CSC 4356
Interactive Computer Graphics
Lecture 20: Texture Mapping (part 3)

Jinwei Ye

http://www.csc.lsu.edu/~jye/CSC4356/

Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

From Last Time

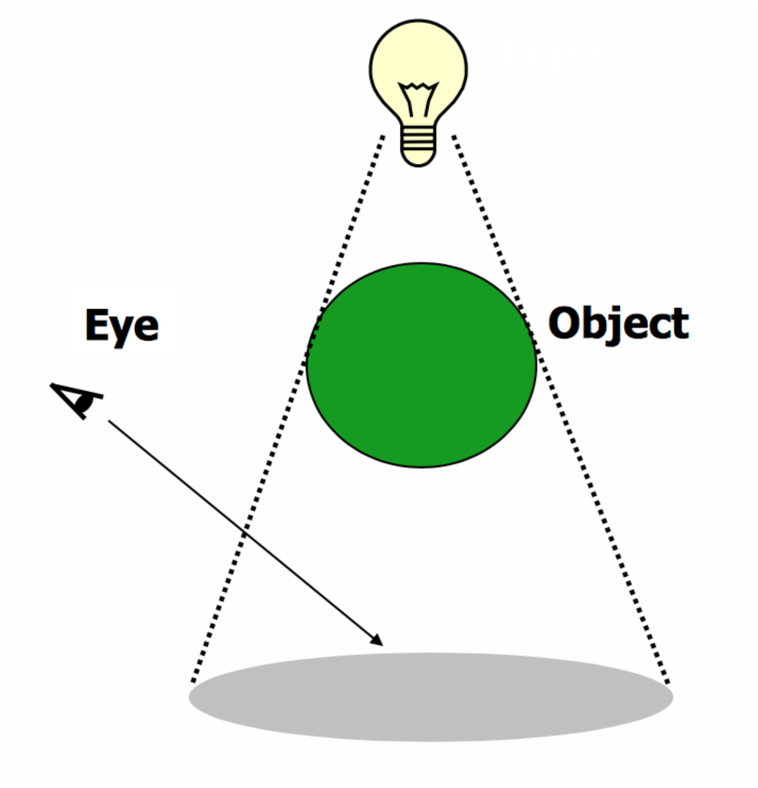
- Texture filtering
  - MIP mapping
  - Summed area table
- Texture synthesis
  - Efros-Leung algorithm

This Lecture

- Advanced texture mapping techniques
  - Shadow Map
  - Environment Map
  - Bump Map
  - Displacement Map
  - Multi-Texture

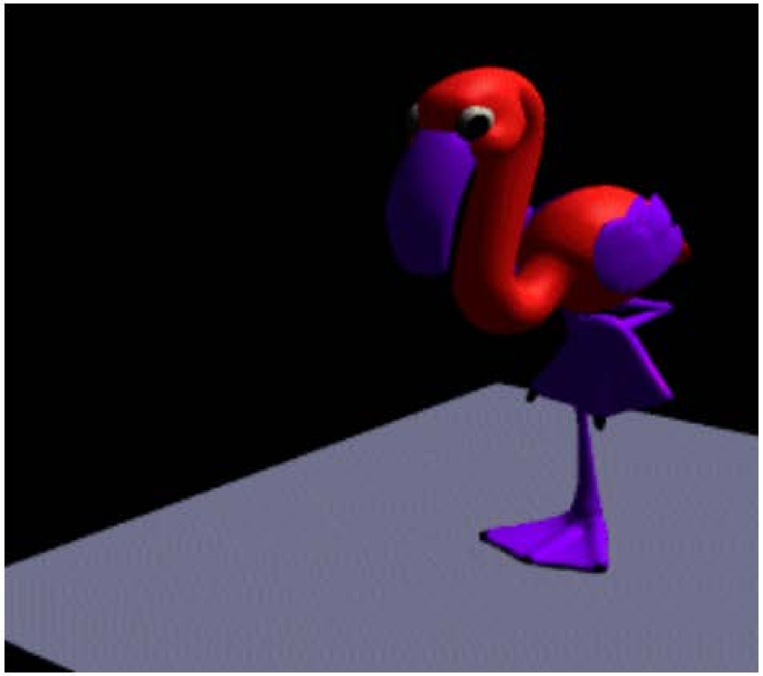
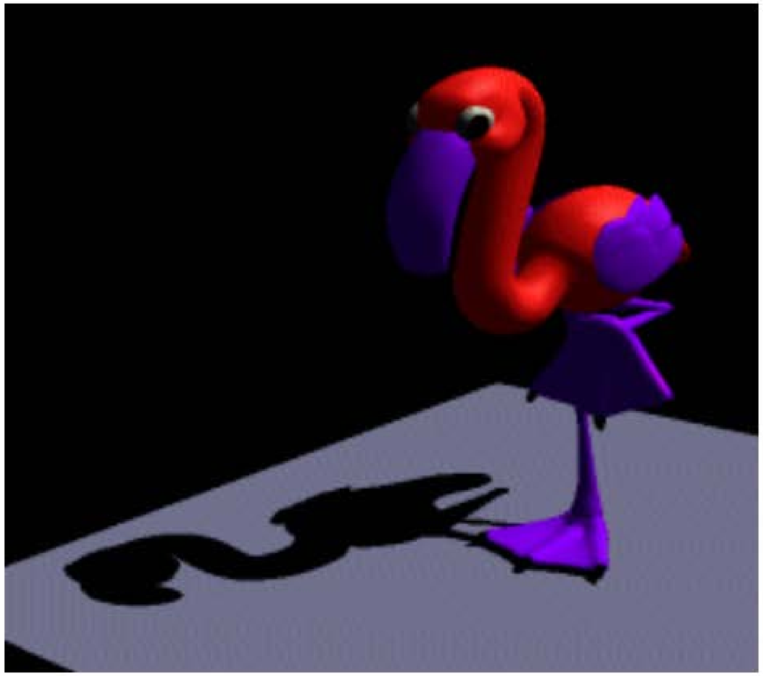
What is Shadow?

- Shadows occur on surfaces that are not fully visible the light

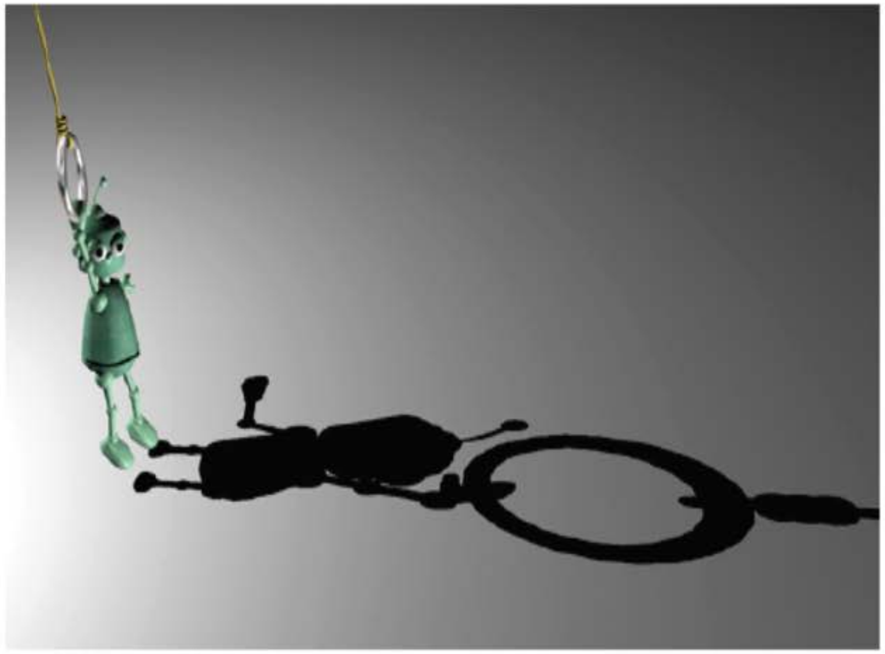


Shadows

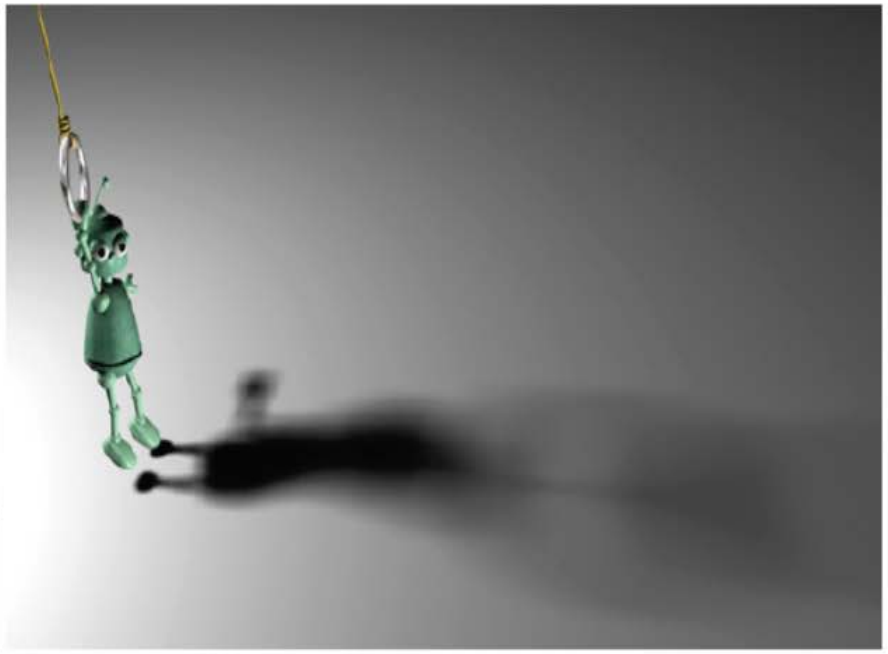
- Improved understanding of an object’s shape and position
- Enhance realism

Types of Shadow



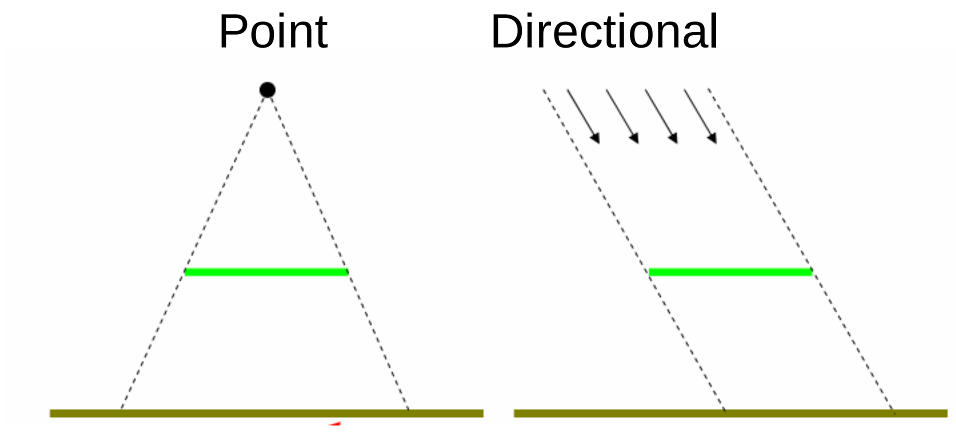
Hard Shadow



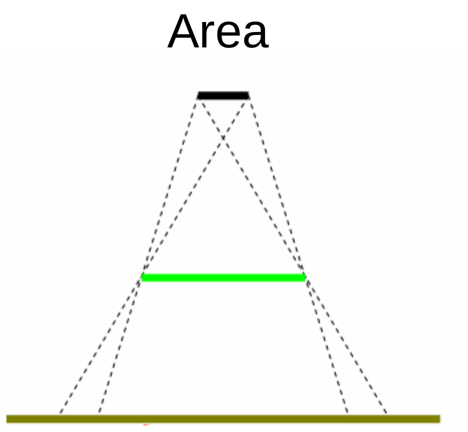
Soft Shadow

Types of Shadow

- Different types of light sources generate different types of shadows
  - Point or directional light (hard shadow)
  - Area light (soft shadow)

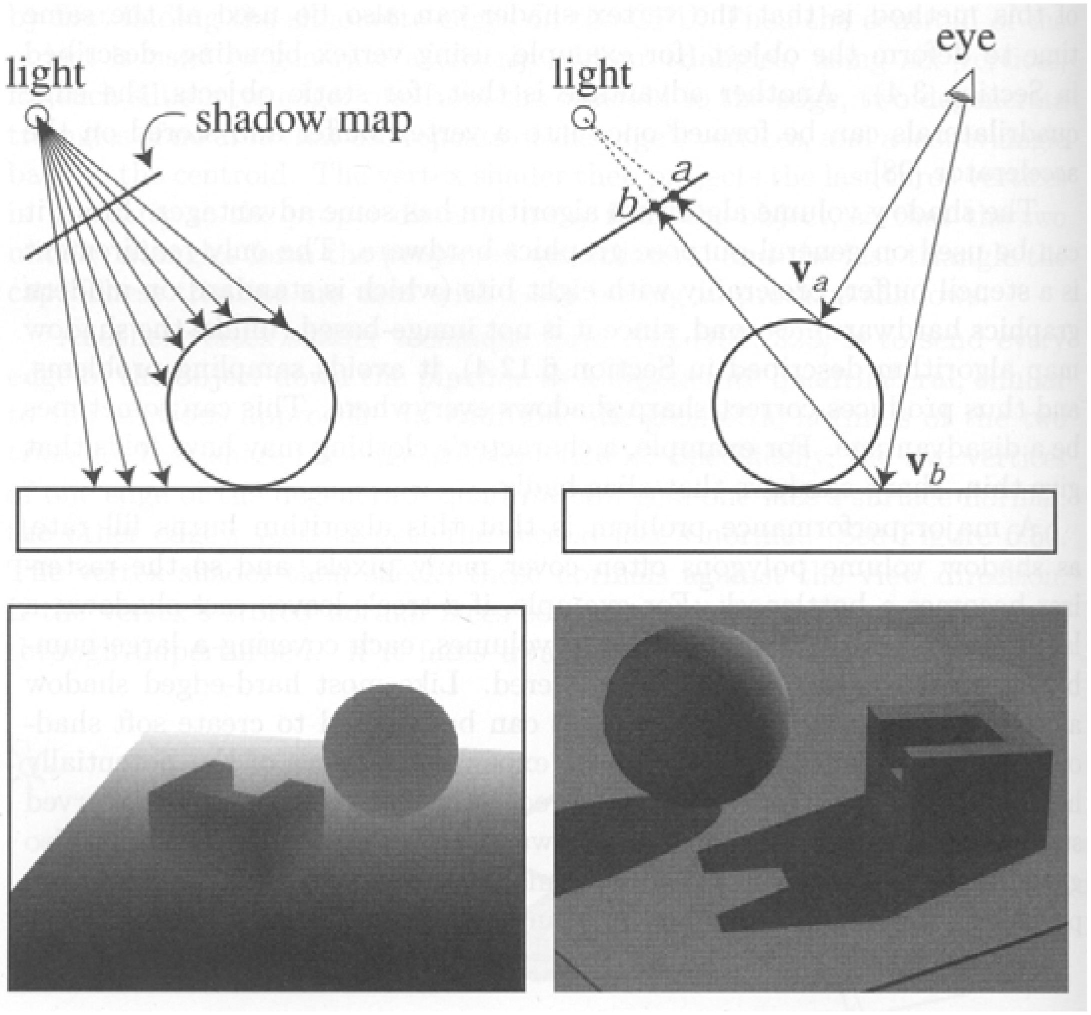


Hard Shadows



Soft Shadow

Shadow Map



Shadow Map

- Render an image from the light's point of view
  - Camera look-from point is the light position
  - Aim camera to look at objects in scene
  - Render only the z-buffer depth values
    - Don't need colors
    - Don't need to compute lighting or shading
      - (unless a procedural shader would make an object transparent)

- Store result in a *shadow map* *(*a.k.a. depth map*)*
  - Store the depth values (z-buffer)
  - Also store the (inverse) camera & projection transform

- Remember, z-buffer pixel holds depth of closest object to the camera
  - A shadow map pixel contains the distance of the closest object to the light (because camera is in the position of light)

Shadow Map

- Directional light source
  - Use orthographic shadow camera
- Point light source
  - Use perspective shadow camera

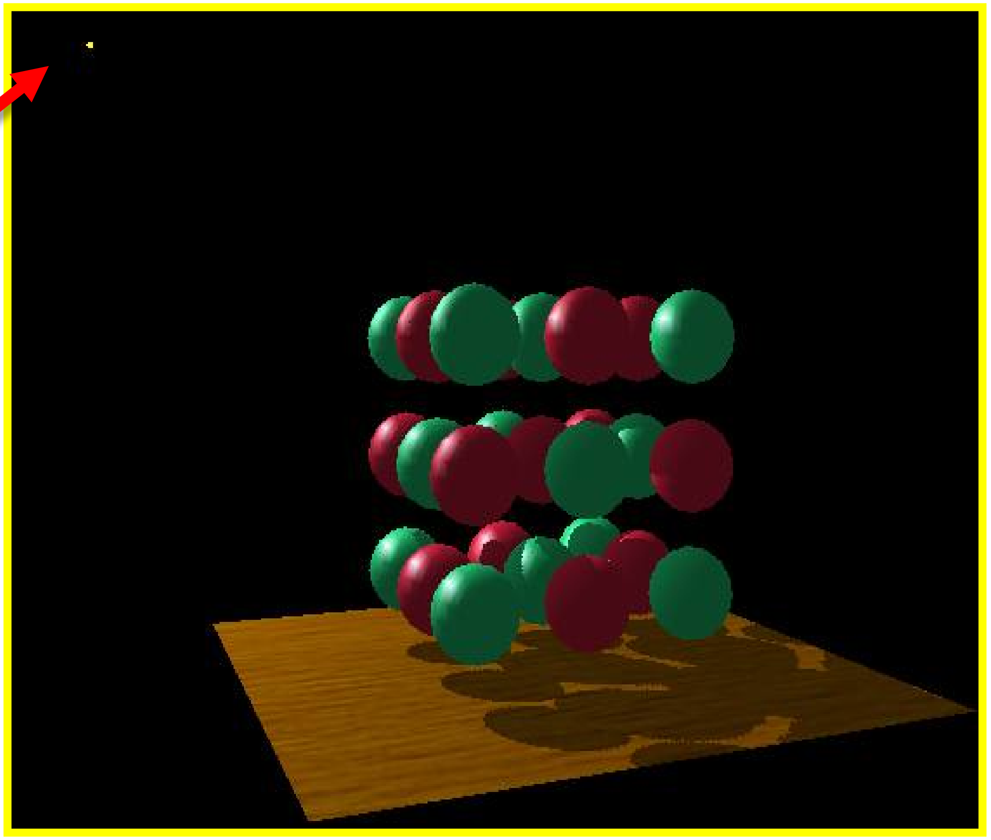
Rendering Shadow

- When lighting a point on a surface
  - For each light that has a shadow map…
  - Transform the point to the shadow map’s image space
    - Get X,Y,Z values
    - Compare Z to the depth value at X,Y in the shadow map
    - If the shadow map depth is less than Z
      - some other object is closer to the light than this point
      - this light is blocked, don’t include it in the illumination
    - If the shadow map is the same as Z
      - this point is the one that’s closest to the light
      - illuminate with this light
      - (because of numerical inaccuracies, test for almost-the-same-as Z)

Shadow Mapping: Example

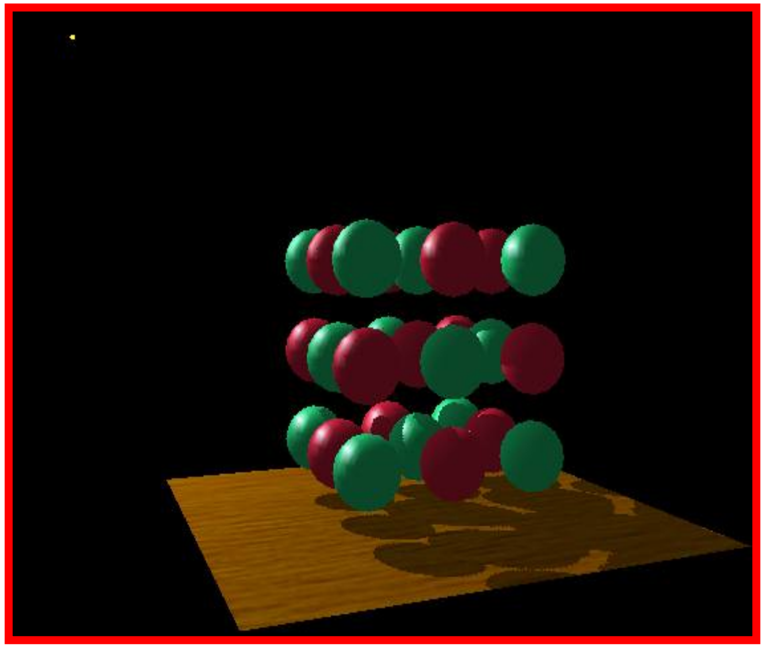
- A scene with shadows

the point light source

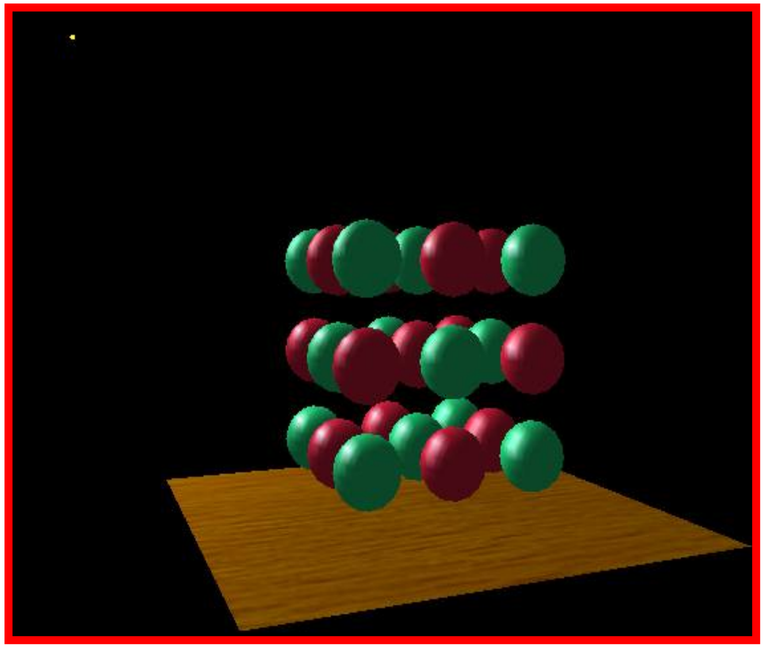


Shadow Mapping: Example

- Without and with shadows



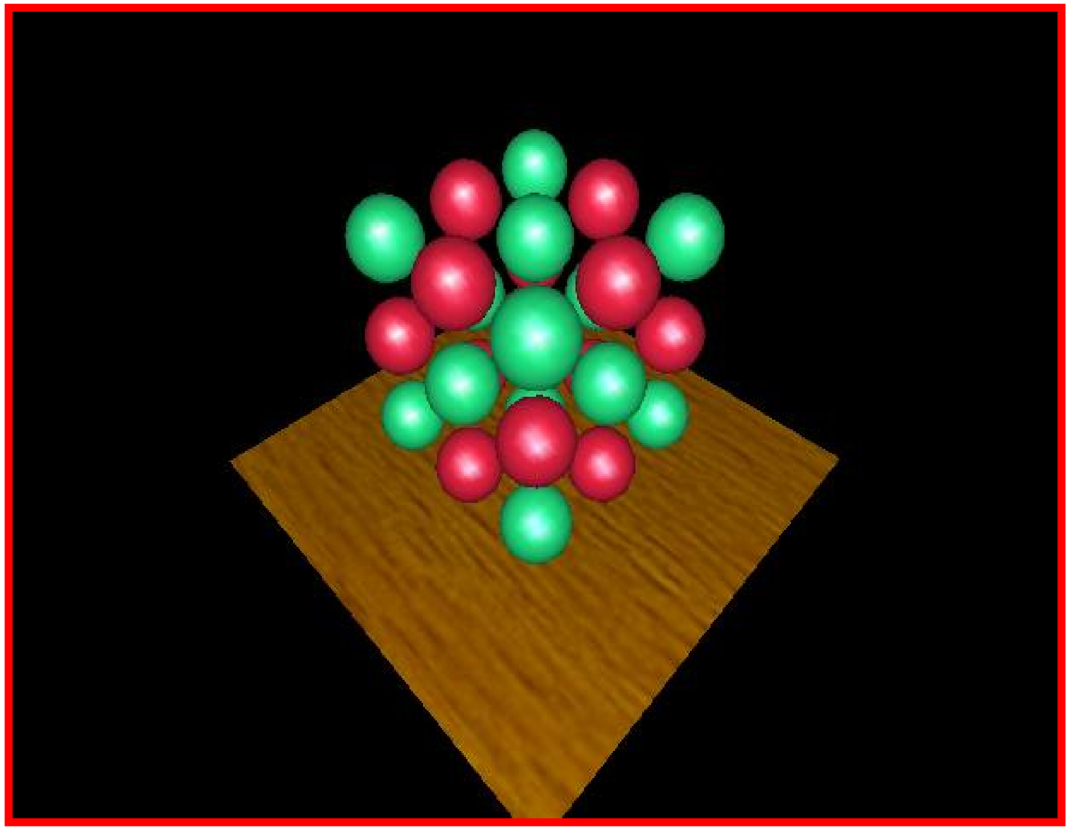
with shadows

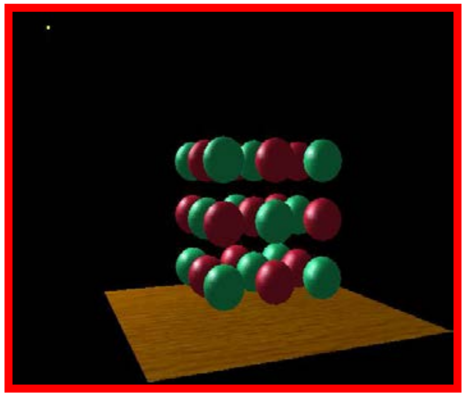


without shadows

Shadow Mapping: Example

- The scene from the shadow camera
  - – no need to save the color image

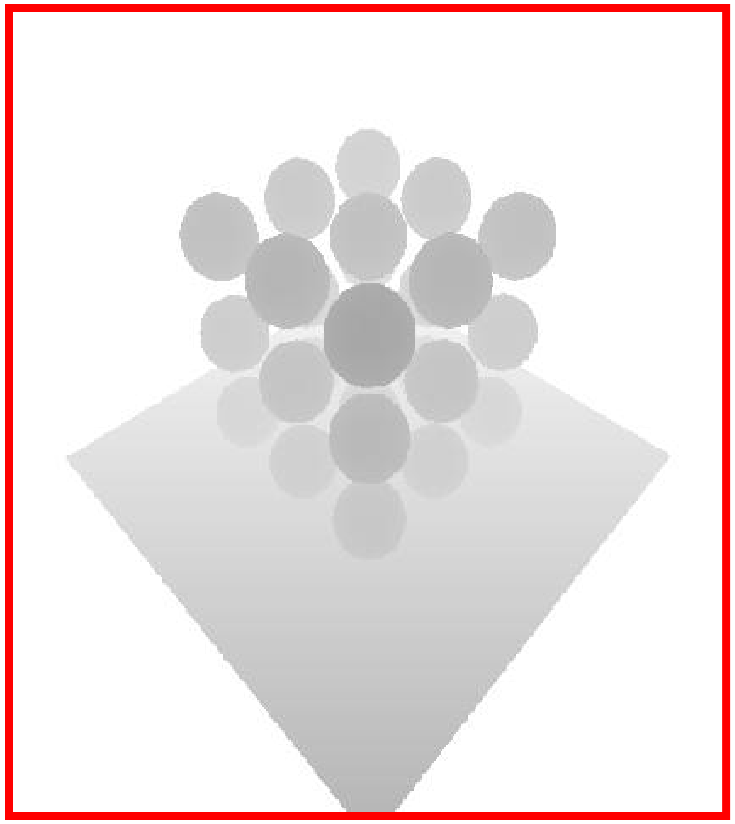


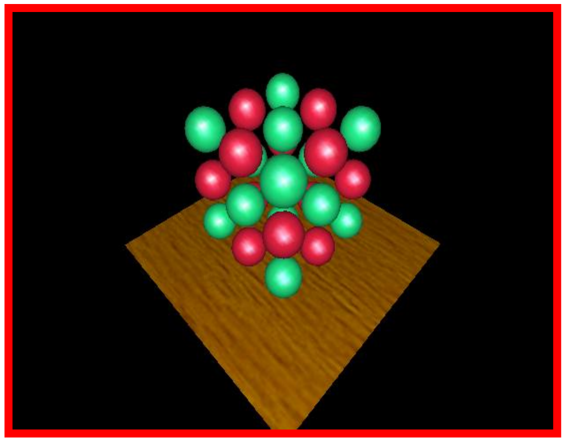


from the eye's point-of-view

Shadow Mapping: Example

- The shadow map depth buffer
  - Darker is closer to the camera

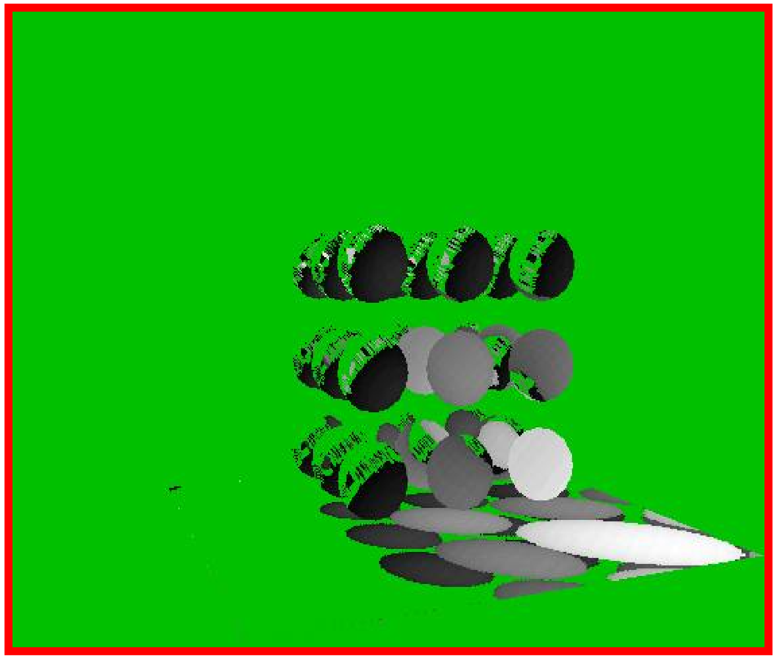
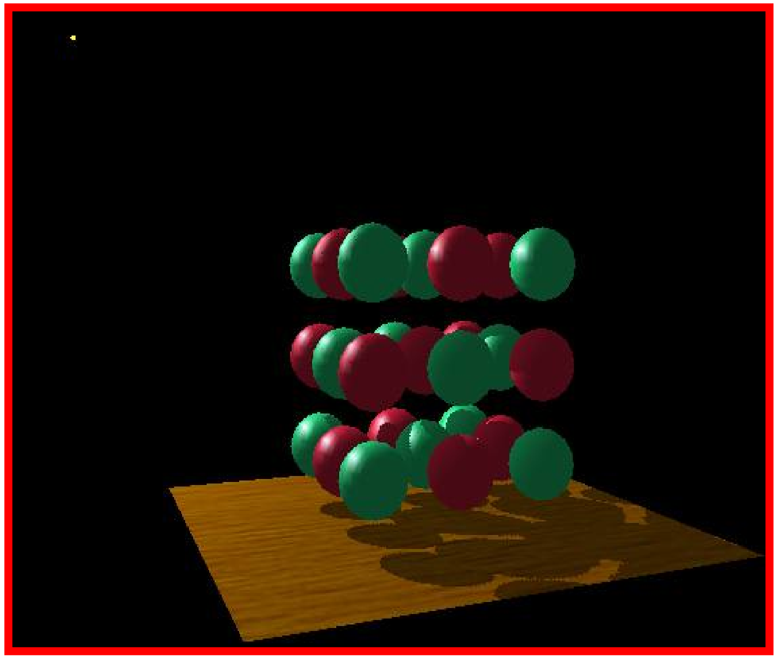




from the light's point-of-view

Shadow Mapping: Example

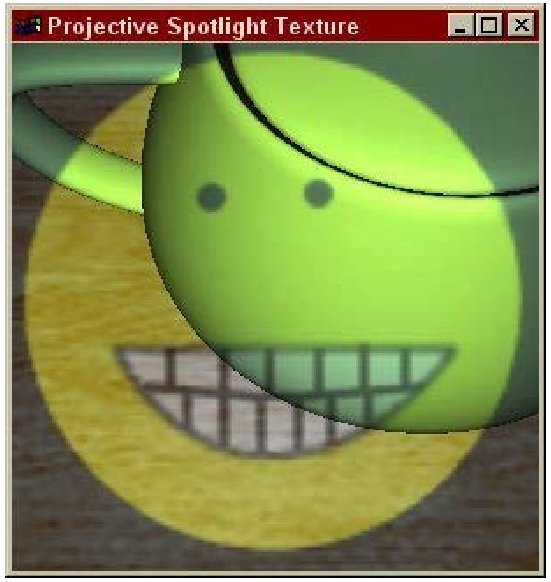
- Visualization
  - Green: surface light Z is (approximately) equal to depth map Z
  - Non-green: surface is in shadow

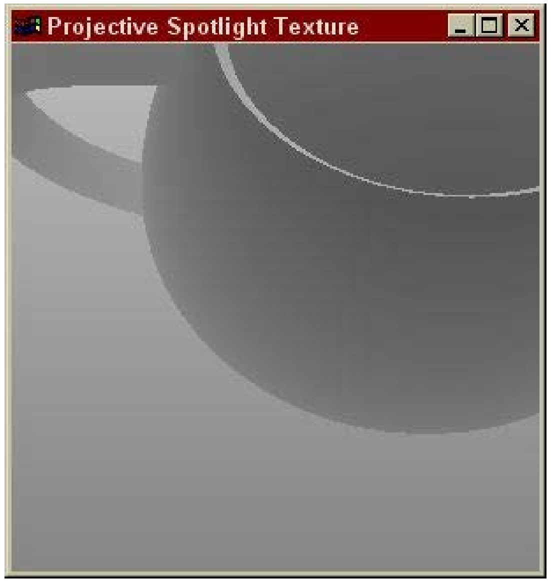
Shadow Mapping: Notes

- Very commonly used
- Problems:
  - Blocky shadows, depending on resolution of shadow map
  - Shadow map pixels & image samples don’t necessarily line up
    - Hard to tell if object is really the closest object
    - Typically add a small bias to keep from self-interfering
    - But the bias causes shadows to separate from their objects
  - No great ways to get soft shadows

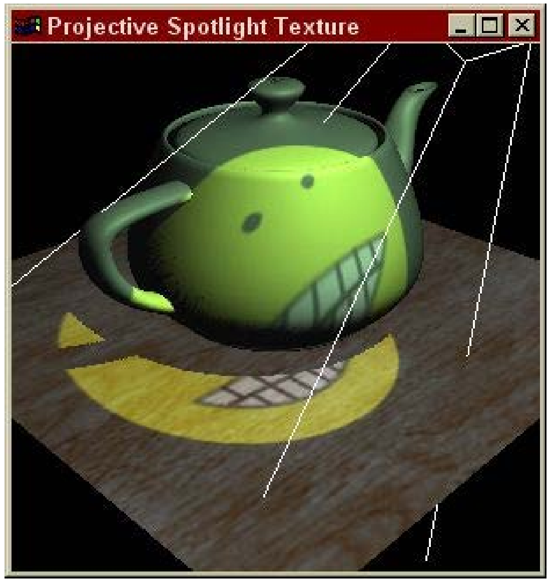
Questions?



Light’s View



Depth/Shadow Map

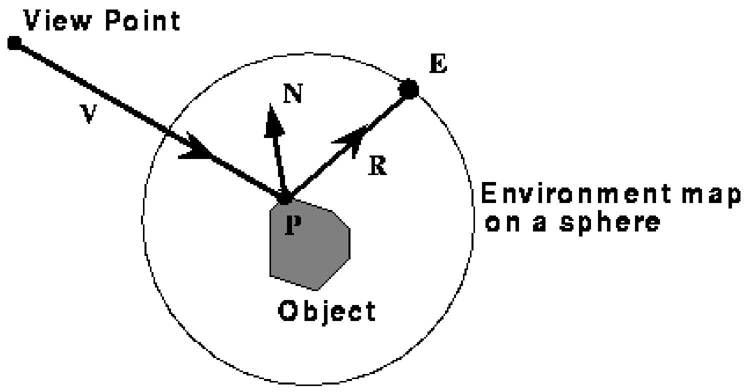


Eye’s View

Images from Cass Everitt et al.,
“Hardware Shadow Mapping”
NVIDIA SDK White Paper

Environment Maps

We can use transformed surface normals to compute indices into the texture map. These sorts of mapping can be used to simulate reflections, and other shading effects. 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.

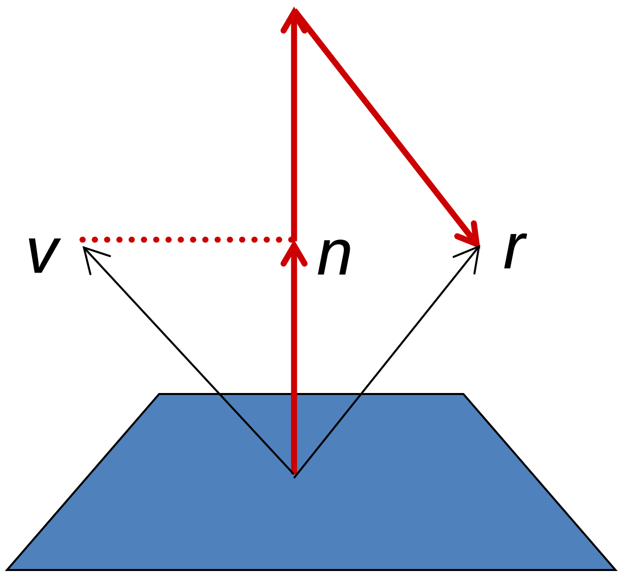


Environment Mapping Steps

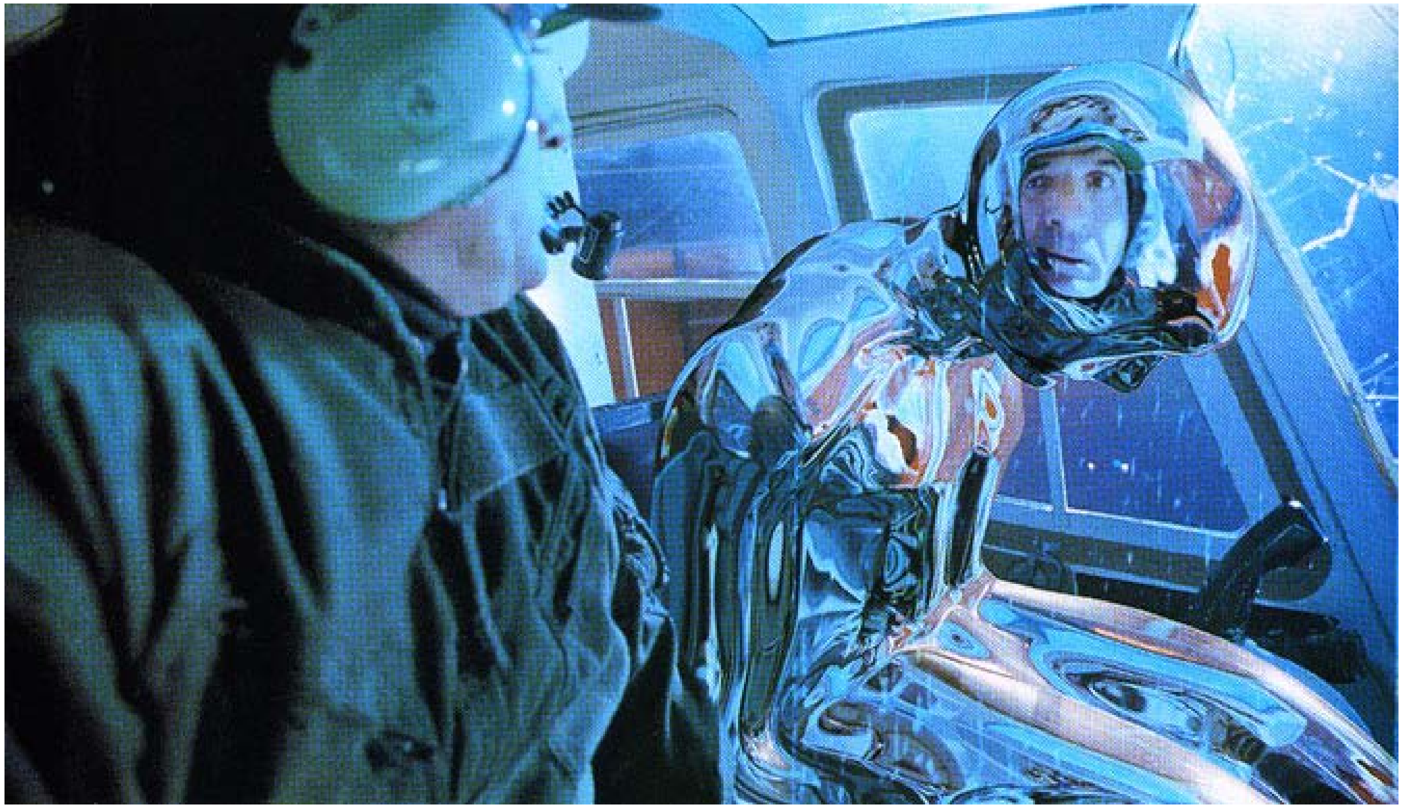
- Create a 2D environment map
- For each pixel on a reflective object, compute the normal
- Compute the reflection vector based on the eye position and surface normal
- Use the reflection vector to compute an index into the environment texture
- Use the corresponding texel to color the pixel

How to compute reflection vector?

- Given viewing vector v and surface normal n, the reflection vector r can be computed as: $\bar{r} = 2(\bar{n} \cdot \bar{v})\bar{n} - \bar{v}$

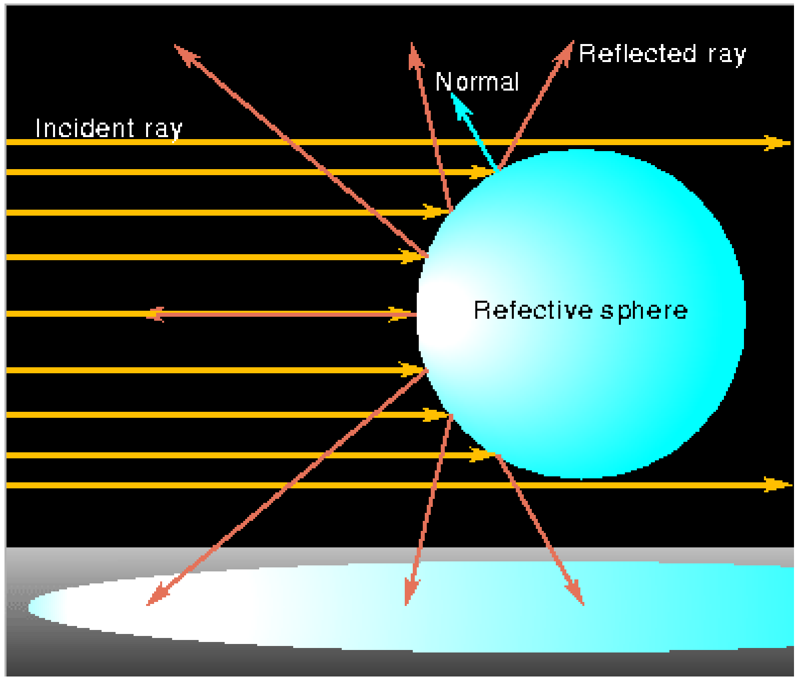


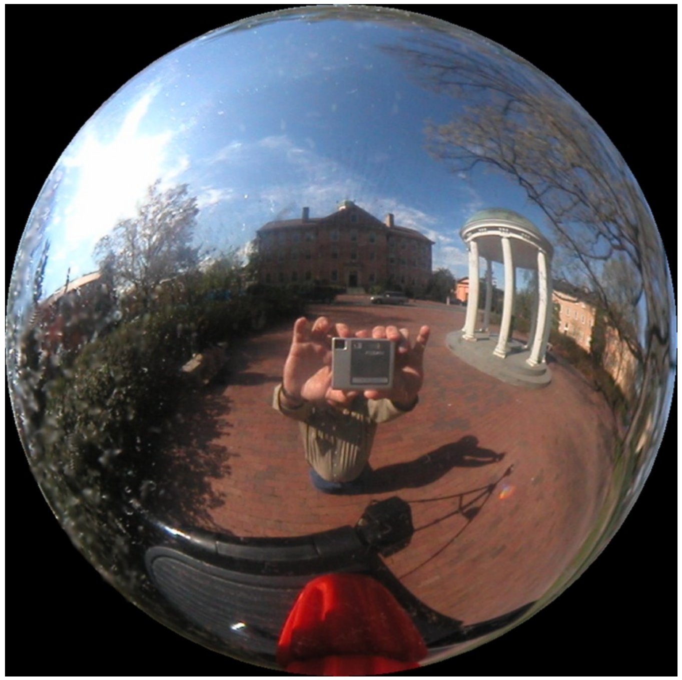
Environment Mapping Example


Terminator II, 1991

Sphere Mapping Basics

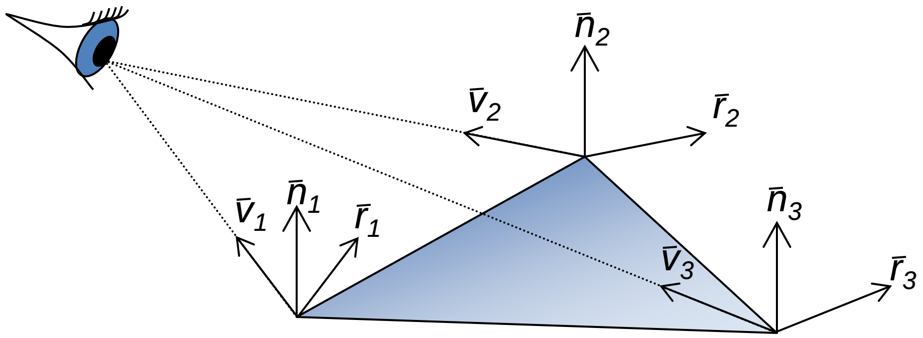
- OpenGL provides special support for a particular form of Normal mapping called sphere mapping. It maps the normals of the object to the corresponding normal of a sphere. It uses a texture map of a sphere viewed from infinity to establish the color for the normal.



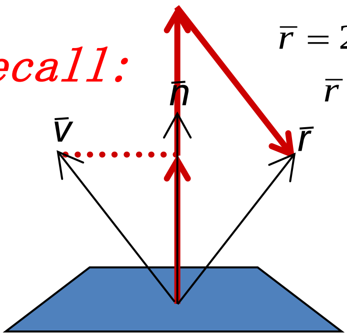


Sphere Mapping

- Mapping the normal to a point on the sphere



Recall:



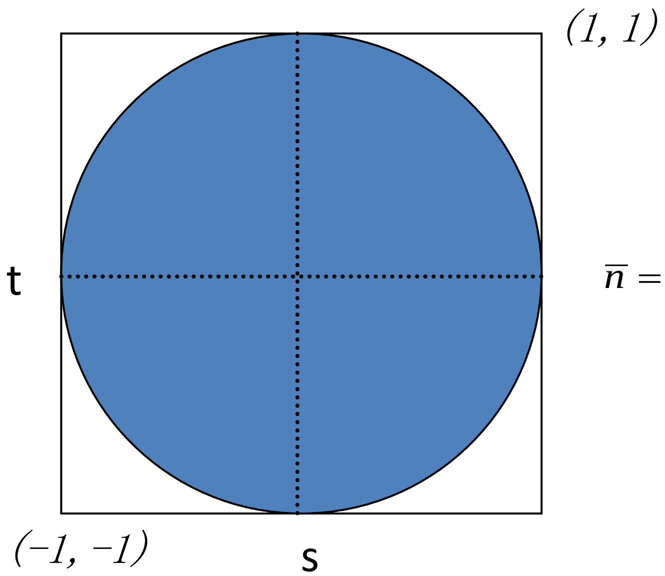
$$\bar{r} = 2(\bar{n}\cdot\bar{v})\bar{n} - \bar{v}$$

$$\bar{r} + \bar{v} = \alpha\bar{n}$$

$$\alpha\bar{n} = \bar{r} + \bar{v} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\frac{\alpha\bar{n}}{\|\alpha\bar{n}\|} = \begin{bmatrix} \frac{r_x}{p} \\ \frac{r_y}{p} \\ \frac{r_z+1}{p} \\ 0 \end{bmatrix}$$

$$p = \sqrt{r_x^2 + r_y^2 + (r_z+1)^2}$$



$$\bar{n} = \begin{bmatrix} s \\ t \\ \sqrt{1-s^2-t^2} \\ 0 \end{bmatrix}$$

$$s = \frac{r_x}{p} \qquad t = \frac{r_y}{p}$$

$$s' = \frac{s}{2} + \frac{1}{2} \qquad t' = \frac{t}{2} + \frac{1}{2}$$

$$s' = \frac{r_x}{2p} + \frac{1}{2} \qquad t' = \frac{r_y}{2p} + \frac{1}{2}$$

Sphere Mapping Steps

- To access the sphere map texture
  - The surface normal (n) and view (v) vectors need to be first transformed to the eye space
  - Then compute the reflection vector as usual

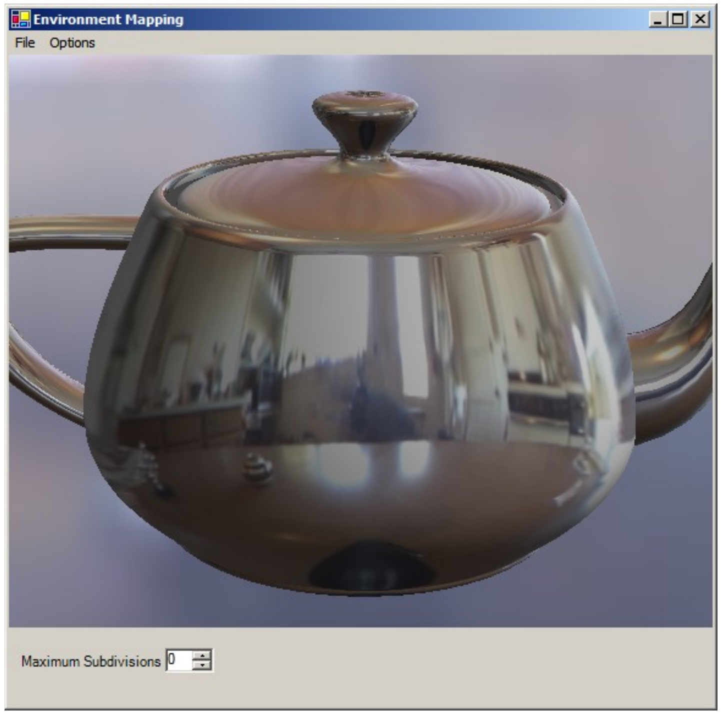
    $(r = (rx,ry,rz) = 2(n\cdot v)n - v)$
  - Normalize it and use x and y to access the sphere texture map: $s = rx / 2p + 1/2$; $t = ry / 2p + 1/2$;
  - Now, compute the *sphere normal* in the local space

    $n = (rx,ry,rz) + (0,0,1)$

    where $p = \mathrm{sqrt}(rx^2 + ry^2 + (rz+1)^2)$

OpenGL code Example

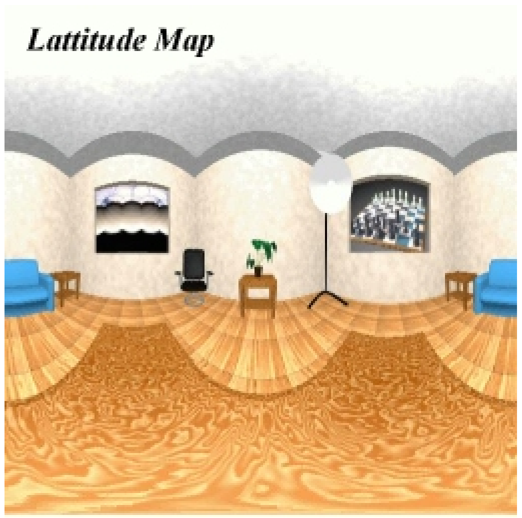
```
// this gets inserted where the texture is created

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, (int) GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, (int) GL_SPHERE_MAP);

glEnable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

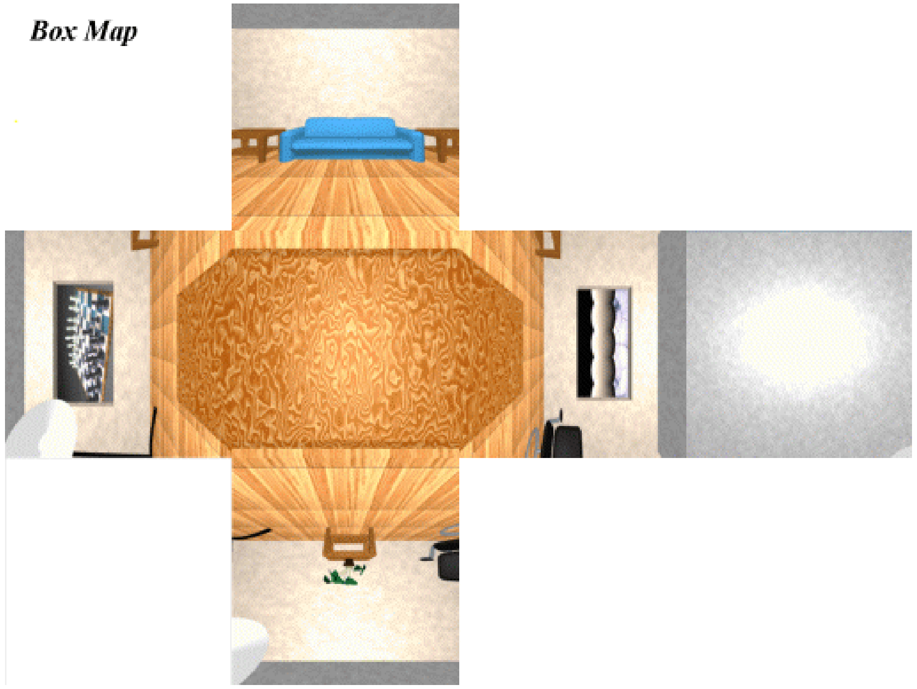


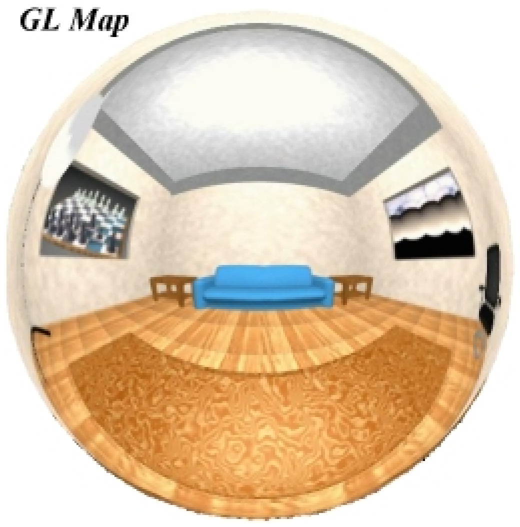
This was a very special purpose hack in OpenGL, however, we have it to thank for a lot of the flexibility in today’s graphics hardware… this hack was the genesis of programmable vertex shading.

What’s the Best Map?

- A sphere map is not the only representation choice for environment maps. There are alternatives, with more uniform sampling properties, but they require different normal-to-texture mapping functions.





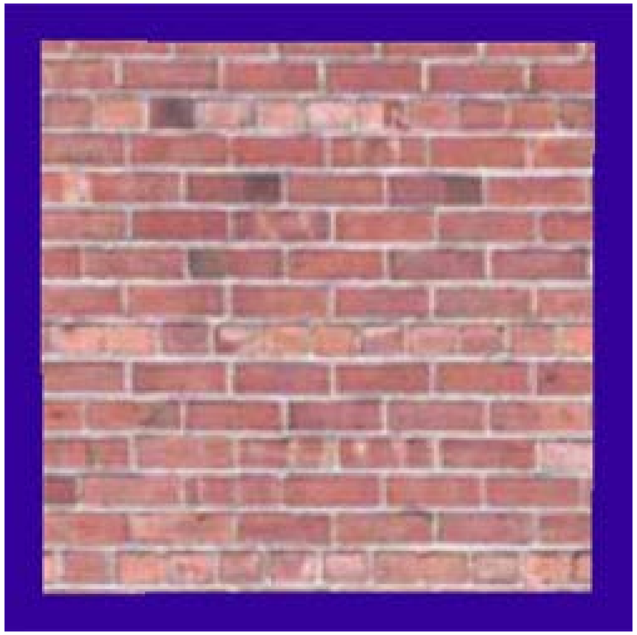
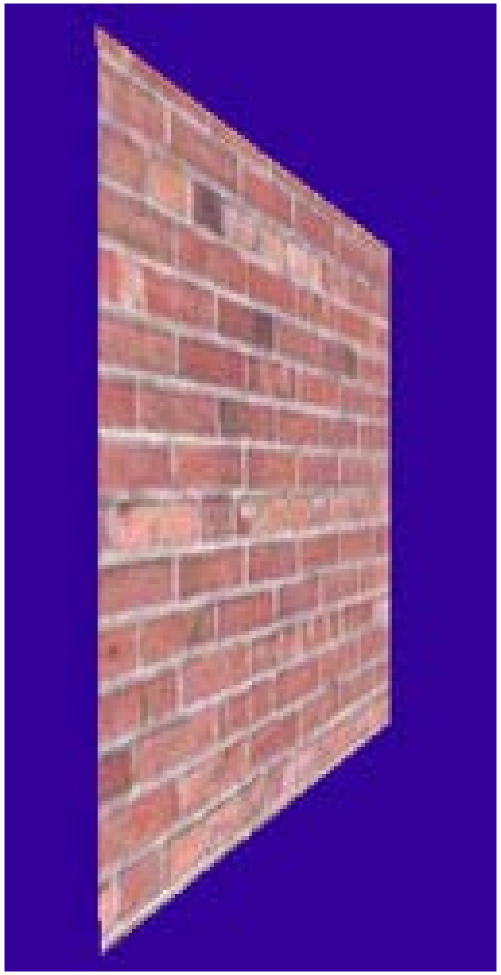


Questions?

Image by Henrik Wann Jensen
Environment map by Paul Debevec

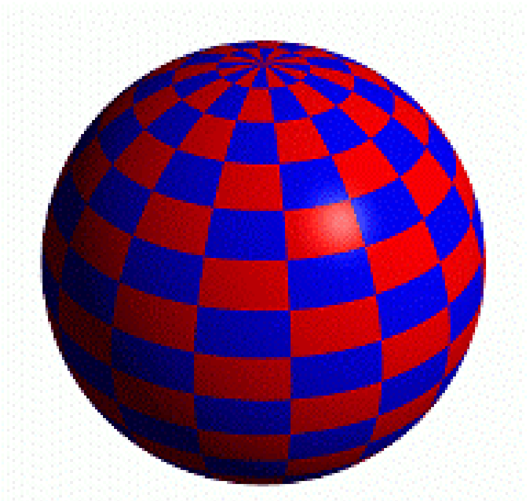
Bump Mapping

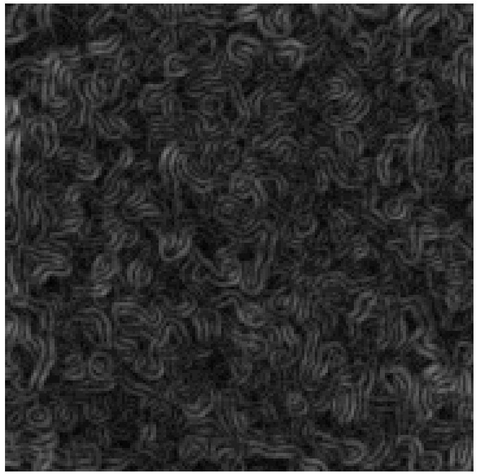
- What's Missing?
  - What's the difference between a real brick wall and a photograph of the wall texture-mapped onto a plane?
  - What happens if we change the lighting or the camera position?

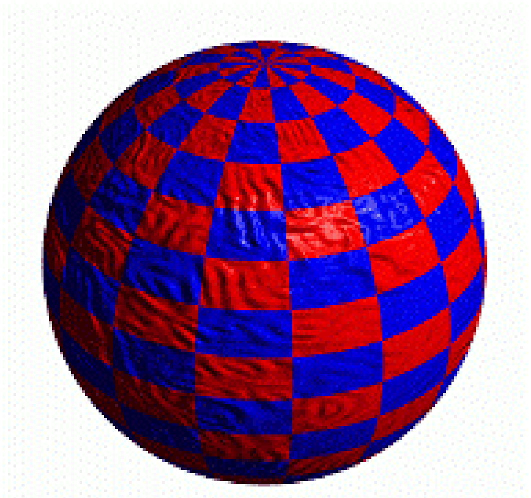



Bump Mapping

- Textures can be used to alter the surface normals of an object. This does not actual shape of the surface -- we are only shading it as if it were a different shape! This technique is called ***bump mapping***.
- The texture map is treated as a single-valued height function. The value of the function is not actually used, just its partial derivatives. The partial derivatives tell how to alter the true surface normal at each point on the surface to make the object appear as if it were deformed by the height function.

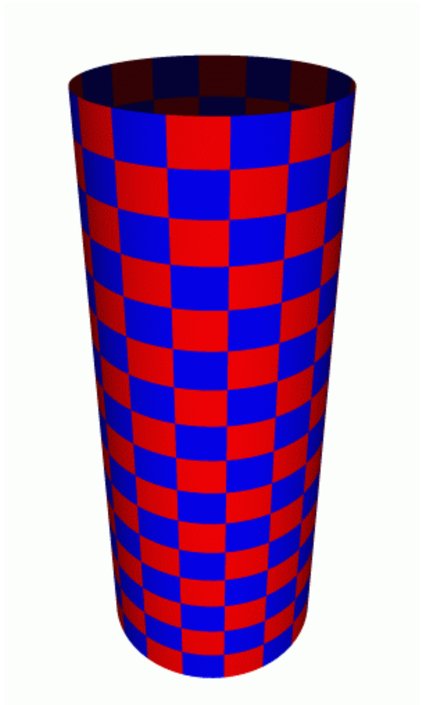

Sphere w/Diffuse Texture

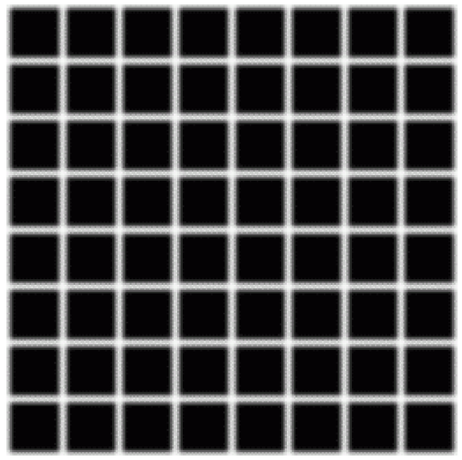

Swirly Bump Map

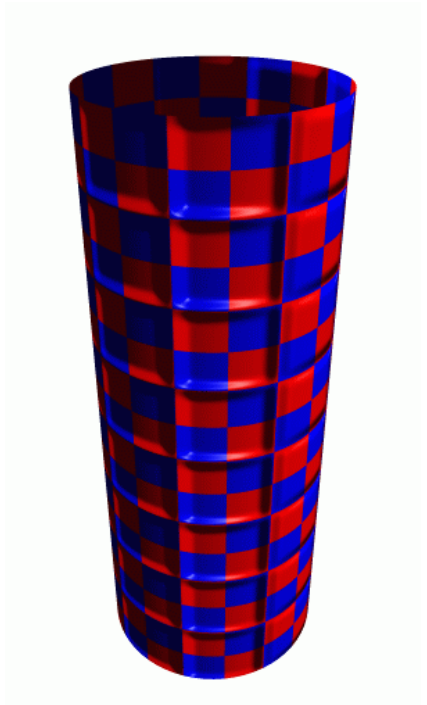

Sphere w/Diffuse Texture & Bump Map

Another Bump Map Example

- 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).

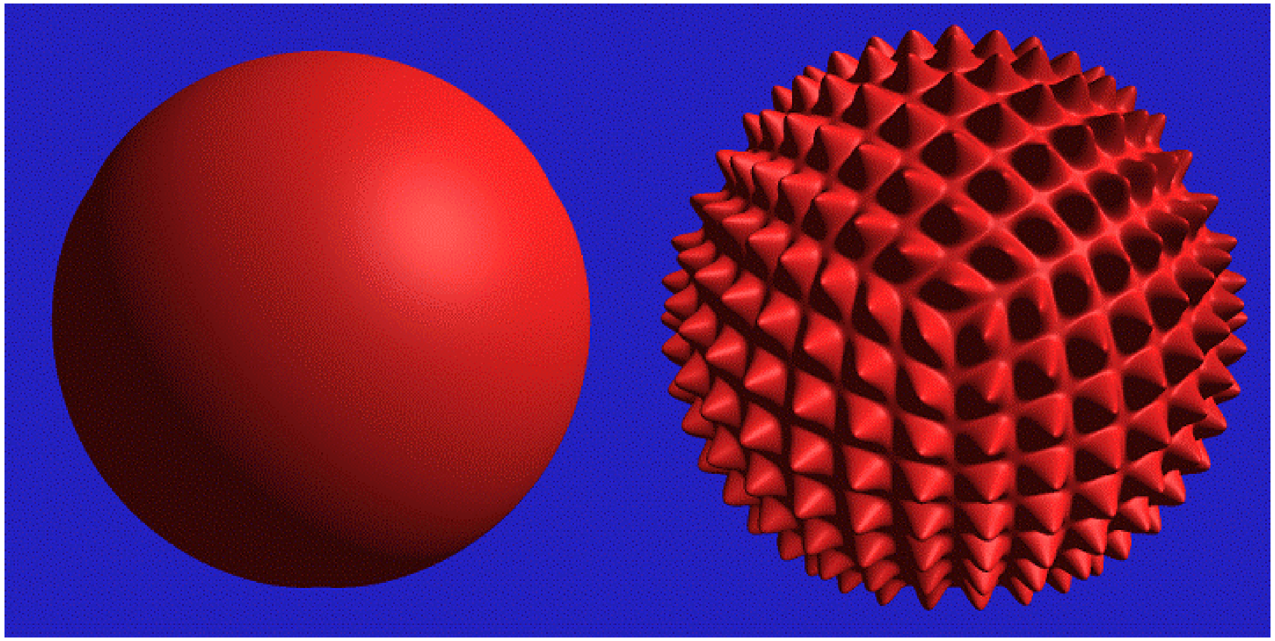

Cylinder w/Diffuse Texture


Bump Map

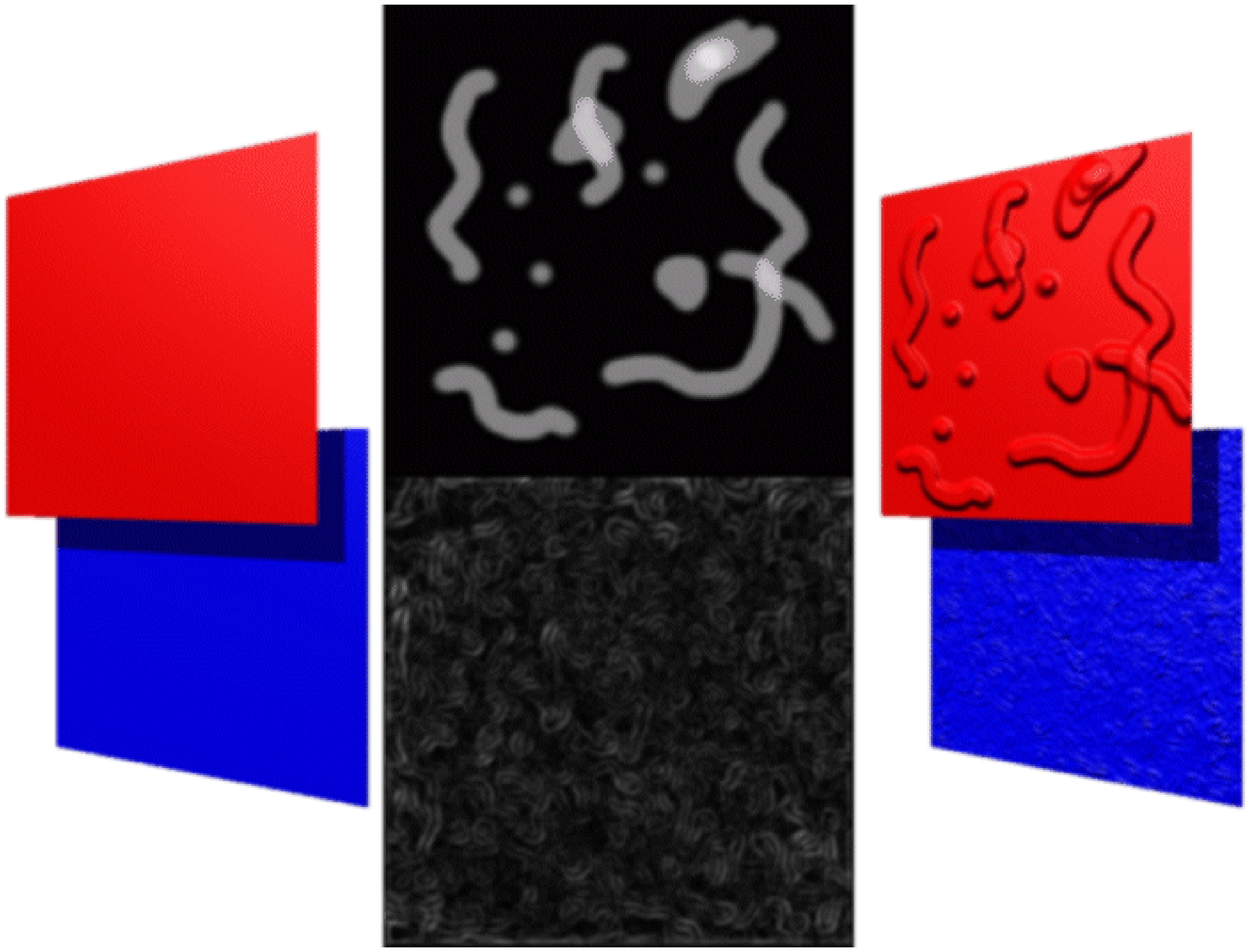

Cylinder w/Diffuse Texture & Bump Map

Displacement Mapping

- Texture maps can be used to actually move surface points.
- This is called ***displacement mapping***. How is this fundamentally different than bump mapping?



Questions?



Textures in GLSL

- Textures are just uniforms of type sampler2D
- Tell GLSL that you want one of these sampler2Ds to be GL_TEXTUREi by setting the corresponding uniform to i

```
uniform sampler2D dayTexture;
uniform sampler2D nightTexture;
```

Multi-Texturing

- Shaders will often have uses for multiple textures, how do you bind them?
- How do you specify multiple texture coordinates for a vertex?

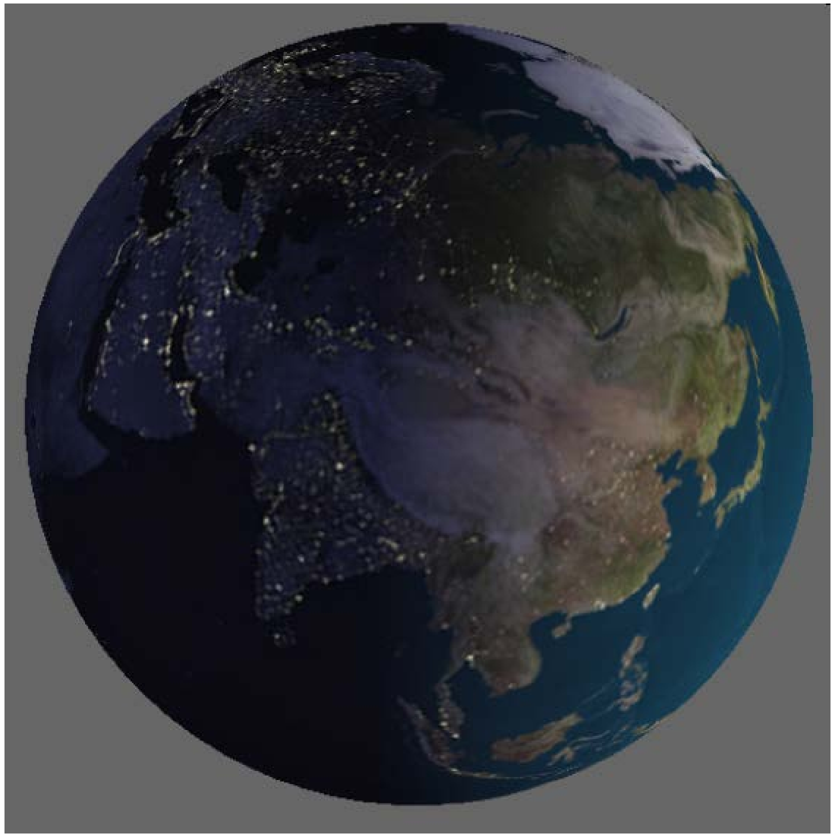
Fragment Program with Multiple Textures

```
uniform sampler2D dayTexture;
uniform sampler2D nightTexture;

void main ()
{
     …
     vec4 dayColor = texture3D (dayTexture, gl_TexCoord[0].xy);
     …
}
```

Multi-Texture: Globe

- How will you render this?

Night texture



Day texture

Night and Day

- Use one texture for day, one for night
- Use the same texture coordinates for both texture
- Components
  - Vertex program
  - Fragment program
  - OpenGL application

Night and Day
Vertex Program

```
/* this vector will store the normal in eye coordinates */
varying vec3 normal;

void main ()
{

    /* store the transformed normal in normal */
    normal = gl_NormalMatrix * gl_Normal;

    /* pass texture coordinate to the fragment program */
    gl_TexCoord[0] = glMultiTexCoord0;

    /* use the standard OpenGL transformation matrix */
    gl_Position = ftransform ();

}
```

Night and Day Fragment Program

```
/* this vector will store the normal in eye coordinates */
varying vec3 normal;

/* the fraction of overlap between textures */
uniform float overlap;

/* uniform texture map for the day’ s texture */
uniform sampler2D dayTexture;

/* uniform texture map for the night’ s */
uniform sampler2D nightTexture;

void main ()
{

     /* make sure we have unit normal while interpolating */
     vec3 N = normalize (normal);

     /* unit light direction, (assume) stored in eye coordinates */
     vec3 L = normalize (gl_LightSoure[0].position.xyz);

     …

}
```

Night and Day Fragment Program (Cont.)

```
float NdotL = dot (N,L);


/* calculate the day’ s and night’ s weights */
vec4 dayWeight =
     gl_FronetMaterial.diffuse * gl_LightSoure[0].diffuse *
     max (NdotL + overlap, 0.0) / (1.0 + overlap);
vec4 nightWeight = max (overlap - dayWeight, 0.0) / overlap;


/* sample day’ s color */
vec4 dayColor = texture2D (dayTexture, gl_TexCoord[0].xy);


/* sampel explosion color */
vec4 nightColor = texture2D (nightTexture, gl_TexCoord[0].xy);


/* set output color to the weighted combination */
gl_FragColor =
      dayColor  *  dayWeight +
2.5 * nightColor * nightWeight;


/* make alpha always 1 */
gl_FragColor.w = 1.0;
```

Night and Day OpenGL Program

```
// bind and enable texture unit 0, set it to be the day’ s texture map
glActiveTexture (GL_TEXTURE0);
glBindTexture (GL_TEXTURE_2D, dayTexture);
glEnable (GL_TEXTURE_2D);


// tell the program that earthTexture is in GL_TEXTURE0
Glunit dayLoc = glGetUniformLocation (program,  “dayTexture” );
glUniformi (dayLoc, 0);


// bind and enable texture unit 1, set it to be night’ s texture map
glActiveTexture (GL_TEXTURE1);
glBindTexture (GL_TEXTURE_2D, nightTexture);
glEnable (GL_TEXTURE_2D);


// tell the program that nightTexture is in GL_TEXTURE1
Glunit nightLoc = glGetUniformLocation (program,  “nightTexture” );
glUniformi (dayLoc, 1);


// draw the earth
TexturedSphere ();
```

Next Time…

- Ray tracing
  - Textbook Chapter 21-1
- Programming Assignment 3 is due on 11/9