Computer Graphics (CS 543)
Lecture 9 (Part 1): Environment Mapping (Reflections and Refractions)

Prof Emmanuel Agu

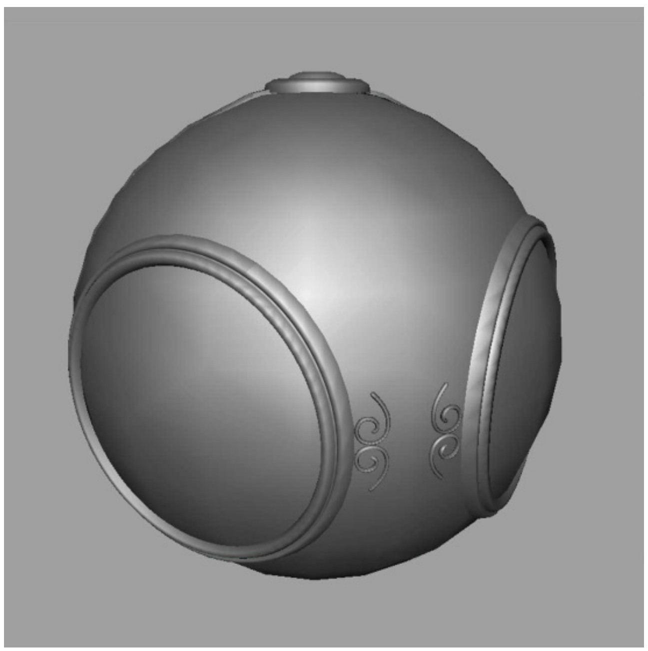
(Adapted from slides by Ed Angel)

Computer Science Dept.

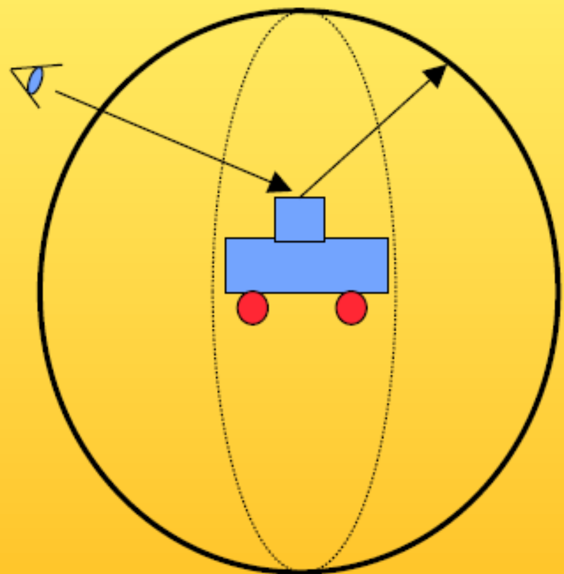
Worcester Polytechnic Institute (WPI)

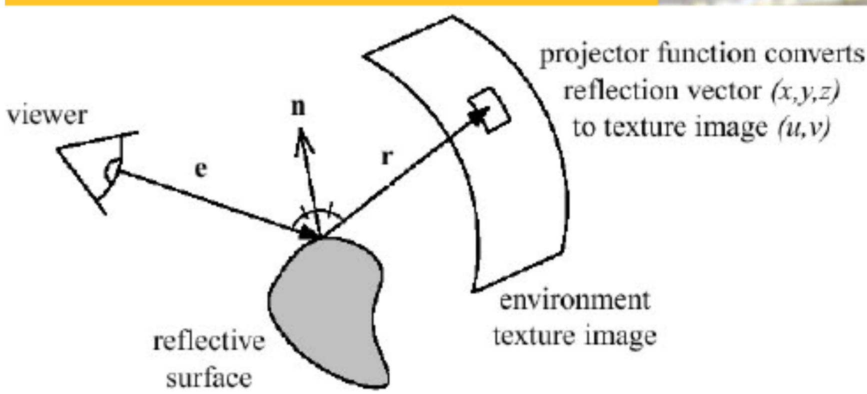
Environment Mapping

- Used to create appearance of **reflective** and **refractive** surfaces without ray tracing which requires global calculations



Environment mapping

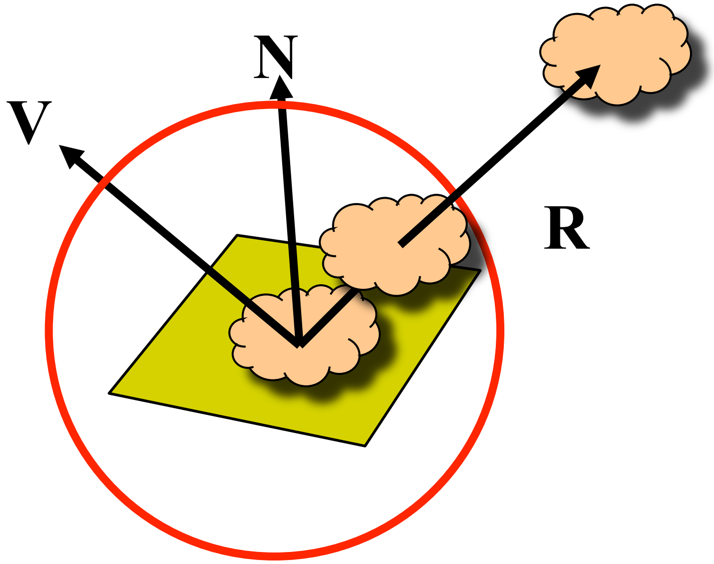




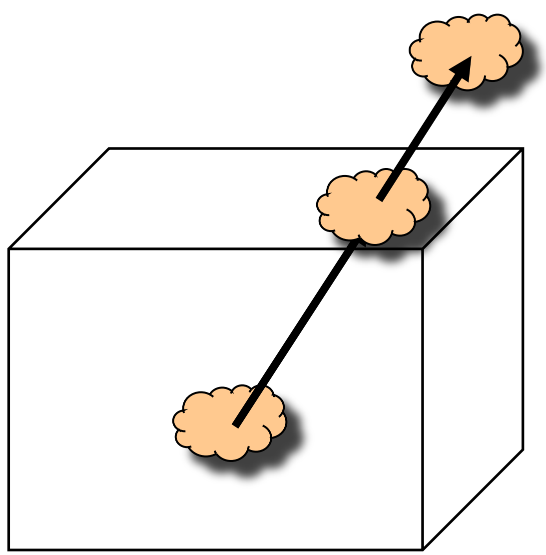
Types of Environment Maps

- Assumes environment infinitely far away
- Options: Store "object's environment as

a) Sphere around object (sphere map)



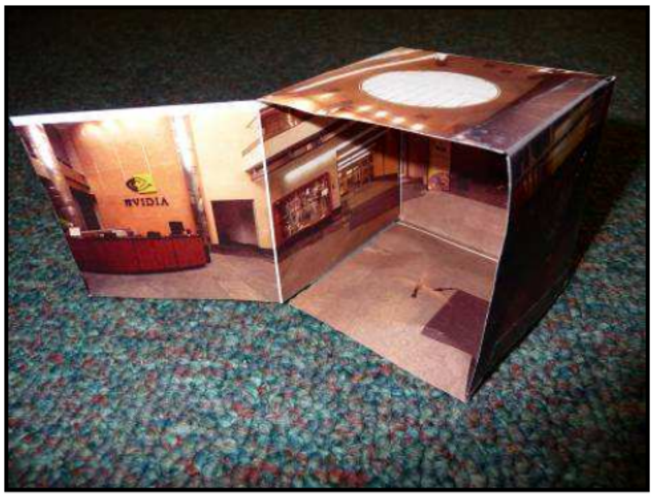
b) Cube around object (cube map)

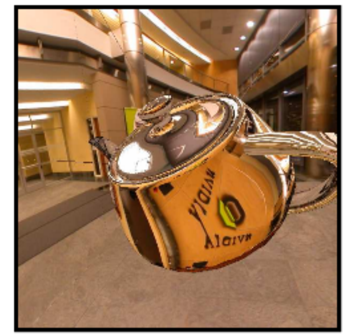


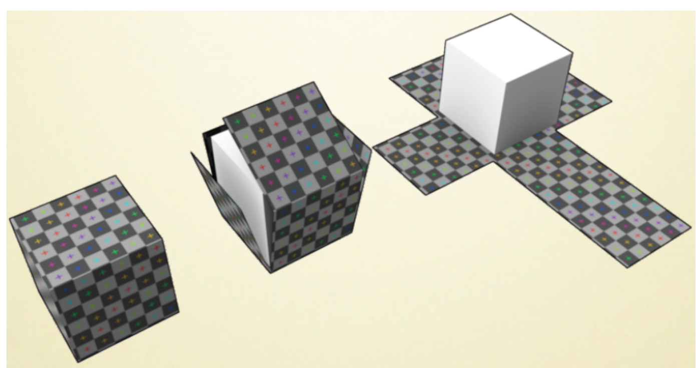
- OpenGL supports **cube maps** and **sphere maps**

Cube Map

- Stores "**environment**" around objects as 6 sides of a cube (1 texture)

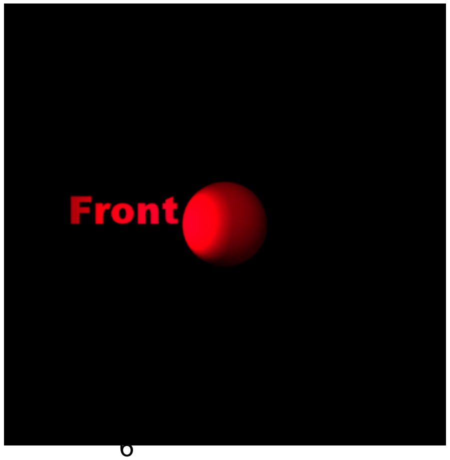


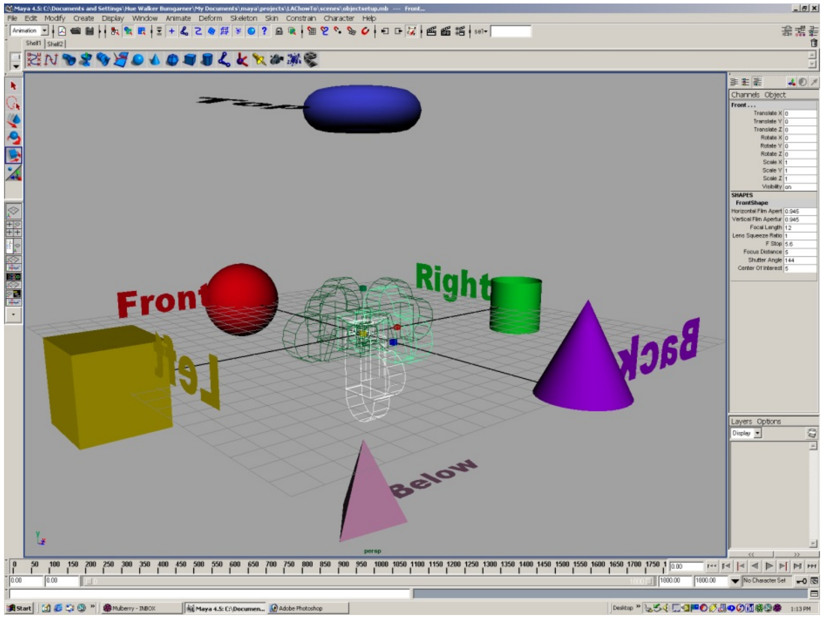




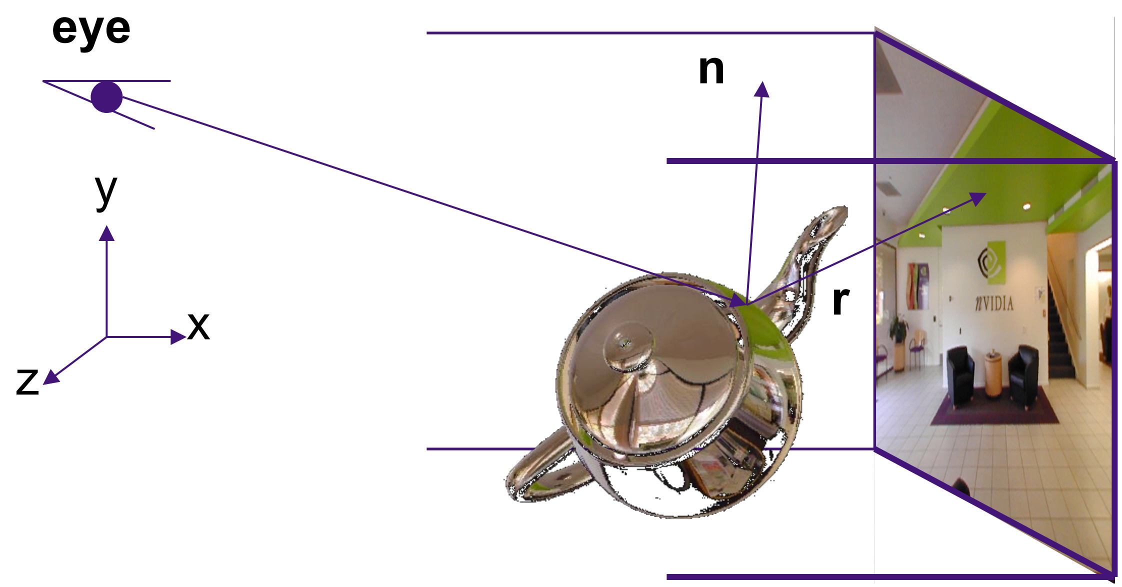
Forming Cube Map

- Use 6 cameras directions from scene center
    - each with a 90 degree angle of view





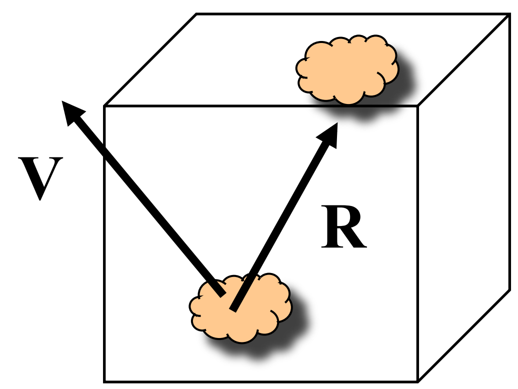
Reflection Mapping



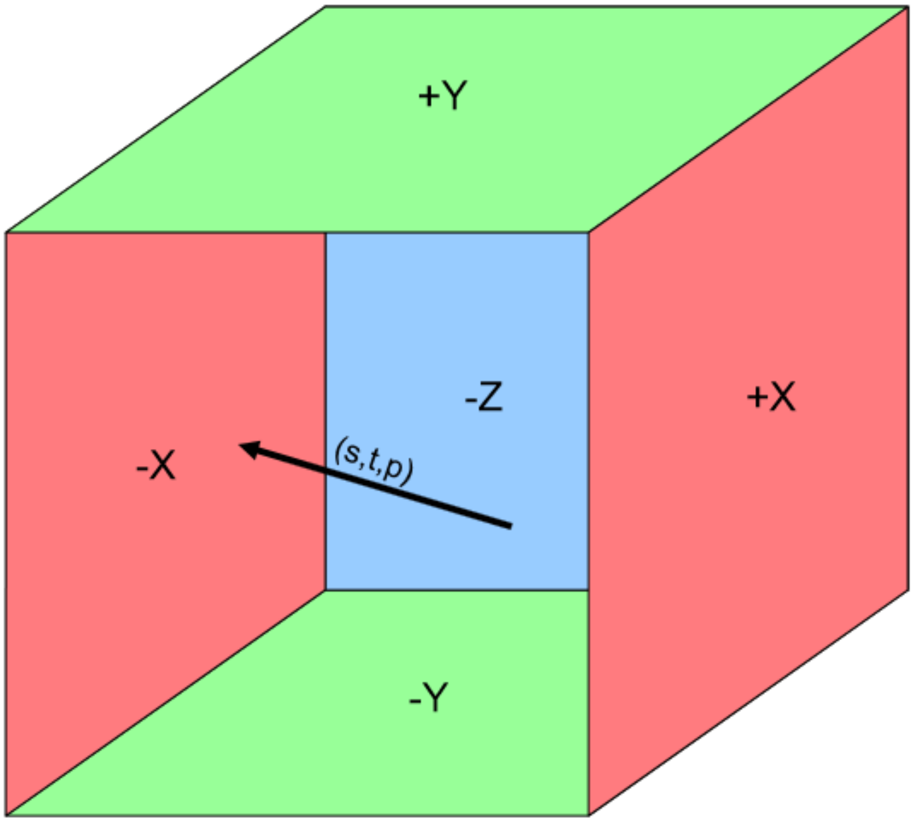
- Need to compute reflection vector, $\mathbf{r}$
- Use $\mathbf{r}$ by for lookup
- OpenGL hardware supports cube maps, makes lookup easier

Indexing into Cube Map

- Compute **R** = 2(**N·V**)**N**-**V**
- Object at origin
- Use **largest magnitude component** of R to determine face of cube
- Other 2 components give texture coordinates



Cube Map Texture Lookup:
Given an (s,t,p) direction vector , what (r,g,b) does that correspond to?



- Let L be the texture coordinate of (s, t, and p) with the largest magnitude
- L determines which of the 6 2D texture “walls” is being hit by the vector (-X in this case)
- The texture coordinates in that texture are the remaining two texture coordinates divided by L: (a/L,b/L)

Built-in GLSL functions

```
vec3 ReflectVector = reflect( vec3 eyeDir, vec3 normal );

vec3 RefractVector = refract( vec3 eyeDir, vec3 normal, float Eta );
```

Example

- **R** = (-4, 3, -1)
- Same as **R** = (-1, 0.75, -0.25)
- Use face x = -1 and y = 0.75, z = -0.25
- Not quite right since cube defined by x, y, z = ± 1 rather than [0, 1] range needed for texture coordinates
- Remap by $s = \frac{1}{2} + \frac{1}{2} y$, $t = \frac{1}{2} + \frac{1}{2} z$
- Hence, s =0.875, t = 0.375

Declaring Cube Maps in OpenGL

```
glTextureMap2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, level, rows,
               columns, border, GL_RGBA, GL_UNSIGNED_BYTE, image1)
```

- Repeat similar for other 5 images (sides)
- Make 1 texture object from 6 images
- Parameters apply to all six images. E.g

```
glTexParameteri( GL_TEXTURE_CUBE_MAP,
                 GL_TEXTURE_MAP_WRAP_S,  GL_REPEAT)
```

- **Note:** texture coordinates are in 3D space (s, t, r)

Cube Map Example (init)

```
// colors for sides of cube
   GLubyte red[3] = {255, 0, 0};
   GLubyte green[3] = {0, 255, 0};
   GLubyte blue[3] = {0, 0, 255};
   GLubyte cyan[3] = {0, 255, 255};
   GLubyte magenta[3] = {255, 0, 255};
   GLubyte yellow[3] = {255, 255, 0};

  glEnable(GL_TEXTURE_CUBE_MAP);

// Create texture object
  glGenTextures(1, tex);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_CUBE_MAP, tex[0]);
```

You can also just load
6 pictures of environment

Cube Map (init II)

You can also just use
6 pictures of environment

```
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, red);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, green);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, blue);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, cyan);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, magenta);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z ,
                    0,3,1,1,0,GL_RGB,GL_UNSIGNED_BYTE, yellow);
glTexParameteri(GL_TEXTURE_CUBE_MAP,
                    GL_TEXTURE_MAG_FILTER,GL_NEAREST);
```

Cube Map (init III)

```
GLuint texMapLocation;
GLuint tex[1];

texMapLocation = glGetUniformLocation(program, "texMap");
glUniform1i(texMapLocation, tex[0]);
```

Connect texture map (tex[0])
to variable texMap in fragment shader
(texture mapping done in frag shader)

Adding Normals

```
void quad(int a, int b, int c, int d)
{
    static int i =0;

    normal = normalize(cross(vertices[b] - vertices[a],
      vertices[c]  - vertices[b]));


    normals[i] = normal;
    points[i] = vertices[a];
    i++;

// rest of data
```

Vertex Shader

```
out vec3 R;
in vec4 vPosition;
in vec4 Normal;
uniform mat4 ModelView;
uniform mat4 Projection;

void main() {
   gl_Position = Projection*ModelView*vPosition;
   vec4 eyePos  = vPosition;                 // calculate view vector V
   vec4 NN = ModelView*Normal;      // transform normal
   vec3 N =normalize(NN.xyz);          // normalize normal
   R = reflect(eyePos.xyz, N);           // calculate reflection vector R
}
```

Fragment Shader

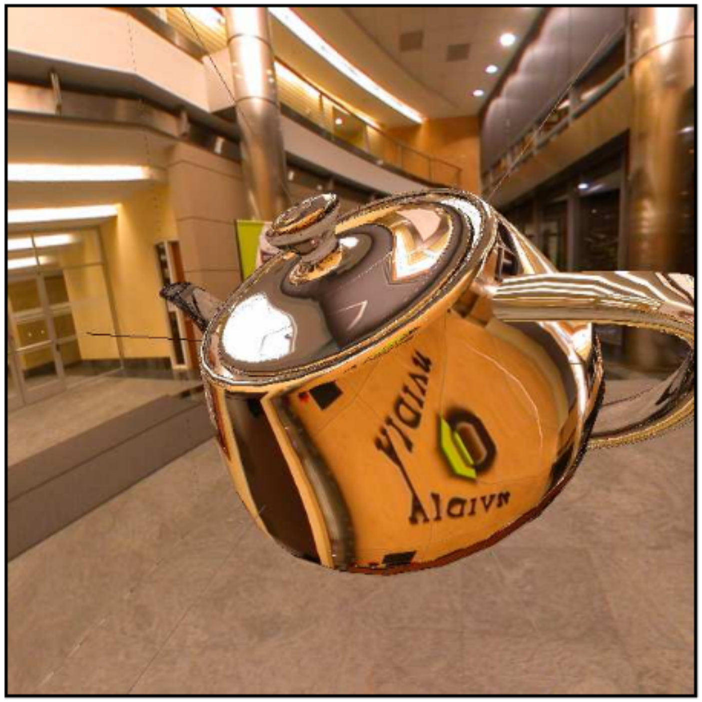
```
in vec3 R;
uniform samplerCube texMap;

void main()
{
    vec4 texColor = textureCube(texMap, R);   // look up texture map using R

    gl_FragColor = texColor;
}
```

Refraction using Cube Map

- Can also use cube map for refraction (transparent)



Reflection

Refraction

Reflection vs Refraction

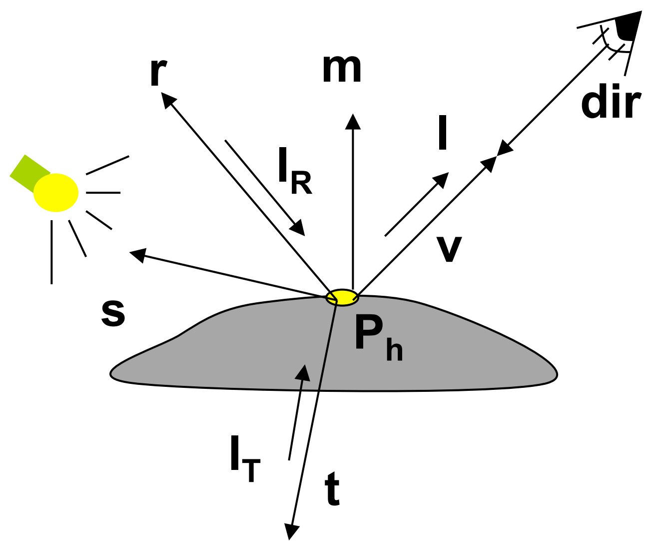
Reflection

Refraction

Reflection and Refraction

- At each vertex

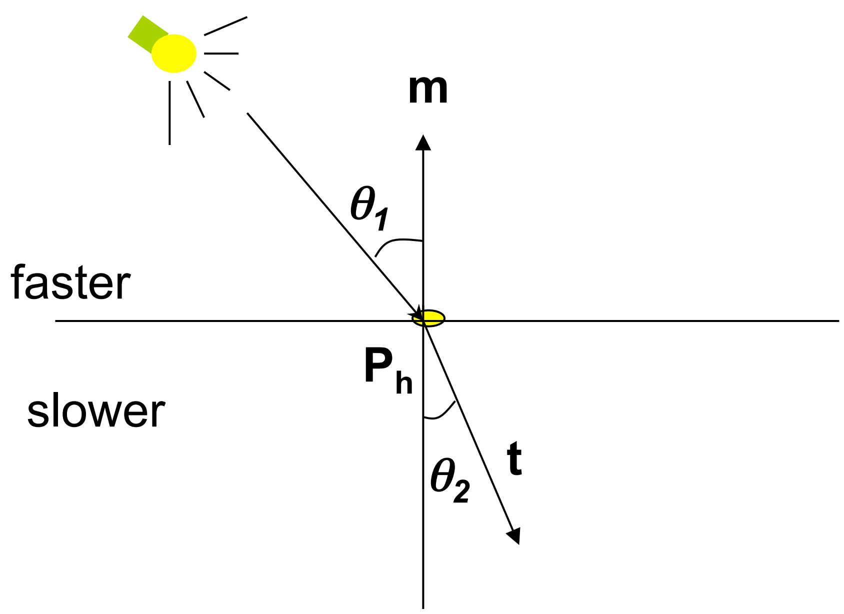
$$I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran}$$



- Refracted component I_T is along transmitted direction **t**

Finding Transmitted (Refracted) Direction

- Transmitted direction obeys **Snell's law**
- Snell's law: relationship holds in diagram below



$$\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}$$

c_1, c_2 are speeds of light in medium 1 and 2

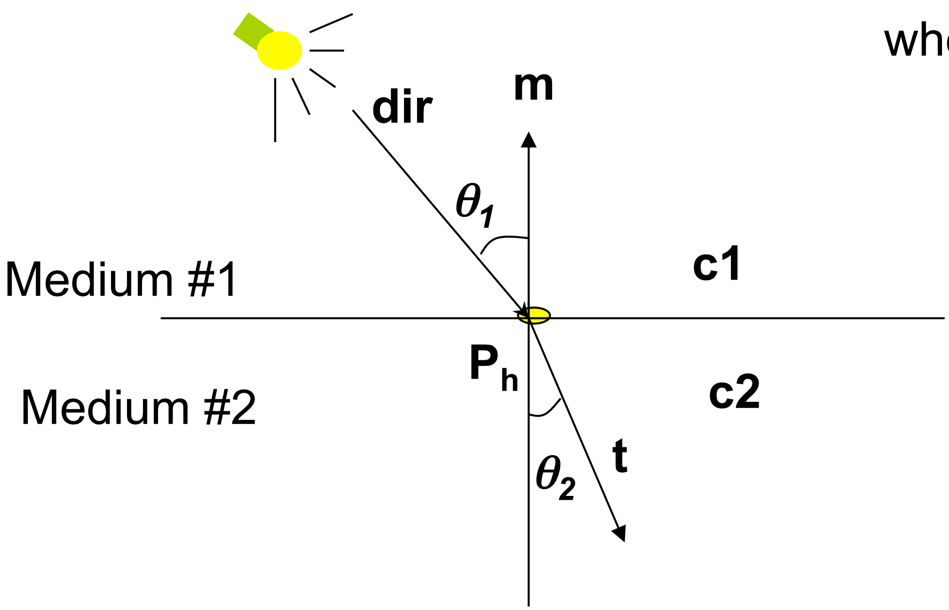
Finding Transmitted Direction

- If ray goes from faster to slower medium, ray is bent **towards** normal
- If ray goes from slower to faster medium, ray is bent **away** from normal
- c1/c2 is important. Usually measured for medium-to-vacuum. E.g water to vacuum
- Some measured relative c1/c2 are:
  - Air: 99.97%
  - Glass: 52.2% to 59%
  - Water: 75.19%
  - Sapphire: 56.50%
  - Diamond: 41.33%

Transmission Angle

- Vector for transmission angle can be found as

$$\mathbf{t} = \frac{c_2}{c_1}\mathbf{dir} + \left(\frac{c_2}{c_1}(\mathbf{m} \bullet \mathbf{dir}) - \cos(\theta_2)\right)\mathbf{m}$$



where

$$\cos(\theta_2) = \sqrt{1 - \left(\frac{c_2}{c_1}\right)\left(1 - (\mathbf{m} \bullet \mathbf{dir})^2\right)}$$

Refraction Vertex Shader

```
out vec3 T;
in vec4 vPosition;
in vec4 Normal;
uniform mat4 ModelView;
uniform mat4 Projection;

void main() {
   gl_Position = Projection*ModelView*vPosition;
   vec4 eyePos  = vPosition;                  // calculate view vector V
   vec4 NN = ModelView*Normal;       // transform normal
   vec3 N =normalize(NN.xyz);             // normalize normal
   T = refract(eyePos.xyz, N, iorefr);        // calculate refracted vector T
}
```

Was previously R = reflect(eyePos.xyz, N);

Refraction Fragment Shader

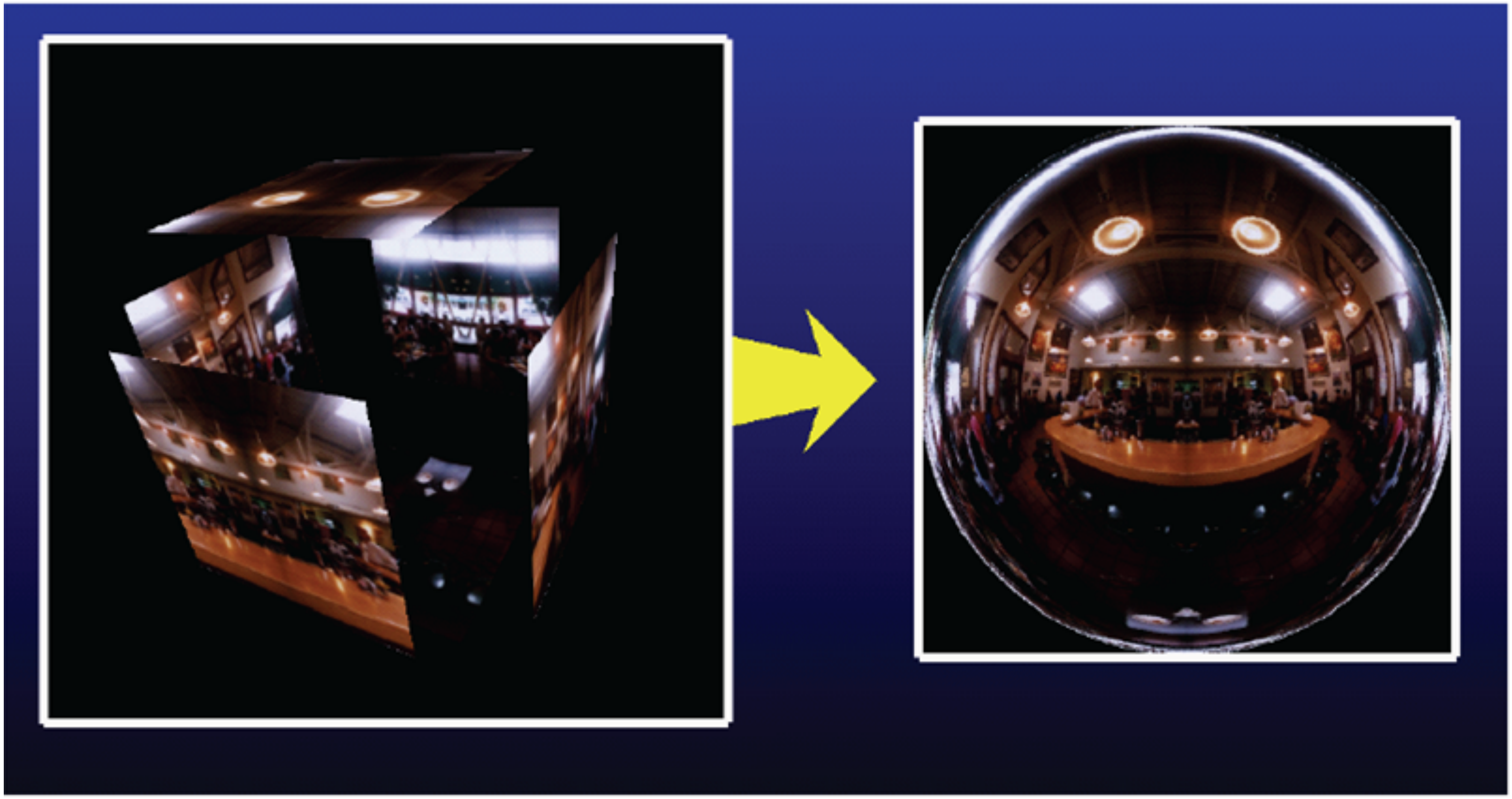
```
in vec3 T;
uniform samplerCube RefMap;

void main()
{
    vec4 refractColor = textureCube(RefMap, T);   // look up texture map using T
    refractcolor = mix(refractcolor, WHITE, 0.3);  // mix pure color with 0.3 white

   gl_FragColor = texColor;
}
```

Sphere Environment Map

- Cube can be replaced by a sphere (sphere map)



Sphere Mapping

- Original environmental mapping technique
- Proposed by Blinn and Newell
- Uses lines of longitude and latitude to map parametric variables to texture coordinates
- OpenGL supports sphere mapping
- Requires a circular texture map equivalent to an image taken with a fisheye lens

Sphere Map

- A sphere maps is basically a photograph of a reflective sphere in an environment

Paul DeBevec, www.debevec.org

Sphere map

- example

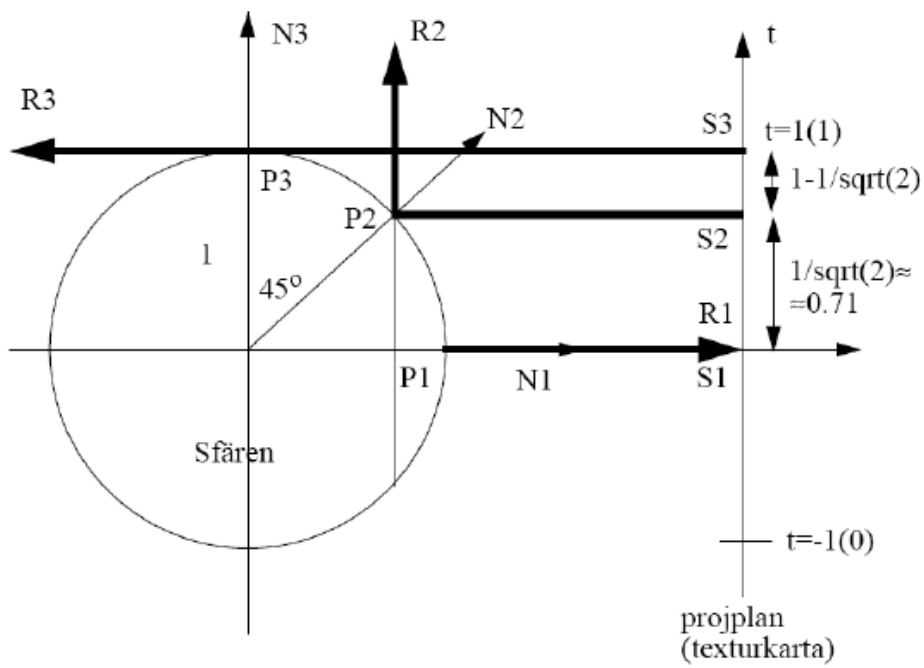
Sphere map
(texture)

Sphere map
applied on torus

Capturing a Sphere Map

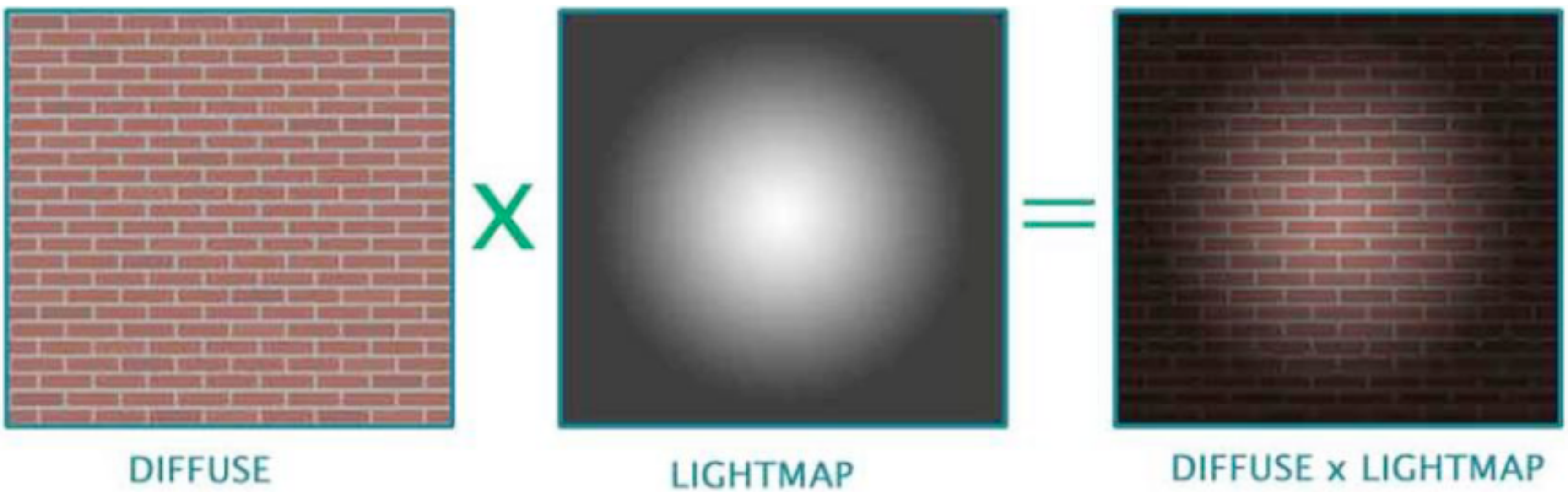
Sphere Map

- Infinitesimally small reflective sphere (infinitely far away)
  - i.e., orthographic view of a reflective unit sphere
- Create by:
  - Photographing metal sphere
  - Ray tracing
  - Transforming cube map to sphere map



For derivation of sphere map, see section 7.8 of your text

Light Maps

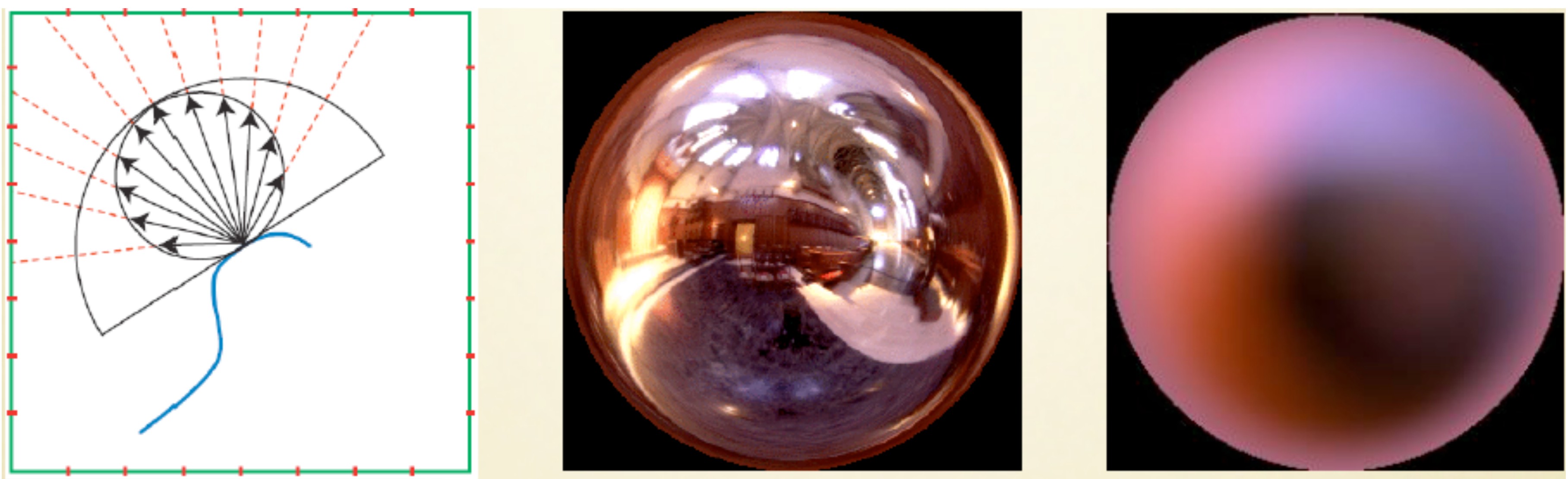


Specular Mapping

- Use a greyscale texture as a multiplier for the specular component

Irradiance Mapping

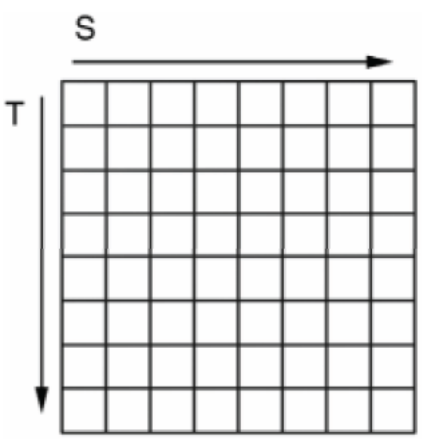
- You can reuse environment maps for diffuse reflections
- Integrate the map over a hemisphere at each pixel (basically blurs the whole thing out)

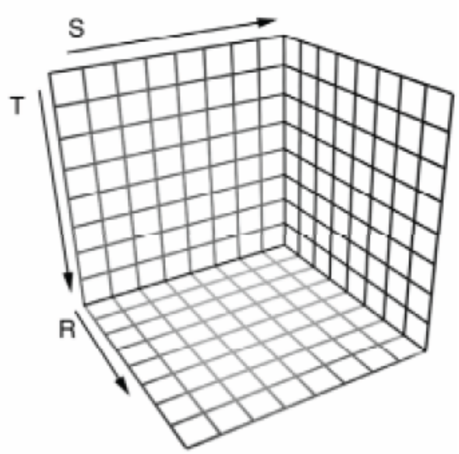


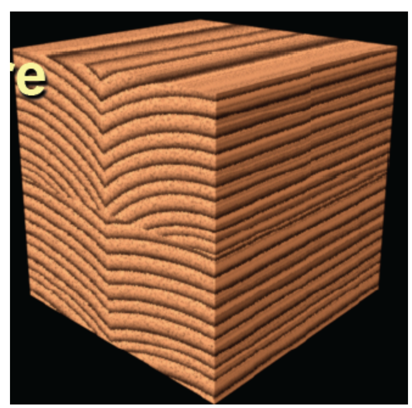
Irradiance Mapping Example

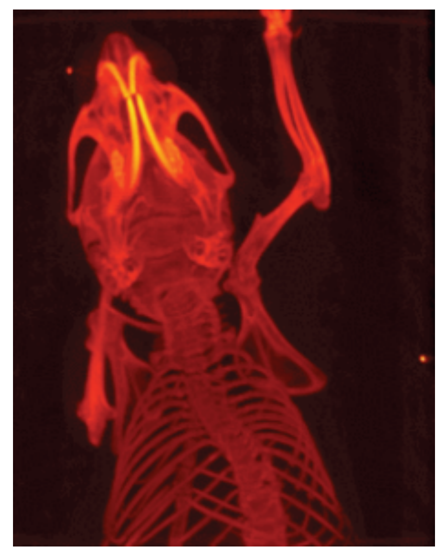
3D Textures

- 3D volumetric textures exist as well, though you can only render slices of them in OpenGL
- Generate a full image by stacking up slices in Z
- Used in visualization



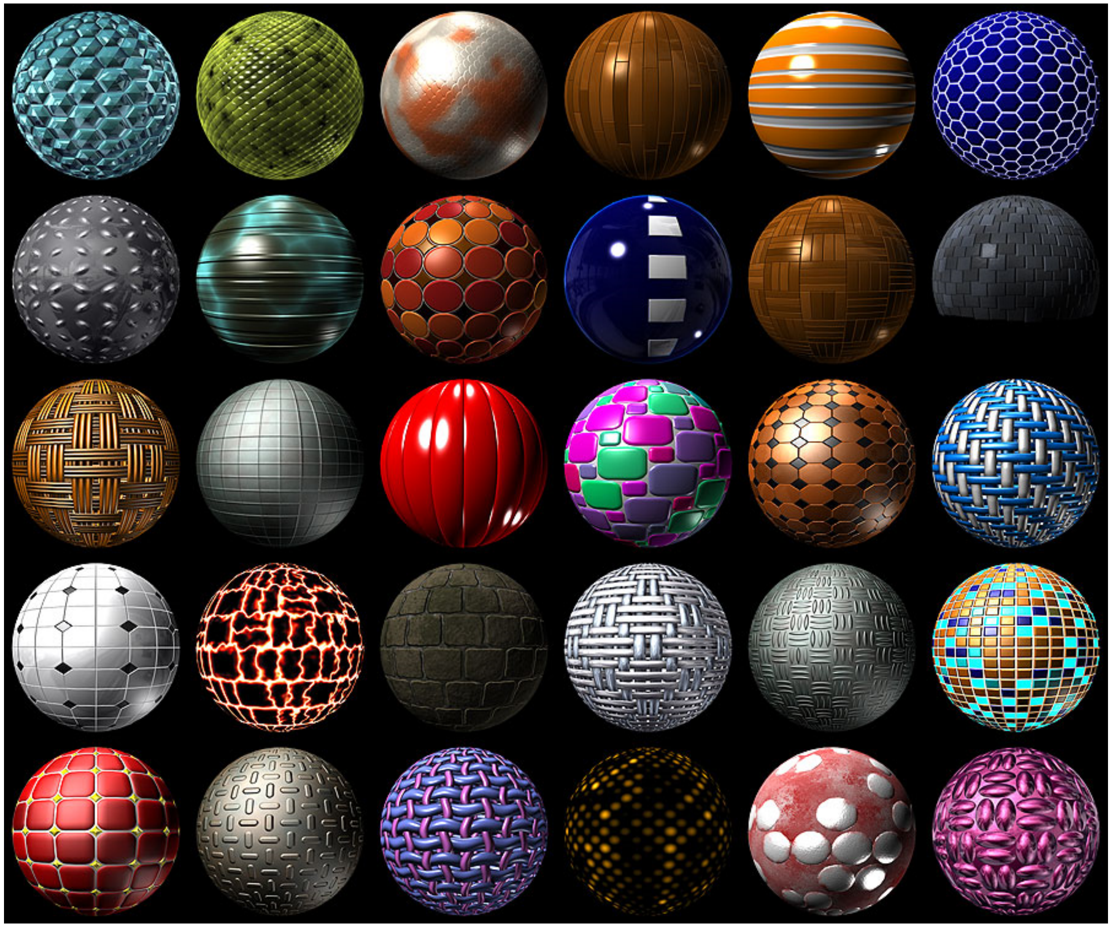






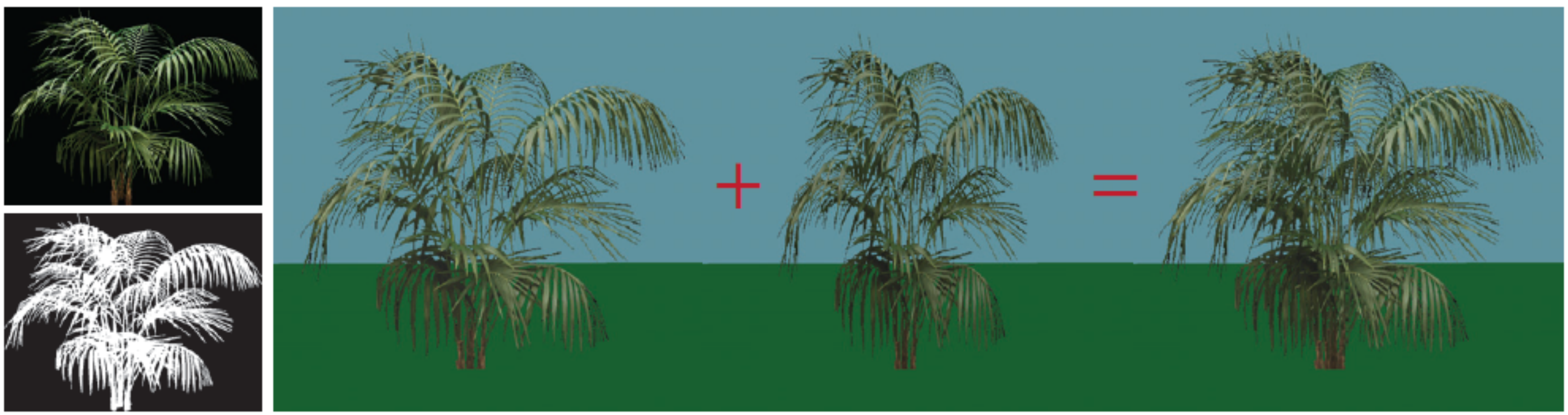
Procedural Texturing

- Math functions that generate textures



Alpha Mapping

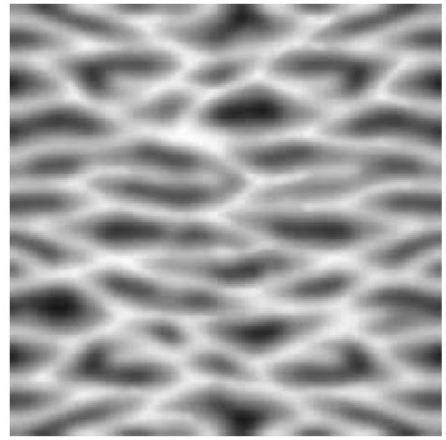
- Represent the alpha channel with a texture
- Can give complex outlines, used for plants

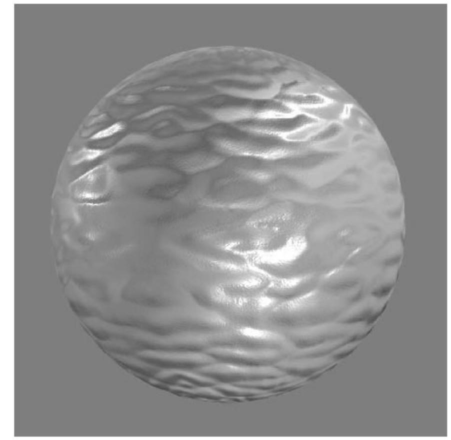


Render Bush
on 1 polygon

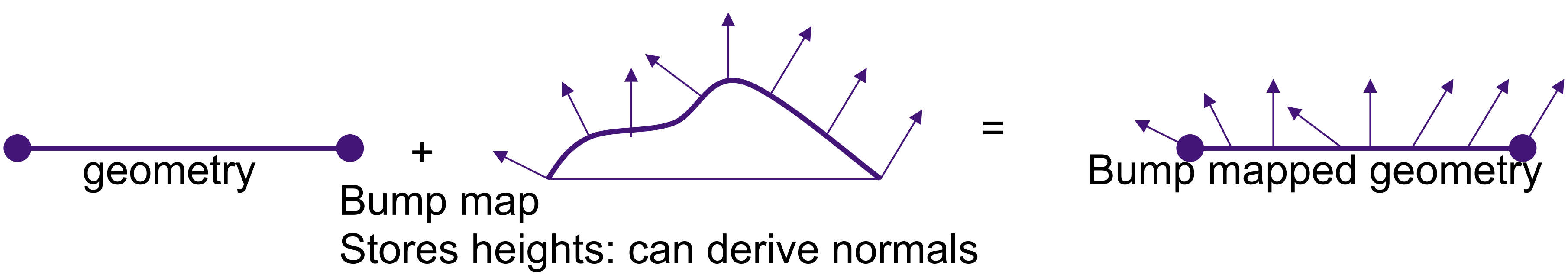
Render Bush
on polygon rotated
90 degrees

Bump mapping



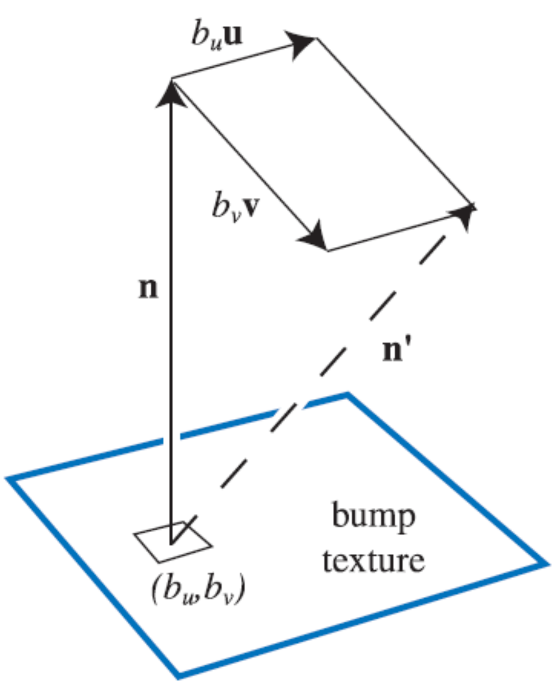


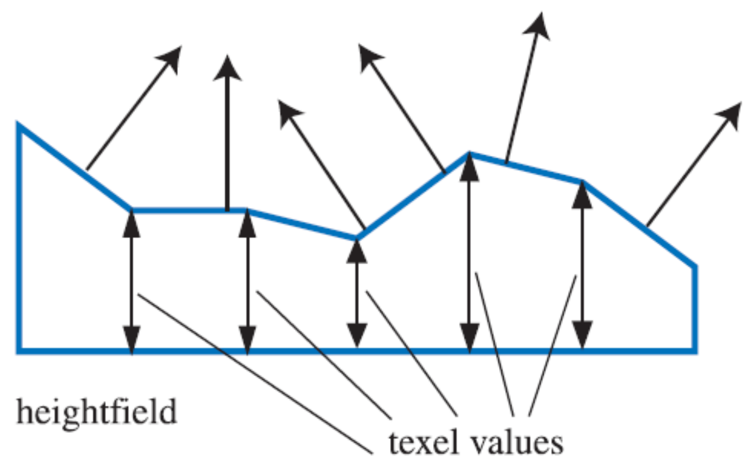
- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting



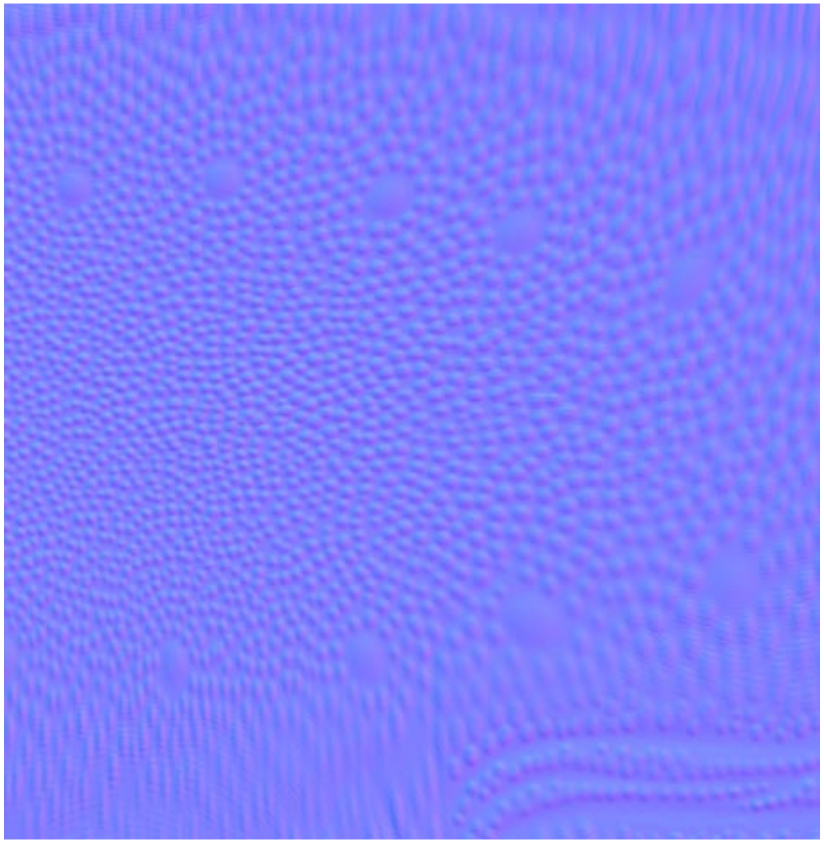
Bump mapping: Blinn's method

- Basic idea:
  - Distort the surface along the normal at that point
  - Magnitude is equal to value in heighfield at that location



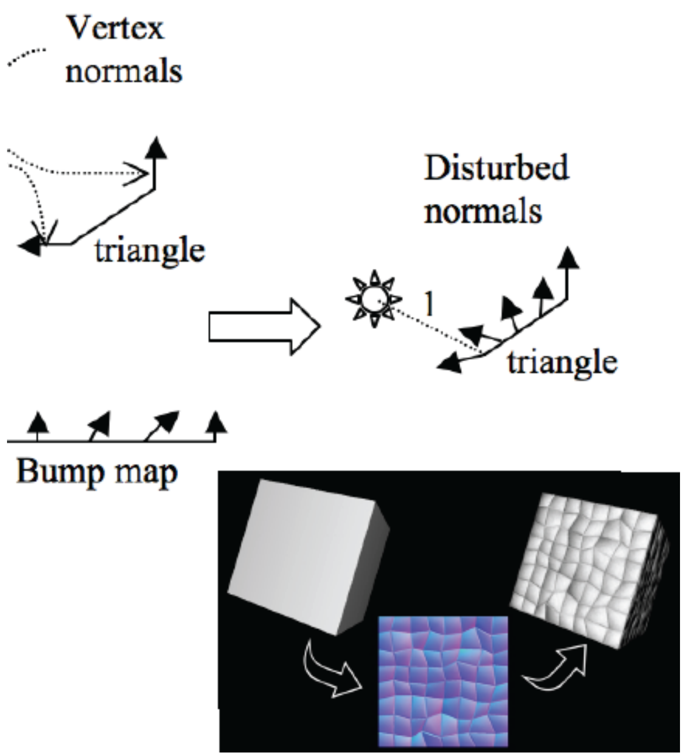


Bump mapping: examples

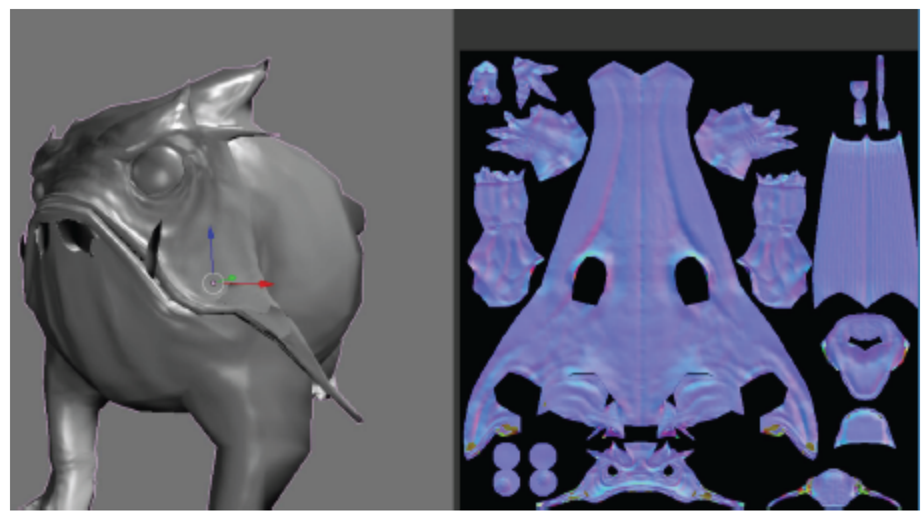


Bump Mapping Vs Normal Mapping

- **Bump mapping**
- (Normals $\mathbf{n}=(n_x, n_y, n_z)$ stored as *distortion of face orientation*. Same bump map can be tiled/ repeated and reused for many faces)

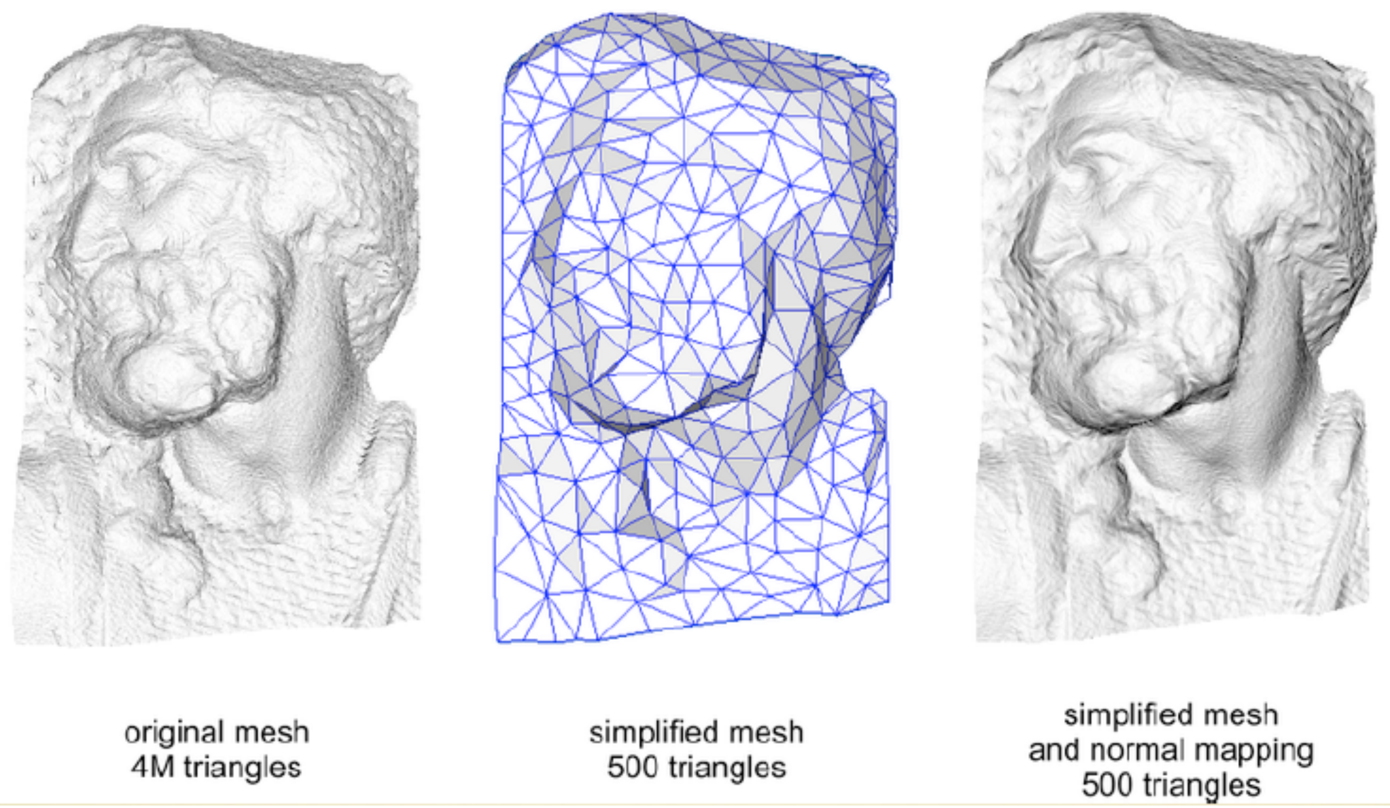


- **Normal mapping**
- Coordinates of normal (relative to tangent space) are encoded in color channels
- Normals stored include face orientation + plus distortion.)



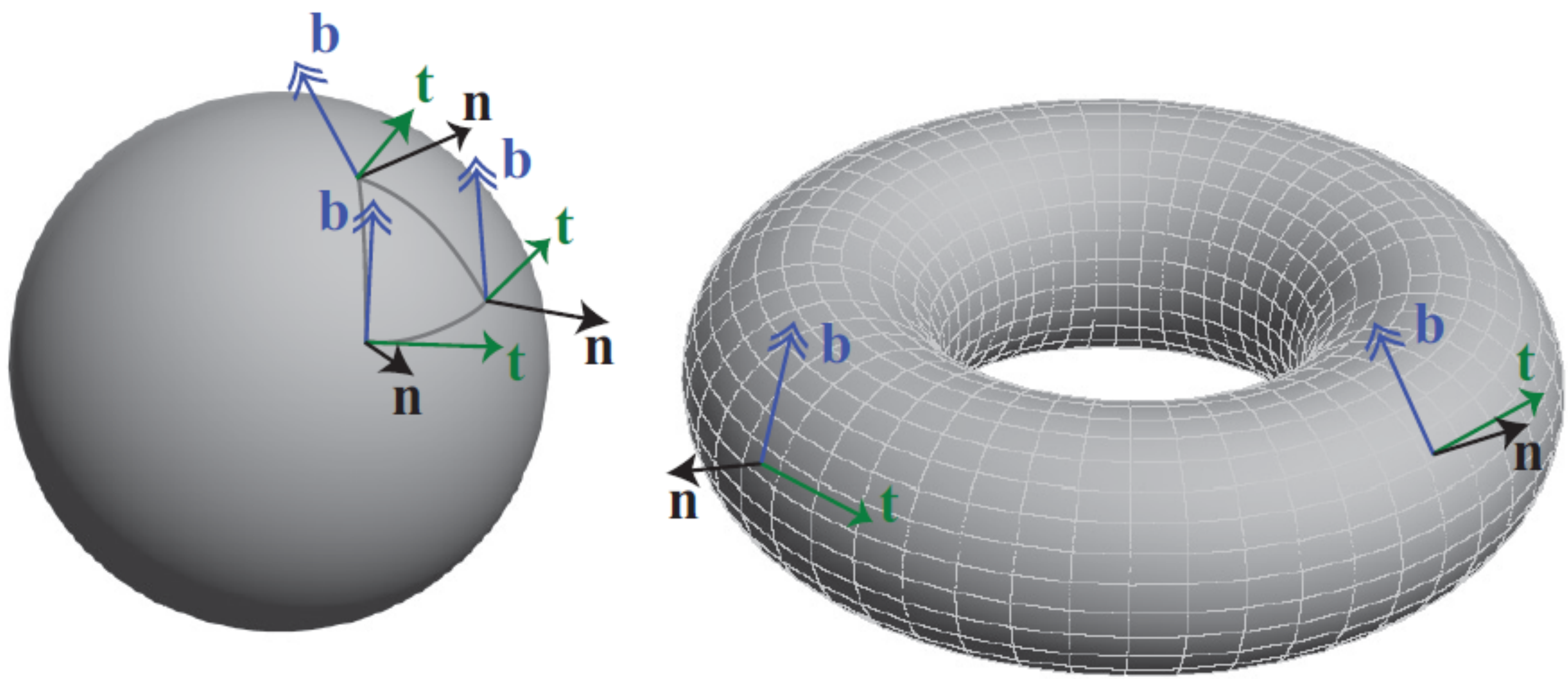
Normal Mapping

- Very useful for making low-resolution geometry look like it's much more detailed



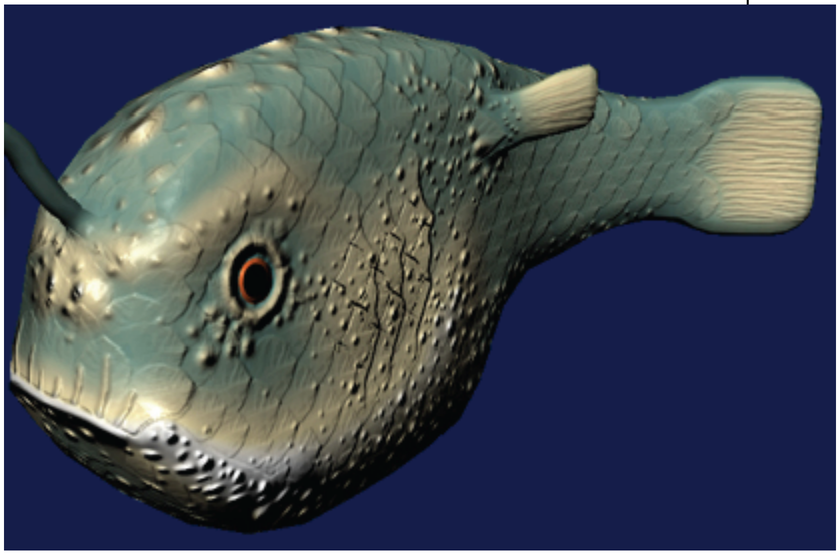
Tangent Space Vectors

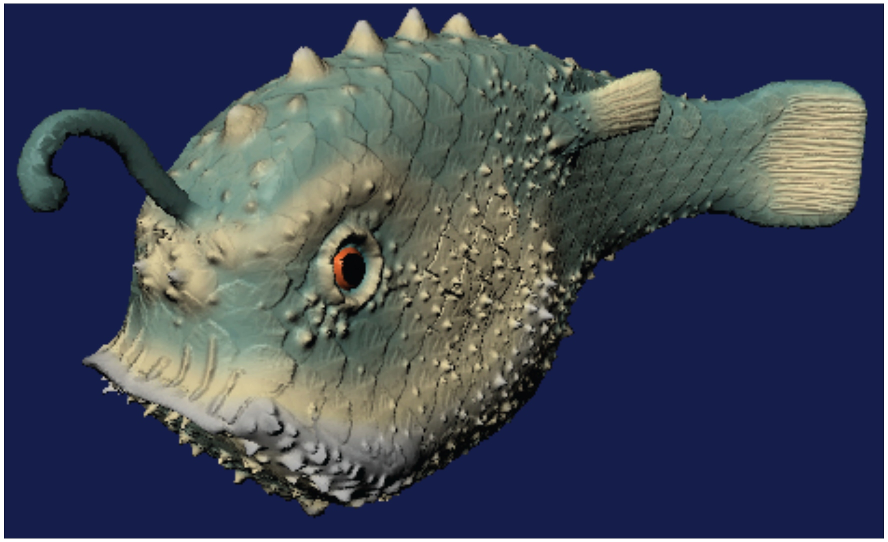
- Normals stored in local coordinate frame
- Need Tangent, normal and bi-tangent vectors



Displacement Mapping

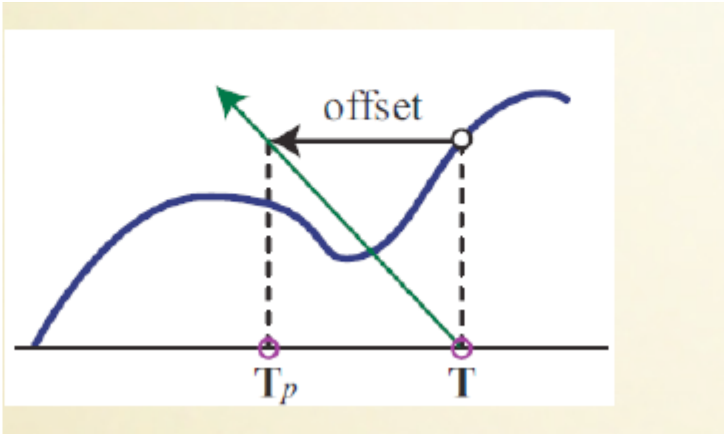
- Uses a map to displace the surface geometry at each position
- Offsets the position per pixel or per vertex
  - Offsetting per vertex is easy in vertex shader
  - Offsetting per pixel is architecturally hard





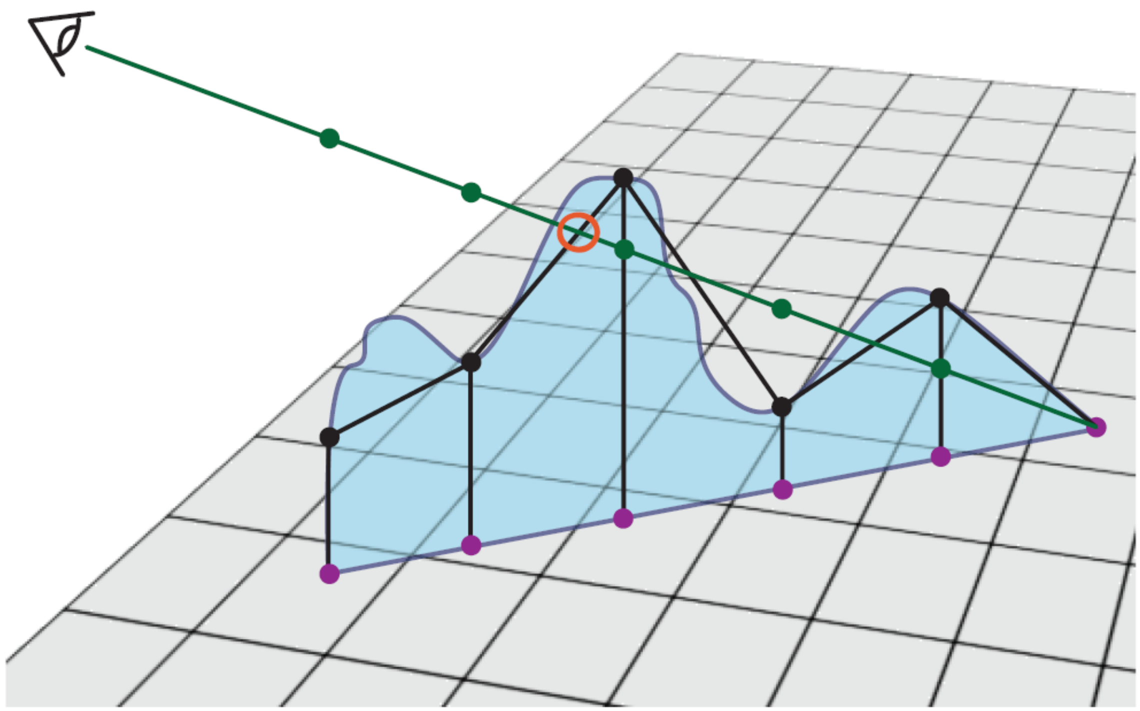
Parallax Mapping

- Normal maps increase lighting detail, but they lack a sense of depth when you get up close
- Parallax mapping
  - simulates depth/blockage of one part by another
  - Uses heightmap to offset texture value / normal lookup
  - Different texture returned after offset

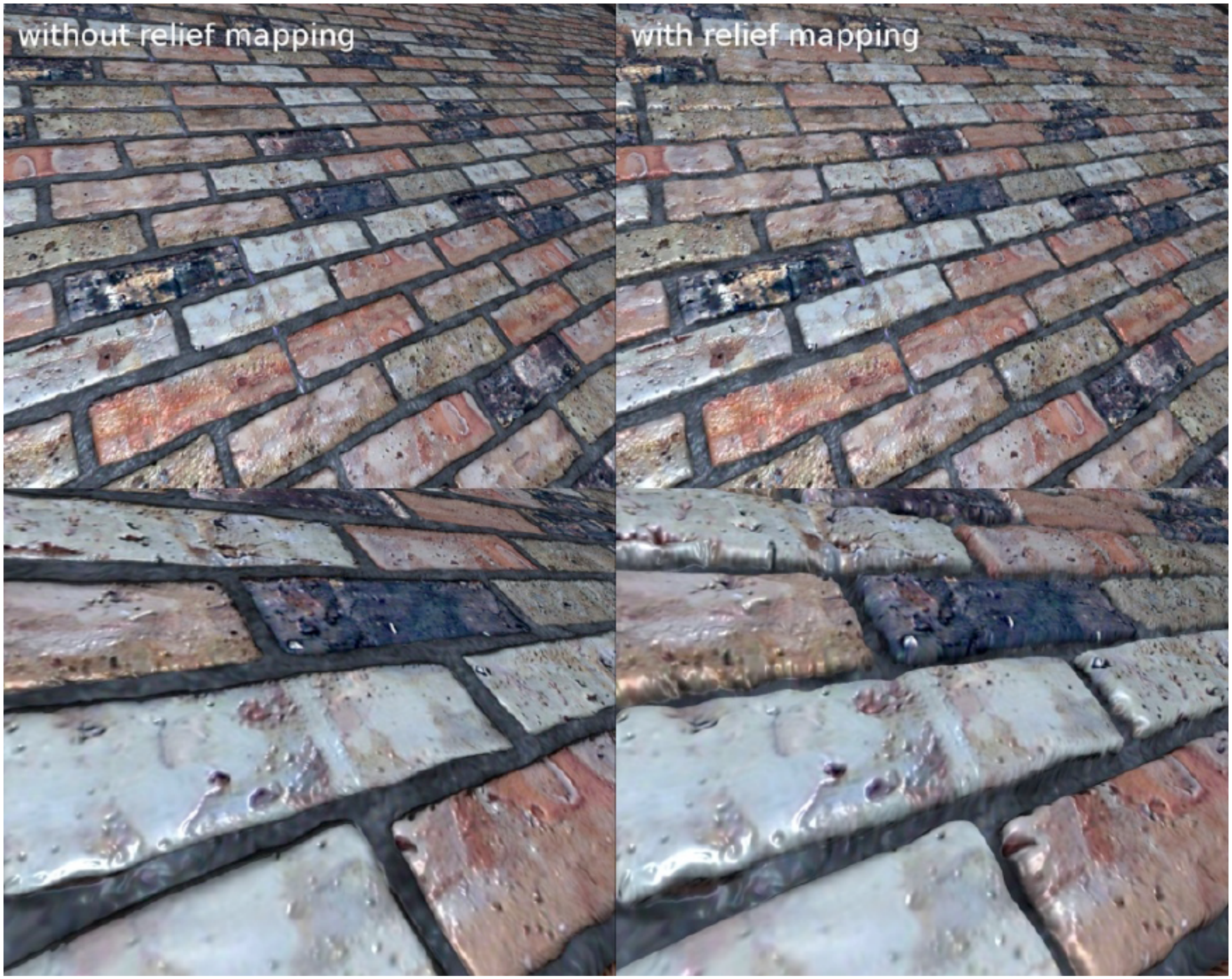


Relief Mapping

- Implement a heightfield raytracer in a shader
- Pretty expensive, but looks amazing



Relief Mapping Example



References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition
- UIUC CS 319, Advanced Computer Graphics Course
- David Luebke, CS 446, U. of Virginia, slides
- Chapter 1-6 of RT Rendering
- Hanspeter Pfister, CS 175 Introduction to Computer Graphics, Harvard Extension School, Fall 2010 slides
- Christian Miller, CS 354, Computer Graphics, U. of Texas, Austin slides, Fall 2011
- Ulf Assarsson, TDA361/DIT220 - Computer graphics 2011, Chalmers Instititute of Tech, Sweden