CS 432 – Interactive Computer Graphics

Lecture 8 – Part 1

Environment Mapping, Reflections and Refractions

Topics

- Environment Mapping
- Reflections/Refractions/Mirrors

Reading

- Angel
  - Section 7.8 (p388)
  - Section 7.9 (p393)
  - Section 7.10 (p396)
- Red Book
  - Chapter 6
    - Environment Mapping p313
    - Rendering to Texture Maps p351

Environment Mapping

Environment Mapping

- Environment mapping is very similar to building a skybox
- Put texture on a *highly reflective* object where the texture is an image of the environment in which the object is immersed
- Realized as two-step process
  - Project the environment (excluding the object) onto an intermediate object
    - Usually a cube or sphere
  - Place object back, and map texture from intermediate surface back to object

Environment Cube Map

Approach:

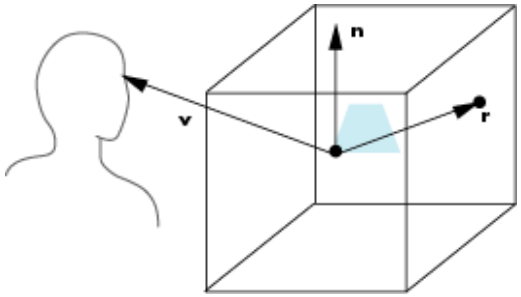
1. Do 6 renderings of the scene with the reflective object removed
   - Change the frame buffer we're using for rendering (use one other than the one used to display to the screen)
   - Render the scene for the desired point-of-view (POV)
     - Use a 90 degree angle of view
   - Move the content of the frame buffer into a texture
2. Use these textures for cube mapping

Environment Cube Map

3. Use the normals of the reflective object and the current eye location to find the location in the cube map

$$R = 2(N \cdot V)N - V$$

- Use largest magnitude component of R to determine face of cube
- Other 2 components give texture
- Can use GLSL's `reflect(vec3 V, vec3 N)` function to compute R for us
- As always we want to make sure everything's in the same coordinate system (probably the camera coordinate system for this)



Render To Texture

- The first thing we need to do is put the camera (temporarily) at the center of the object and take 90 degree “shots” in all 6 directions

Move camera to center of object
Open up projection to 90 degrees

In this case my reflective object is at (0,0,0) but otherwise we must translate to the center of it

```
vec4 eye =  vec4(modelmatrix[0].w, modelmatrix[1].w, modelmatrix[2].w, 1);
mat4 proj_matrix = Perspective(90.0f, 1, 0.1f, 200);
Camera tempCam;
tempCam.setProjection(proj_matrix);
glViewport(0, 0, 256, 256);
```

Project onto 256x256 window

OpenGL Render to Texture

- We need
  1. A framebuffer to capture the image in
     - The framebuffer inverts the y-direction we have to deal with this
  2. A depth buffer to help with the computations (hidden surface removal)
  3. A texture to put the content of the framebuffer on

- Generate these just once in the `init` so we don't keep getting more and more IDs
  - `glGenFramebuffers(1,&framebuffer);`
  - `glGenRenderbuffers(1,&depthrenderbuffer);`
  - `glGenTextures(1,&texture);`

OpenGL Render to Texture

- We should also set up our parameters for our texture and allocate space

```
glBindTexture(GL_TEXTURE_CUBE_MAP, texture);

glTexParameterf(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);

glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
```

OpenGL Render to Texture

- When it's time to render to texture we must first:
  - Make our desired target frame and render buffers active
    - `glBindFramebuffer(GL_FRAMEBUFFER,framebuffer);`
    - `glBindRenderbuffer(GL_RENDERBUFFER,depthrenderbuffer);`
  - Specify the size of the depth buffer
    - `glRenderbufferStorage(GL_RENDERBUFFER,GL_DEPTH_COMPONENT,256,256);`

OpenGL Render to Texture

- Finally we can render each face

Make our cubemap texture active

Choose the at/up directions for current rendering. For the right/left/front/back we choose up=-y since the framebuffer inverts y

```
for(int i=0; i<6; i++){
    glBindTexture(GL_TEXTURE_CUBE_MAP,texture);

    // choose the at, up and desired face of the cubemap for this face
    switch (i)
    {
        case 0:  //right
            at = vec4(1,0,0,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_X, texture, 0);
            break;

        case 1:  //left
            at=vec4(-1,0,0,1);
            up=vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, texture, 0);
            break;

        case 2:  //top
            at = vec4(0,1,0,1);
            up = vec4(0,0,1,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, texture, 0);
            break;

        case 3: //bottom
            at = vec4(0,-1,0,1);
            up = vec4(0,0,-1,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, texture, 0);
            break;

        case 4: //front
            at = vec4(0,0,1,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, texture, 0);
            break;

        case 5:  //back
            at=vec4(0,0,-1,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, texture, 0);
            break;

        default:
            break;
    };
```

```
for(int i=0; i<6; i++){
    glBindTexture(GL_TEXTURE_CUBE_MAP,texture);

    // choose the at, up and desired face of the cubemap for this face
    switch (i)
    {
        case 0:  //right
            at = vec4(1,0,0,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_X, texture, 0);
            break;

        case 1:  //left
            at=vec4(-1,0,0,1);
            up=vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, texture, 0);
            break;

        case 2:  //top
            at = vec4(0,1,0,1);
            up = vec4(0,0,1,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, texture, 0);
            break;

        case 3: //bottom
            at = vec4(0,-1,0,1);
            up = vec4(0,0,-1,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, texture, 0);
            break;

        case 4: //front
            at = vec4(0,0,1,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, texture, 0);
            break;

        case 5:  //back
            at=vec4(0,0,-1,1);
            up = vec4(0,-1,0,1);
            glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, texture, 0);
            break;

        default:
            break;
    };
```

OpenGL Render to Texture

- Finally we can render each face (continued)

Test to make sure we were able to bind to the new framebuffer ok

```
    if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
        std::cout << "ERROR\n";

    at += eye;
    at.w = 1;
    vec3 right = cross(vec3(at.x, at.y, at.z) - vec3(eye.x, eye.y, eye.z), vec3(up.x, up.y, up.z));
    tempCam.positionCamera(eye, normalize(eye - at), up, normalize(vec4(right.x, right.y, right.z, 0)));

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    for (unsigned int n = 0; n < objects.size(); n++) {
        if(objects[n]!=this)
          objects[n]->draw(tempCam, lights);
    }
}
```

Compute the new camera matrix using the at/up for this view

Draw the scene without the reflecting object
(drawing will use the new projection and camera matrices)

End of *for* loop over the faces

Restore from the Chaos

- After rendering all 6 faces into the framebuffer and the faces of the cube map we must restore to our prior rendering state


```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glViewport(0, 0, ww, wh);
```

Using the Environment Map

- Now when rendering our reflective object we just need to do our standard texture/cubemap stuff:
  - Choose a texture unit
  - Ensure that the desired texture type is active in that unit
  - Bind a texture to that texture type
  - Tell the shader which texture unit to use for its `samplerCube` variable
  - Draw!

```
glActiveTexture(GL_TEXTURE0);
glEnable(GL_TEXTURE_CUBE_MAP);
glBindTexture(GL_TEXTURE_CUBE_MAP,texture);
GLuint cubemap_loc =glGetUniformLocation(program, "cubeMap");
glUniform1i(cubemap_loc, 0);


glDrawArrays(GL_TRIANGLES,0,numSphereVertices);
```

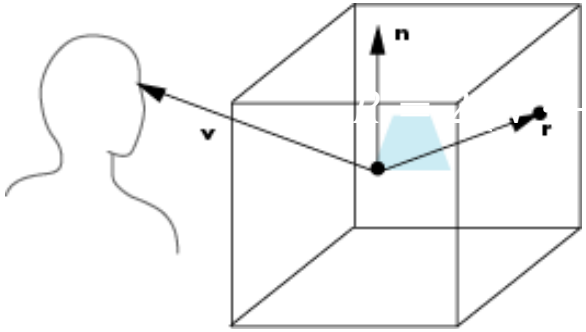
Drawing the Scene

- Now in our main display callback we want to:
  - Make the environment map
  - Draw skybox (if applicable)
  - Draw all the objects

```
void display( void )
{
      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
      sphere->makeEnvironmentMap(lights, objects);
      glViewport(0, 0, ww, wh);

      skybox->draw(skyboxCamera, lights);
      for (unsigned int i = 0; i < objects.size(); i++) {
            if (objects[i] != skybox)
                  objects[i]->draw(camera, lights);
      }

      glutSwapBuffers();
}
```

Reflective Map Vertex Shader

- Instead of hard coding the texture coordinate for each vertex, we're going to compute them using *reflection*



- To compute V we need to know the eye and vertex locations in the same coordinate system
  - And the normal N must also be adjusted to that coordinate system

Reflective Map Vertex Shader

- We need our normal (N) and to (V) vector to be in the same coordinate system in order to compute R
- And our "final R" should be relative to the center of the cubemap (which is just a 2x2x2 cube at the origin)
- So let's do everything in world coordinate
  - Since the cubemap lives at the center of the world (or really it just has and identity model matrix)

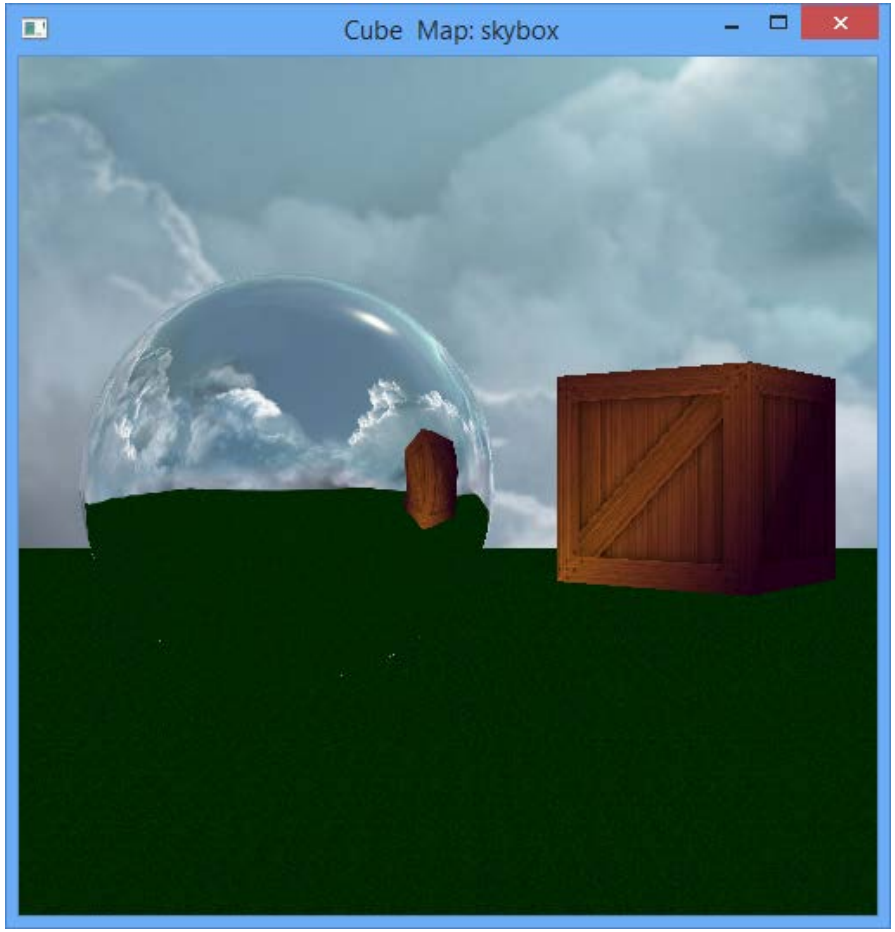
Reflective Map Vertex Shader

- So let's pass into our vertex shader:
  - Normal (in model space)
  - Position (in model space)
  - Model matrix (to go from model→world)
  - Camera Position (in world space)
- Let us define the *incident* vector to be opposite the *to* vector. Then we can compute:
  - `I=model_matrix*vPosition-camera_loc`
  - `NWorld=model_matrix*vec4(vNormal,0);`
- And use the built-in GLSL function to get the reflector
  - `R = reflect(normalize(I.xyz),normalize(Nworld.xyz));`

```
vec4 I=model_matrix*vPosition - camera_loc;
vec4 NWorld = model_matrix*vec4(vNormal,0);

R = reflect(normalize(I.xyz),normalize(NWorld.xyz));
```

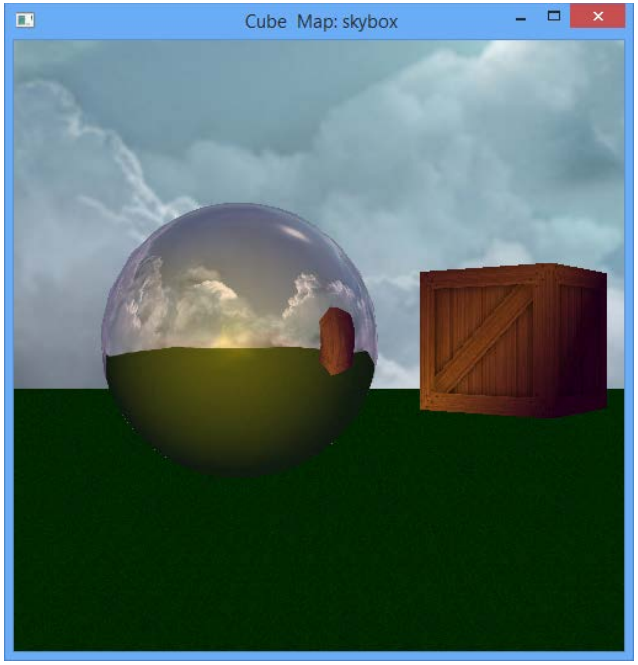
Reflection Map Fragment Shader

```
in vec3 R;
uniform samplerCube cubeMap;


void main(){

        gl_FragColor = texture(cubeMap,R);

}
```



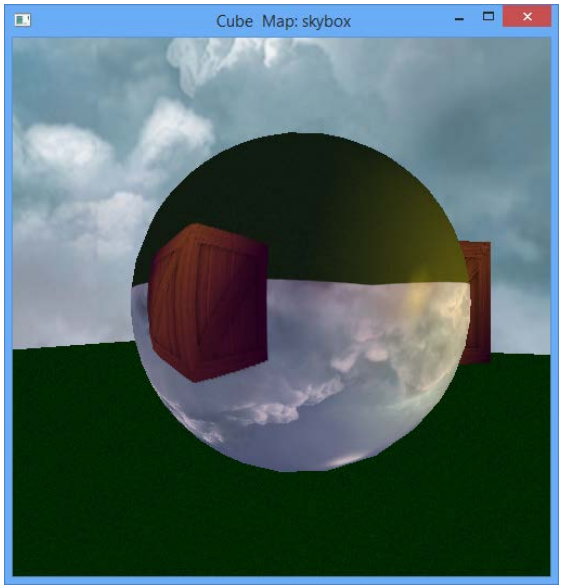
Blending Textures and Shades

- We can also blend/mix the shading color with the texture
- `gl_FragColor = mix(color,texture(cubeMap,R),0.70);`
  - Or come up with your own way to combine them!



Refraction

- We can also use the GLSL *refract* function to model things like water, glass, etc..
- It works just like the reflect function but with an additional refraction parameter
  - `vec3 refract(vec3 Inc, vec3 N, float eta)`
  - With `eta=0.1`



Mirrors

- Hopefully you can imagine how to easily adapt this to things like mirrors:
  - Do it the same way as we did with the sphere
  - But obviously only one surface is mapped to
- A more efficient way to do this would be:
  - Take a “photo” of the environment using the location and orientation of the center of the mirror
  - Only need 1 photo, but take it with a super wide lens (180 fov?)
  - Use this photo for a texture map (NOT cube map)