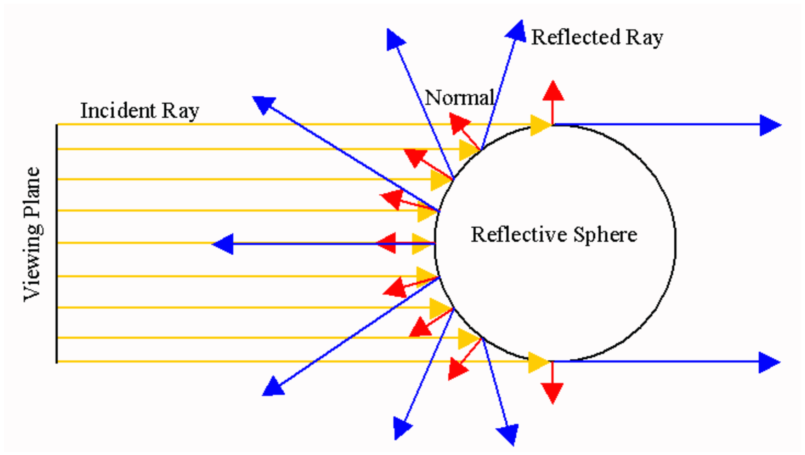
The map should contain a view of the world with the point of interest on the object as the eye

- 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

Spherical Mapping



Implemented in hardware

Single texture map

Problems:

- Highly non-uniform sampling
- Highly non-linear mapping

Cubic Mapping

The map resides on the surfaces of a cube around the object

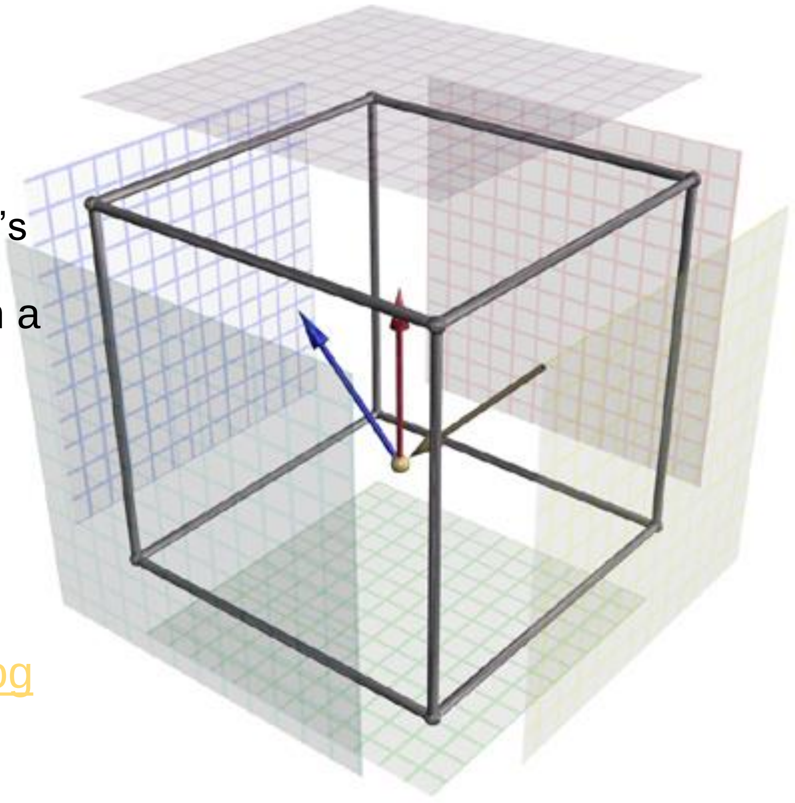
- Typically, align the faces of the cube with the coordinate axes

To generate the map:

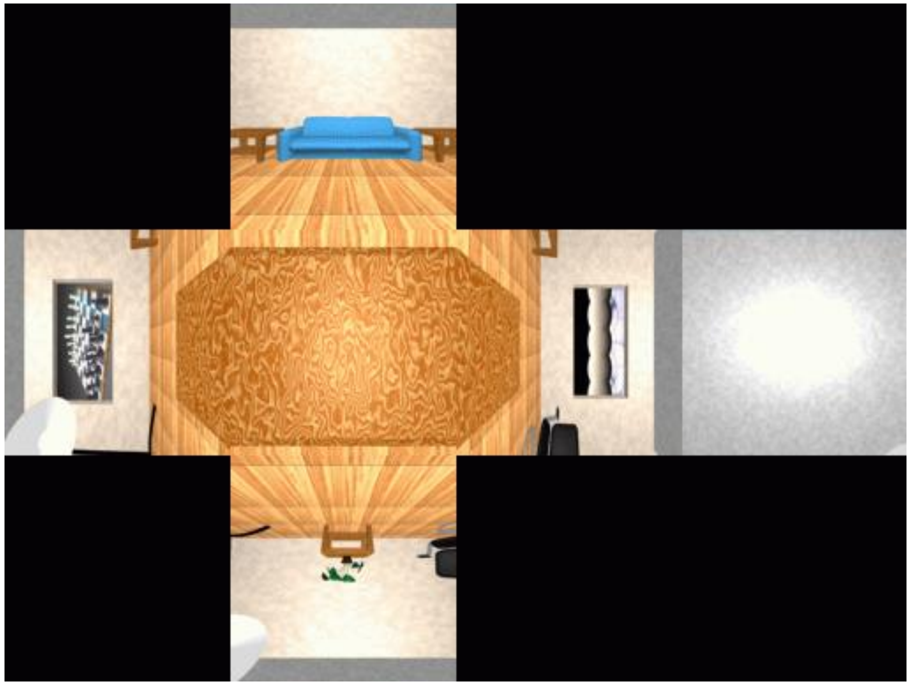
- For each face of the cube, render the world from the center of the object with the cube face as the image plane
  - Rendering can be arbitrarily complex (it's off-line)
- Or, take 6 photos of a real environment with a camera in the object's position
  - Actually, take many more photos from different places the object might be
  - Warp them to approximate map for all intermediate points

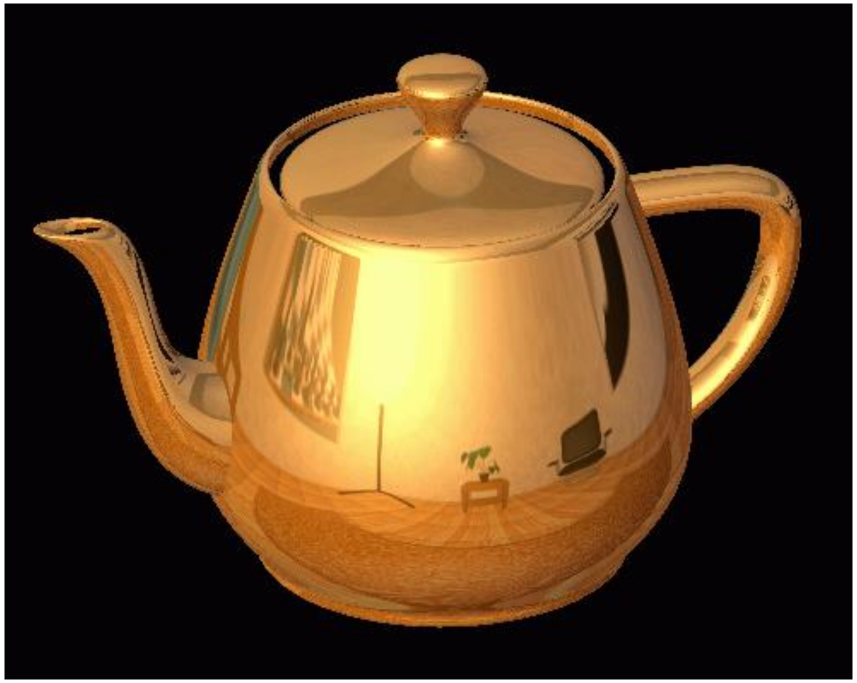
Remember *Terminator 2*?

http://developer.nvidia.com/object/cube_map_ogl_tutorial.html



Cubic Map Example



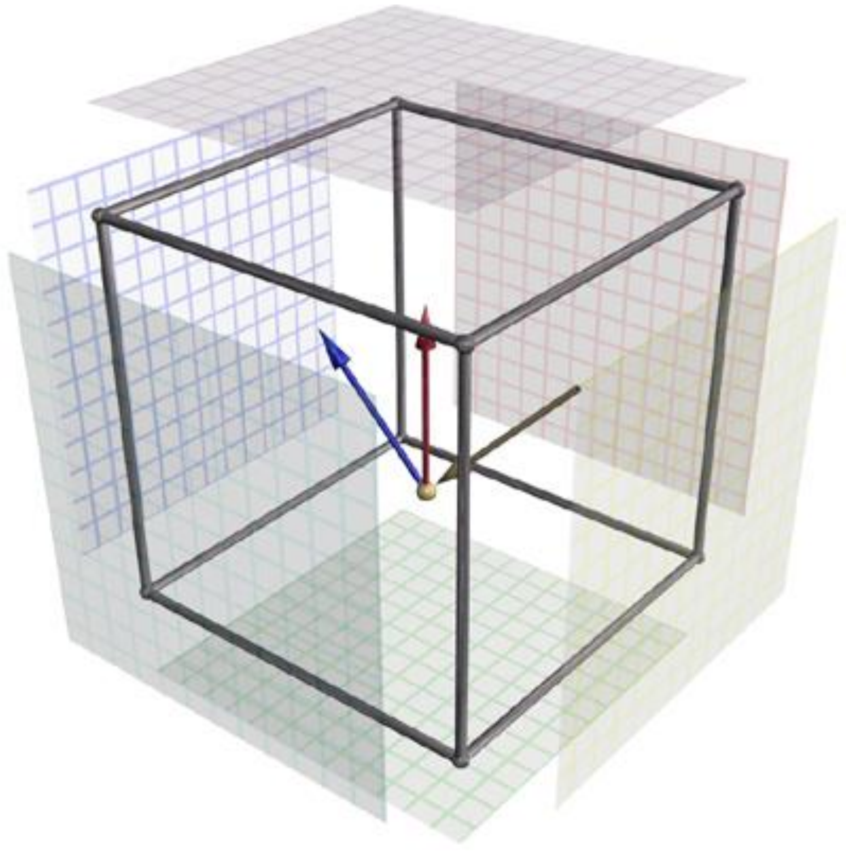


Indexing Cubic Maps

Assume you have ***R*** and the cube's faces are aligned with the coordinate axes, and have texture coordinates in [0,1]x[0,1]

- How do you decide which face to use?
- How do you decide which texture coordinates to use?

What is the problem using cubic maps when texture coordinates are only computed at vertices?



OpenGL Spherical Map

We can use automatically generated texture coordinates in OpenGL. For example, to generate the texture coordinates of spherical mapping

```
// Build the environment as a texture object
// Automatically generate the texture coordinates
glTexGenf(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGenf(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
// Bind the environment texture
…
// Draw object
…
```

Example

OpenGL Cubemap Texture

Enabling and disabling the cube map texture is done as follows:

```
glEnable(GL_TEXTURE_CUBE_MAP);
glDisable(GL_TEXTURE_CUBE_MAP);


glGenTextures(1,&cubemap_id);

glBindTexture(GL_TEXTURE_CUBE_MAP,cubemap_id);
```

Load images into a cube map. Each face in the example is a 64x64 RGB image.

```
GLubyte face[6][64][64][3];

for (i=0; i<6; i++) {
    glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
      0,                    //level
      GL_RGB8,              //internal format
      64,                   //width
      64,                   //height
      0,                    //border
      GL_RGB,               //format
      GL_UNSIGNED_BYTE,     //type
      &face[i][0][0][0]); // pixel data
  }
```

OpenGL Implementation (cont.)

We can use automatically generated texture coordinates in OpenGL

```
glTexGenfv(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
  glTexGenfv(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
  glTexGenfv(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);
  glEnable(GL_TEXTURE_GEN_R);

  // Bind the environment texture
   …
   // Draw object
   …
```

For the cube map to operate correctly, correct per-vertex normals must be supplied

Rendering With Natural Light

Paul Debevec

link