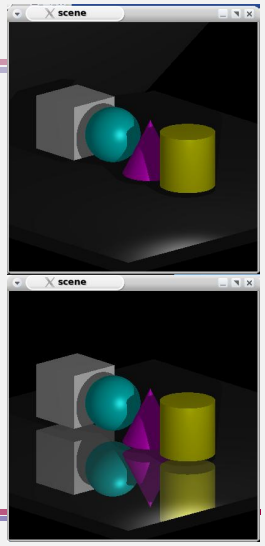
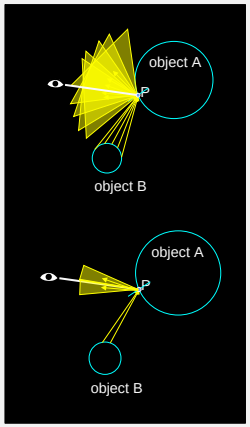
Handling Reflections

- so far we have no way to render reflective surfaces



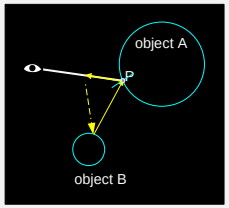
Handling Reflections

- matte surfaces reflect light equally in all directions
  - diffuse term models direct illumination
  - ambient term models indirect illumination
    - the color we see for a point is a blend of light rays coming from many other points in the scene
- shiny surfaces are highly directional in how they reflect light
  - specular term models direct illumination
  - nothing models indirect illumination



Environment Mapping

- reflections can be computed through raytracing
  - this is expensive

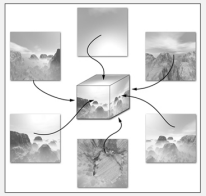


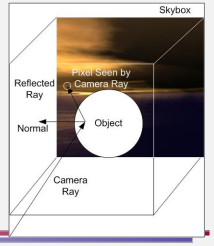
- reflections can be faked by applying the right texture to the surface
  - this is environment mapping

 http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Implementing Environment Mapping

- concepts
  - a *skybox* is a large cube surrounding the entire scene (including the camera)
  - a *cubemap* texture applied to the skybox represents the rest of the world outside of what is being modeled
- rendering
  - render the skybox with the cubemap texture
  - render the object(s) using the skybox as a map shape
    - use the reflection vector to map object point → map shape
    - use the texture color alone for perfectly reflective objects or combine with lighting equation color for shiny but not mirrored surfaces
      - "combine" = add or mix (rather than replace or modulate) – reflected light is the sum of contributions from each source





https://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter07.html
https://commons.wikimedia.org/wiki/File:Cube_mapped_reflection_example.jpg

Working With Cubemap Textures

- WebGL has built-in support for cubemaps
- overall process is similar to working with 2D image textures
  - in javascript –
    - set up texture object
    - pass texture to use to the fragment shader
  - in fragment shader –
    - receive or compute texture coordinates
    - get the texture's color at that position
    - use texture color to compute pixel color

Working With Cubemap Textures

create a texture object

set the current texture object, being used as a cubemap rather than a 2D texture

don't flip for cubemap

CLAMP_TO_EDGE helps avoid seams on edges

load the image – done 6 times, one for each face of the cube

generate mipmaps for the whole cubemap once image loading is complete (onload is a callback which runs each time load completes)

textureObject is the global JS variable, set once the initialization of the whole texture object is complete

```
function loadCubemapTexture() {
    var tex = gl.createTexture();
    var imageCt = 0;
    load( "cubemap-textures/park/negx.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_X );
    load( "cubemap-textures/park/posx.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_X );
    load( "cubemap-textures/park/negy.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_Y );
    load( "cubemap-textures/park/posy.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_Y );
    load( "cubemap-textures/park/negz.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_Z );
    load( "cubemap-textures/park/posz.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_Z );
    function load(url, target) {
        var img = new Image();
        img.onload = function() {
            gl.bindTexture(gl.TEXTURE_CUBE_MAP, tex);
            gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,0);
            gl.texParameteri(gl.TEXTURE_CUBE_MAP,
                gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
            gl.texParameteri(gl.TEXTURE_CUBE_MAP,
                gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
            gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, img);
            imageCt++;
            if (imageCt == 6) {  // all 6 images have been loaded
                gl.generateMipmap( gl.TEXTURE_CUBE_MAP );

                textureObject = tex;
                draw();
            }
        }
        img.onerror = function() {
            document.getElementById("headline").innerHTML =
                                        "SORRY, COULDN'T LOAD TEXTURES";
        }
        img.src = url;
    }
}
```



```
function loadCubemapTexture() {
    var tex = gl.createTexture();
    var imageCt = 0;
    load( "cubemap-textures/park/negx.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_X );
    load( "cubemap-textures/park/posx.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_X );
    load( "cubemap-textures/park/negy.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_Y );
    load( "cubemap-textures/park/posy.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_Y );
    load( "cubemap-textures/park/negz.jpg", gl.TEXTURE_CUBE_MAP_NEGATIVE_Z );
    load( "cubemap-textures/park/posz.jpg", gl.TEXTURE_CUBE_MAP_POSITIVE_Z );
    function load(url, target) {
        var img = new Image();
        img.onload = function() {
            gl.bindTexture(gl.TEXTURE_CUBE_MAP, tex);
            gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,0);
            gl.texParameteri(gl.TEXTURE_CUBE_MAP,
                gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
            gl.texParameteri(gl.TEXTURE_CUBE_MAP,
                gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
            gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, img);
            imageCt++;
            if (imageCt == 6) {  // all 6 images have been loaded
                gl.generateMipmap( gl.TEXTURE_CUBE_MAP );

                textureObject = tex;
                draw();
            }
        }
        img.onerror = function() {
            document.getElementById("headline").innerHTML =
                                        "SORRY, COULDN'T LOAD TEXTURES";
        }
        img.src = url;
    }
}
```

```
// load texture image asynchronously - calls draw() when loading is complete
// if flip is true, flip the image
// also generates mipmaps - texture image must be a power-of-two size
function loadTexture( url, textureObject, flip ) {
    var img = new Image();  //  A DOM image element to represent the image.
    img.onload = function() {
        // This function will be called after the image loads successfully.
        // We have to bind the texture object to the TEXTURE_2D target before
        // loading the image into the texture object.
        gl.bindTexture(gl.TEXTURE_2D, textureObject);
        if ( flip ) {
            gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,1);
        } else {
            gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL,0);
        }
        gl.texImage2D(gl.TEXTURE_2D,0,gl.RGBA,gl.RGBA,gl.UNSIGNED_BYTE,img);
        gl.generateMipmap(gl.TEXTURE_2D);
        draw();  // Draw the canvas, with the texture.
    }
    img.onerror = function(e,f) {
        // This function will be called if an error occurs while loading.
        document.getElementById("headline").innerHTML =
                        "<p>Sorry, texture image could not be loaded.</p>";
        draw();  // Draw without the texture; triangle will be black.
    }
    img.src = url;  // Start loading of the image.
                    // This must be done after setting onload and onerror.
}
```


Working With Cubemap Textures

- pass texture information to shaders
  - shader has parameter of type `samplerCube`

```
uniform samplerCube cubemap;
```

  - associate texture object with a particular texture unit
  - set shader parameter to reference the texture unit

```
gl.activeTexture(gl.TEXTURE0);   // working with texture unit 0
gl.bindTexture(gl.TEXTURE_CUBE_MAP,textureObj);  // associate texture obj
                                                 // with active texture unit
gl.uniform1i( cubemap_loc ,0);  // set sampler var to texture unit 0
```

Working With Cubemap Textures

- fragment shader
  - sample texture to get color
  - may be passed or compute texture coordinates
  - may use texture color in lighting equation or in some other way

```
precision mediump float;
varying vec3 v_objCoords;
uniform samplerCube cubemap;

void main () {
  gl_FragColor =
    textureCube(cubemap,v_objCoords);
}
```

cubemap textures are sampled using 3D vector from the origin to the point

Working With Cubemap Textures

- notes
  - two texture objects can be bound to the same texture unit at the same time, as long as one is a 2D texture and one is a cubemap texture
  - it is common to load cubemap images from a file, but the data can be generated by other means, just like for 2D textures
  - cubemap images must all be the same size, square, and a power-of-two size
  - recommended to set texture wrap mode to `CLAMP_TO_EDGE` to avoid chance of visible seams between cube faces