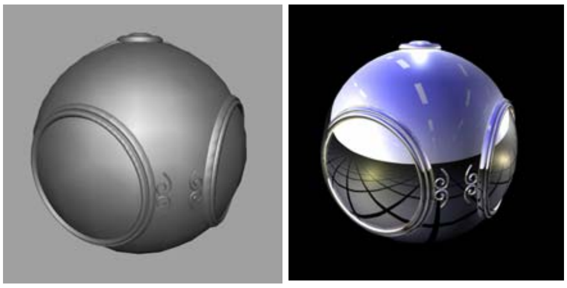
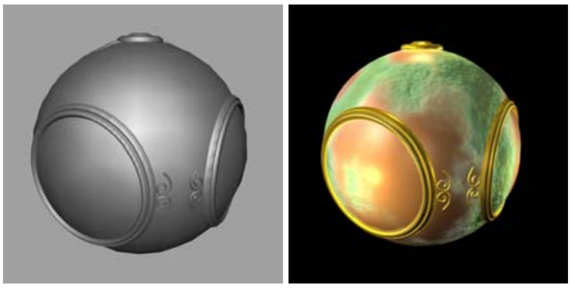
Environment - Bump Mapping

Niels Jørgen Christnesn
IMM . DTU

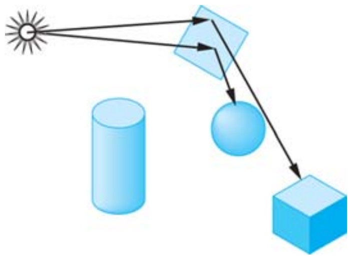
Environment mapping

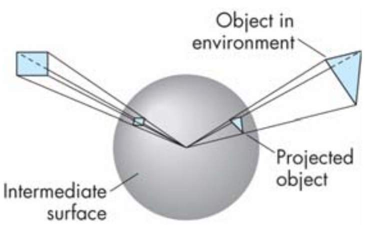


Bump Mapping



Mirrors



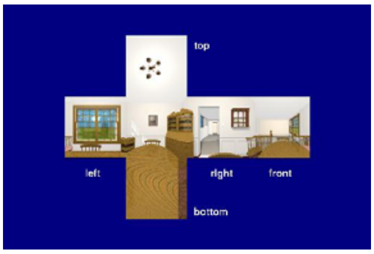


Environment Mapping

- Used to simulate reflective objects
- Can't do it accurately like in raytracing

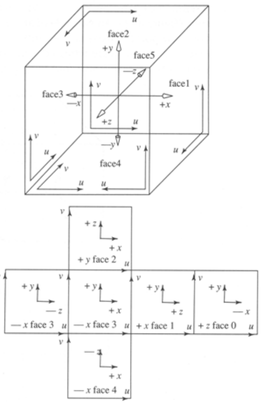
Cube Maps

- A kind of texture in OpenGL
- Stores six 2D textures

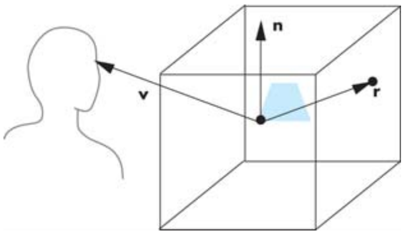


Cube Maps

- Indexed with vec3
- Direction
- Normalization not req.

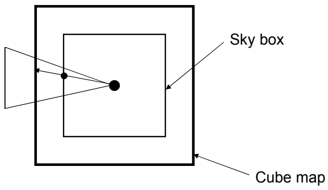


Reflection



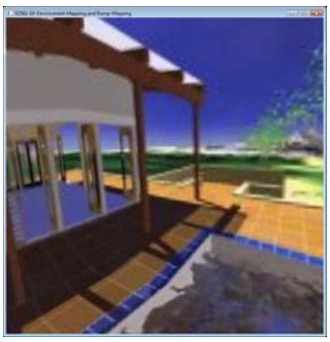
Sky Boxes

- Draw cube map texture on inside of cube
- Camera is in center of cube
- Use object space position as texture coordinate



Sky Boxes

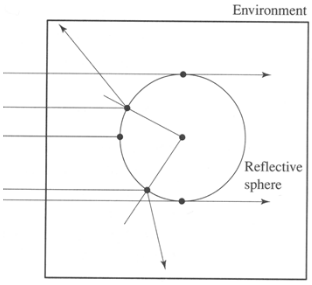
- Often used in games
- Easy to draw

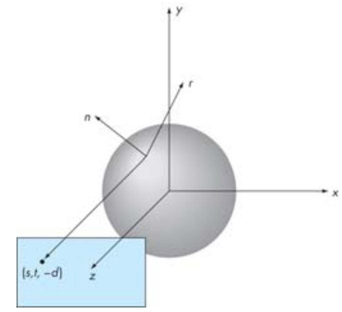


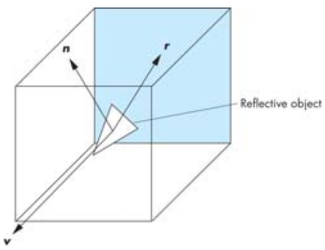
Cube Maps

```
glGenTextures(1, &cubemap);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemap);
for (int i=0; i<6; ++i)
{
    int width, height;
    void* data = load_ppm(cube_fn[i], width, height);
    glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT + i,
          0,                        //level
          GL_RGB8,                  //internal format
          width,                    //width
          height,                   //height
          0,                        //border
          GL_RGB,                   //format or GL_BGR for BMP
          GL_UNSIGNED_BYTE,         //type
          data);                    //pixel data

    delete [] data;
}
```

Environment Mapping







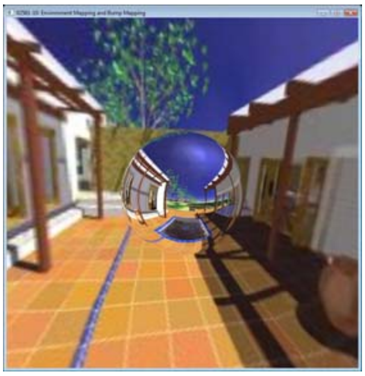
Environment Mapping

- How?
  - Find normal
  - Find incident direction
  - Use GLSL reflect function to find reflected direction
  - Use samplerCube to do lookup in environment map

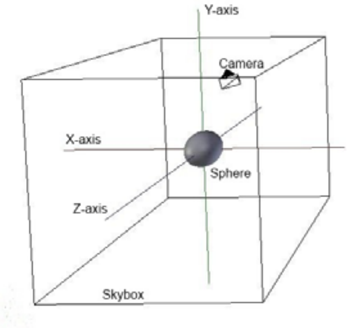
Environment Mapping - mirror

```
uniform samplerCube cubemap;
uniform vec3 cameraPos;
in vec3 vNormal;
in vec3 vPosition;
out vec4 fragColor;

void main(void)
{
    vec3 n = normalize(vNormal);
    vec3 i  = normalize(vPosition - cameraPos);
    vec3 rv = normalize(reflect(i, n));
    fragColor = texture(cubemap, rv);
}
```

Environment Mapping

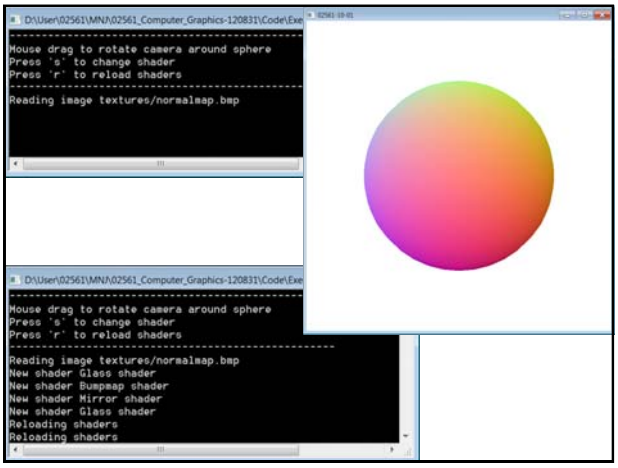


Camera setup

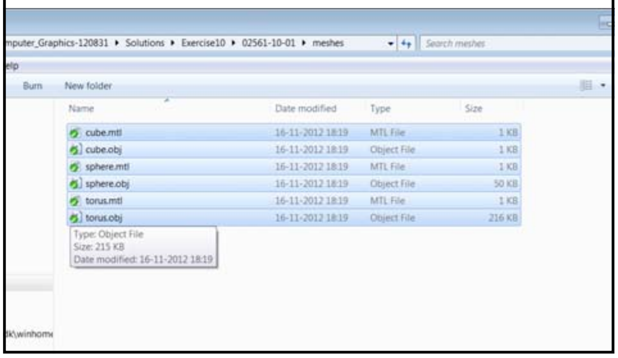


Spherical → Cartesian

```
// spherical coordinates of camera position (angles in radian)
float sphericalPolarAngle = 0;
float sphericalElevationAngle = 0;
float sphericalRadius = 3;

vec3 sphericalToCartesian(float polarAngle, float elevationAngle, float radius);
```



meshes/…



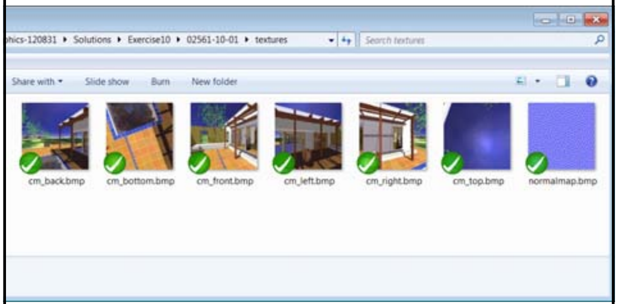
\meshes\sphere.obj

```
# Blender v2.63 (sub 0) OBJ File: ''
# www.blender.org
mtllib sphere.mtl
o pSphere1_pSphereShape1
v 0.148778 -0.048341 0.987688
v 0.126558 -0.091950 0.987688
v 0.091950 -0.126558 0.987688
v 0.048341 -0.148778 0.987688
v 0.000000 -0.156435 0.987688
v -0.048341 -0.148778 0.987688
v -0.091950 -0.126558 0.987688
v -0.126558 -0.091950 0.987688
v -0.148778 -0.048341 0.987688
v -0.156435 0.000000 0.987688
v -0.148778 0.048341 0.987688
v -0.126558 0.091950 0.987688
```

Sphere.mtl

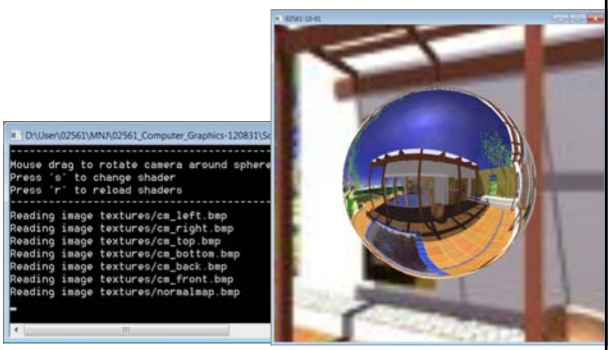
```
# Blender MTL File: 'None'
# Material Count: 1
newmtl lambert1
Ns 96.078431
Ka 0.000000 0.000000 0.000000
Kd 0.400000 0.400000 0.400000
Ks 0.500000 0.500000 0.500000
Ni 1.000000
d 1.000000
illum 2
```

\textures



fragColor = texture(cubemap,normalize(vPosition))

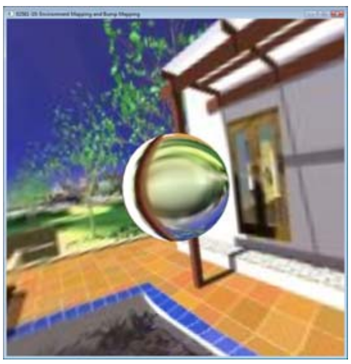
Mirror - reflection



reflect(vec3 incident, vec3 normal)

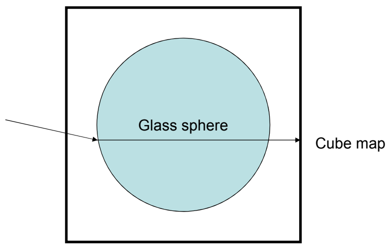
```
#version 150

uniform samplerCube cubemap;
uniform vec3 cameraPos;
in vec3 vNormal;
in vec3 vPosition;
out vec4 fragColor;

void main(void)
{
vec3 n = normalize(vNormal);
vec3 i = normalize(vPosition-cameraPos);
vec3 rv = normalize(reflect(i, n));
fragColor = texture(cubemap, rv);
}
```

Simulating Glass

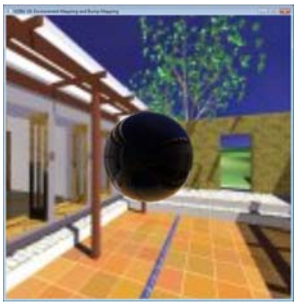


Glas – reflection, refraction

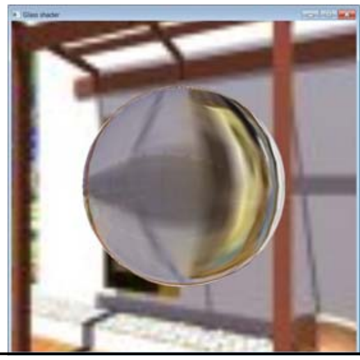
- Refraction can also be simulated
- Use GLSL refract function



Simulating Glass



Glas – mirror, refraction

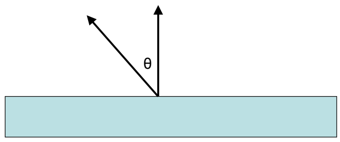


refract(vec3 incident, vec3 normal, float eta)

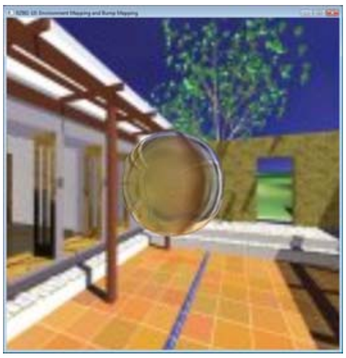
```
uniform samplerCube cubemap;
uniform vec3 cameraPos;
in vec3 vNormal;
in vec3 vPosition; out vec4 fragColor;

void main (void) {
float air = 1.0;
float glass = 1.62;
vec3 n = normalize(vNormal);
vec3 i = normalize(vPosition-cameraPos);
vec3 rv = reflect(i, n);
vec3 refractionDir = refract(i,n, air/glass);
vec4 reflection = texture(cubemap, rv);
vec4 refraction = texture(cubemap, refractionDir);

float R0 = pow(abs((air-glass)/(air+glass)),2.0);
float R = R0 + (1-R0)*pow(1.-max(dot(n, -i),0.), 5);

fragColor = R*reflection + refraction*(1-R);
```

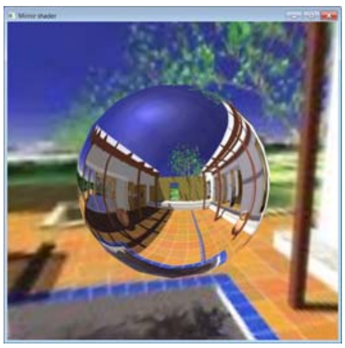
Simulating Glass

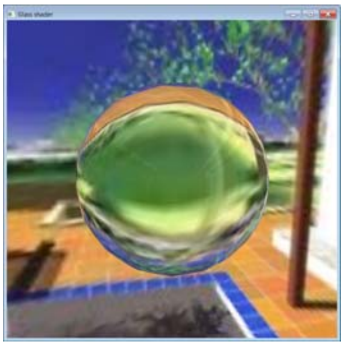
- Combine reflection + refraction
- Fresnel formulae give relative amounts
- Schlick approximation
  - $R = R_0 + (1-R_0)(1-\cos\theta)^5$



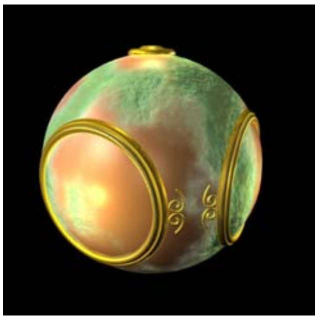
Simulating Glass

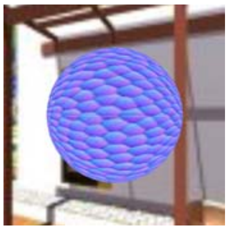


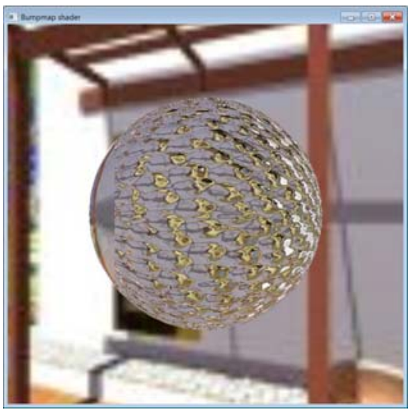




Bump Mapping

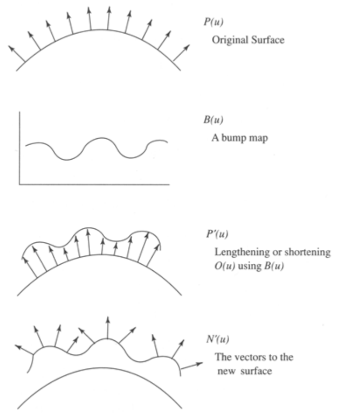




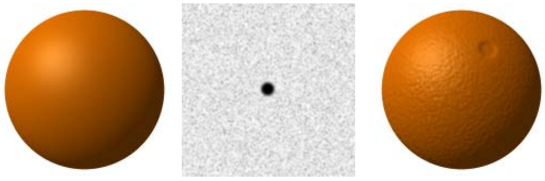


Bump Mapping

- Inexpensive way to add details
- Does not change geometry, only shading calculations
- Basic idea: Replace normal with slightly perturbed normal
- Displacement mapping

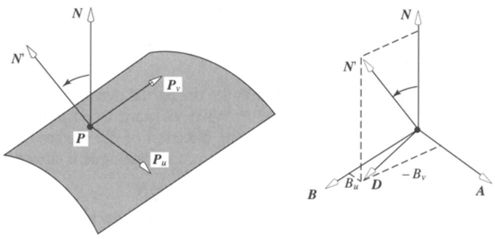


Bump Mapping



(wikipedia.org)

Bump Mapping



Surface normal
at point P
$N = P_v \times P_u$

D is given by
$D = B_u A - B_v B$

Tangents

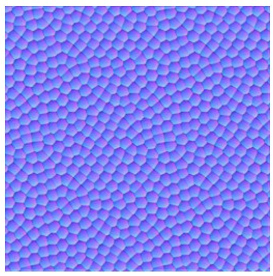
- We will restrict ourselves to spheres
  - Easy tangents!
- Compute (θ, φ)
  - θ = acos(z / radius)
  - φ = atan(y, x)
- Parametric surface
  - x = (sinθ cosφ, sinθ sinφ, cosθ)
  - T = dx/dθ = (cosθ cosφ, cosθ sinφ, -sinθ)
  - B = dx/dφ = (-sinθ sinφ, sinθ cosφ, 0)

Normal Mapping

- A texture that stores normals encoded as colors
- Two kinds: tangent space + object space
  - We use tangent space
- Faster and easier than than traditional bumpmaps
  - Precomputed derivatives
  - Stores normal directly

Normal Mapping, cont'd

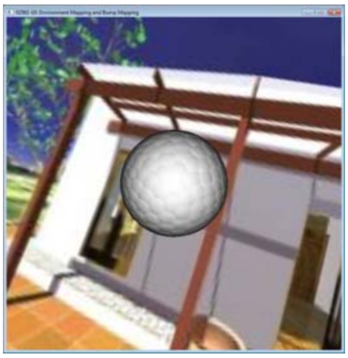
- Normal maps look bluish
- Color corresponds to directions
- V = (C - 0.5) * 2.



Normal Mapping

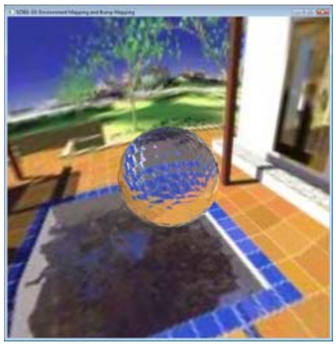
- How?
  - – Lookup in normal map according to (θ, φ)
  - – Transform color to direction V
  - – Compute tangents and normal
  - – Store as columns in matrix [T B N]
  - – Compute normal as N' = [T B N] × V
  - – Use new normal to compute reflection/refraction

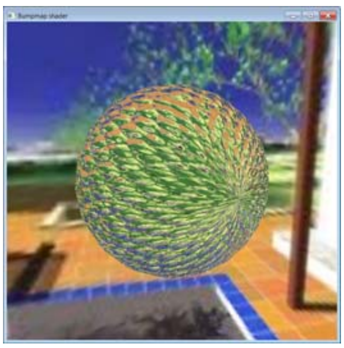
Normal Mapping



```
#version 150
uniform samplerCube cubemap;
uniform sampler2D textureBump;
uniform vec3 cameraPos;
in vec3 vNormal;
in vec3 vPosition; // new
out vec4 fragColor;

vec3 bumpMap(vec3 normal, vec3 position){
vec3 n = normalize(normal);
float radius = length(vec3(position));
float PI = 3.14159265;
vec2 tangent2 = vec2(acos(position.z/radius), atan(position.y, position.x));
vec2 tangentUV = vec2(tangent2.x / (2*PI), tangent2.y / PI);

vec3 T = vec3(cos(tangent2.x)*cos(tangent2.y), cos(tangent2.x)*sin(tangent2.y),-sin(tangent2.x));
vec3 B = vec3(-sin(tangent2.x)*sin(tangent2.y), sin(tangent2.x)*cos(tangent2.y), 0.0);
mat3 tbn = mat3(T,B,n);

vec3 nn = texture(textureBump,tangentUV).xyz;
vec3 V = (nn - vec3(0.5, 0.5, 0.5))*2.0;
return normalize(tbn*V);
}
```

Normal Mapping





Normal/Bump mapping

```
void main(void) {
……
vec3 n = bumpMap(vNormal, vPosition);
vec3 i = normalize(vPosition-cameraPos);
vec3 rv = reflect(i, n);
vec4 reflection = texture(cubemap, rv);
float air = 1.0;
float glass = 1.62;

vec3 refractv = normalize(refract(i,n,air/glass));
vec4 refraction = texture(cubemap, refractv);

float R0 = pow(abs((air-glass)/(air+glass)),2.0);
float R = R0 + (1-R0)*pow(1.-max(dot(n, -i),0.), 5);
fragColor = R*reflection + refraction * (1-R);
}
```