Render to Texture

The idea –

- create a texture object, but without specifying image data
- create a framebuffer to render to
- attach the texture to the framebuffer as a color buffer
- create an additional renderbuffer for use as a depth buffer
- attach that renderbuffer to the framebuffer as a depth buffer
- draw to the framebuffer
- draw the scene using the generated texture

done in `initGL`

if depth buffer is needed

done in `draw`

Render to Texture

- draw to the framebuffer
  - rendering a 2D scene

```
gl.bindFramebuffer(gl.FRAMEBUFFER,framebuffer);
gl.useProgram(prog_texture); // shader program for the texture

gl.clearColor(1,1,1,1);
gl.clear(gl.COLOR_BUFFER_BIT);

gl.disable(gl.DEPTH_TEST); // framebuffer doesn't even have a depth buffer!
gl.viewport(0,0,512,512); // Viewport is not set automatically!

  .
  .  // draw the texture image, which changes in each frame
  .
```

  - for rendering a 3D scene, also clear `gl.DEPTH_BUFFER_BIT` and do not disable `gl.DEPTH_TEST`

make the texture framebuffer the current one (`framebuffer` is the javascript variable for this framebuffer)

if using different shader programs for rendering texture vs rest of scene

set background color as desired

depth buffer not used for 2D drawing

viewport is not set automatically for other framebuffers (size must match size specified in `gl.texImage2D`)

Render to Texture

- draw the scene using the generated texture

```
gl.bindFramebuffer(gl.FRAMEBUFFER,null); // Draw to default framebuffer.
gl.useProgram(prog);  // shader program for the on-screen image
gl.clearColor(0,0,0,1);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.enable(gl.DEPTH_TEST);
gl.viewport(0,0,canvas.width,canvas.height);  // Reset the viewport!

.
.   // draw the scene
.     (use the texture object as usual)
```

set current framebuffer to the default framebuffer (for drawing to the display)

if using different shader programs for rendering texture vs rest of scene

set background color as desired

if depth test was previously disabled (for rendering 2D scene)

must restore the viewport manually

canvas is the javascript variable referring to the canvas on the web page

Dynamic Cubemap Textures

Procedure –

- create a cubemap texture for the environment map
  - create a cubemap texture object, but without specifying image data
  - create a framebuffer to render to
  - attach the texture to the framebuffer as a color buffer
  - create an additional renderbuffer for use as a depth buffer
  - attach that renderbuffer to the framebuffer as a depth buffer
- draw to the framebuffer 6 times, once for each face of the cubemap
  - each time, the full scene (skybox + objects) is drawn – from a different viewing angle
- draw scene (skybox + objects) using the generated cubemap as the skybox texture

Dynamic Cubemap Textures

- create a cubemap texture object, but without specifying image data

```
cubemapTargets = [
         // store texture targets in an array for convenience
   gl.TEXTURE_CUBE_MAP_POSITIVE_X, gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
   gl.TEXTURE_CUBE_MAP_POSITIVE_Y, gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
   gl.TEXTURE_CUBE_MAP_POSITIVE_Z, gl.TEXTURE_CUBE_MAP_NEGATIVE_Z
];

dynamicCubemap = gl.createTexture(); // Create the texture object.
gl.bindTexture(gl.TEXTURE_CUBE_MAP, dynamicCubemap);  // bind it as a cubemap
for (i = 0; i < 6; i++) {
   gl.texImage2D(cubemapTargets[i], 0, gl.RGBA, 512, 512,
                                    0, gl.RGBA, gl.UNSIGNED_BYTE, null);
}
```


Dynamic Cubemap Textures

- create a framebuffer to render to
- attach the texture to the framebuffer as a color buffer
  - repeat the following for each target

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                        gl.TEXTURE_CUBE_MAP_NEGATIVE_Z, dynamicCubemap, 0);
```

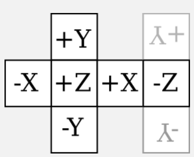
- create an additional renderbuffer for use as a depth buffer
  - can use the same renderbuffer to render all 6 images
- attach that renderbuffer to the framebuffer as a depth buffer

Dynamic Cubemap Textures

- draw to the framebuffer
  - idea
    - place the camera at the center of the reflective object
    - point the camera towards each of the six sides of the skybox
  - “camera” includes both projection and viewing transforms
    - for a cubemap, need a square view window and a 90-degree field of view

```
mat4 .perspective( projection, Math.PI/2, 1, 1, 100 );
```

    - viewing transforms point towards each of the cube faces



cubemap textures are from the outside of the cube, but the camera sees the inside of the cube
– need to flip horizontally
– may also need to flip vertically to deal with WebGL's convention for the image data starting with the bottom row


```
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
gl.viewport(0,0,512,512);  //match size of the texture images
mat4.perspective(projection, Math.PI/2, 1, 1, 100);  // Set projection to give 90-degree field of view.

modelview = mat4.create();

mat4.identity(modelview);
mat4.scale(modelview,modelview,[-1,-1,1]);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_NEGATIVE_Z, dynamicCubemap, 0);
renderSkyboxAndCubes();

mat4.identity(modelview);
mat4.scale(modelview,modelview,[-1,-1,1]);
mat4.rotateY(modelview,modelview,Math.PI/2);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_POSITIVE_X, dynamicCubemap, 0);
renderSkyboxAndCubes();

mat4.identity(modelview);
mat4.scale(modelview,modelview,[-1,-1,1]);
mat4.rotateY(modelview,modelview,Math.PI);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_POSITIVE_Z, dynamicCubemap, 0);
renderSkyboxAndCubes();

mat4.identity(modelview);
mat4.scale(modelview,modelview,[-1,-1,1]);
mat4.rotateY(modelview,modelview,-Math.PI/2);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_NEGATIVE_X, dynamicCubemap, 0);
renderSkyboxAndCubes();

mat4.identity(modelview);
mat4.rotateX(modelview,modelview,Math.PI/2);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_NEGATIVE_Y, dynamicCubemap, 0);
renderSkyboxAndCubes();

mat4.identity(modelview);
mat4.rotateX(modelview,modelview,-Math.PI/2);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_CUBE_MAP_POSITIVE_Y, dynamicCubemap, 0);
renderSkyboxAndCubes();
```

horizontal and vertical flip

draws skybox and non-reflective objects

flip is already incorporated in camera rotation

```
gl.bindTexture(gl.TEXTURE_CUBE_MAP, dynamicCubemap);
gl.generateMipmap( gl.TEXTURE_CUBE_MAP );
```

Other Aspects of Framebuffers

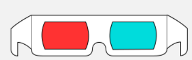
- *blending* refers to how color from fragment shader is combined with the current color in the color buffer
  - default is replace (if at lesser depth)
  - `gl.enable(gl.BLEND)` enables blending
  - `gl.blendFunc` sets how to blend
    - `gl.blendFunc(gl.SRC_ALPHA,gl.ONE_MINUS_SRC_ALPHA)`
      - alpha blending: src*src.a + dest*(1-src.a)
    - `gl.blendFunc(gl.ONE,gl.ZERO)`
      - default: src*1 + dest*0
  - `gl.blendFuncSeparate` allows different blend functions for RGB and alpha components
    - `gl.blendFuncSeparate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ZERO,gl.ONE)`
      - use alpha blending for RGB components but use the alpha already in the color buffer – keeps the canvas itself opaque

Other Aspects of Framebuffers

- control writing to buffers
  - depth buffer – `gl.depthMask(`*`mask`*`)`
    - *`mask`* is boolean – `true` to write
    - note that `gl.enable(gl.DEPTH_TEST)` controls usage of the depth buffer during rendering
  - color buffer – `gl.colorMask(`*`redmask`*`,`*`greenmask`*`,`*`bluemask`*`,`*`alphamask`*`)`
    - *`mask`* values are booleans – `true` to write

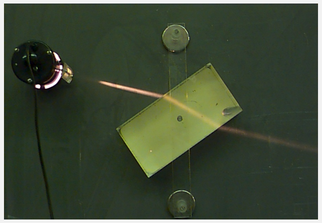
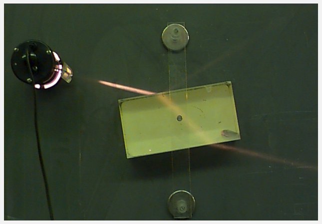
Other Aspects of Framebuffers

- applications
  - rendering translucent objects
    - draw opaque objects with depth mask on
    - draw translucent objects with depth mask off (but use of depth buffer on) and alpha blending on
  - anaglyph stereo
    - draw left and right eye images with red channel for one and green/blue channels for the other
    - clear depth buffer but not color buffer before drawing second image




Refraction

- *refraction* refers to the bending of light at boundary between different materials due to light traveling at different speeds in different materials
  - from faster to slower medium → bends towards normal
  - from slower to faster medium → bends away from normal

http://www.physics.brown.edu/Studies/Demo/optics/demo/6a4210.htm

Refraction

http://www.physics.brown.edu/Studies/Demo/optics/demo/6a4220.htm

Refraction via Environment Mapping

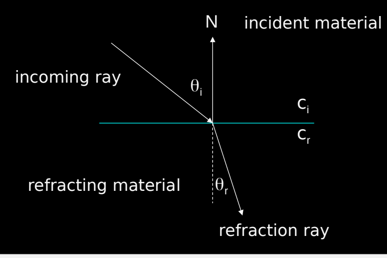
- use a skybox as with reflection
  - use the refraction ray to sample the cubemap rather than the reflection ray

Computing the Refraction Ray

- angle of refraction θ_r depends on
  - index of refraction of each material
    - c_i for the incident material
    - c_r for the refracting material
  - angle of incidence θ_i

$$\sin\theta_r = \frac{c_i}{c_r}\sin\theta_i$$

(Snell's Law)

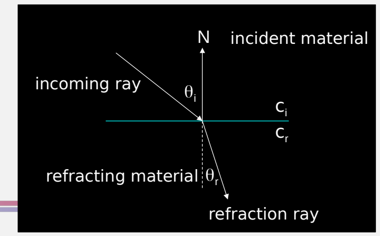


Computing the Refraction Ray

- GLSL has a function `refract` which returns the refraction ray

```
refract(I,N,iorratio)
```

- `I` is the incident vector (normalized)
  - from the camera to the surface point – in EC, this is just the EC surface point (normalized)
- `N` is the outward surface normal (normalized)
- `iorratio` is the ratio c_i/c_r



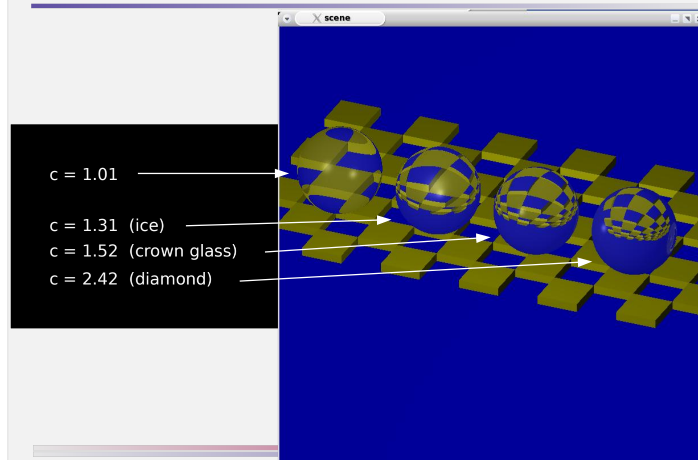
Index of Refraction *n*

common values

vacuum	1.000	crown glass	1.52
air	1.00029	flint glass	1.65
ice	1.31	sapphire	1.77
water	1.33	diamond	2.42

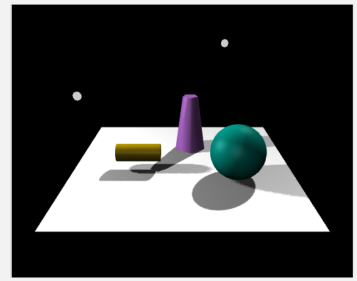
higher value means that light travels more slowly

Effect of the Index of Refraction



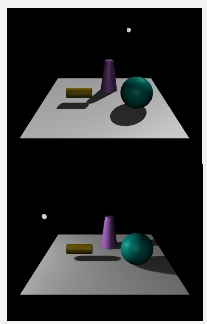
Shadows

- shadow mapping
  - look at the scene from the point of view of the light – the things not visible are in shadow



Shadow Mapping

- method
  - place camera at the light source and render the scene
    - only the depth buffer is needed (*shadow map*)
      - gives the distance from the light to the nearest surface to the light
    - generate a separate shadow map for each light
  - when rendering the scene –
    - transform point into the light's coordinate system
    - only include the contribution from that light if the depth of the transformed point is no greater than depth in the shadow map



- shortcomings
  - does not handle transparent objects
  - assumes only direct illumination from light sources

Recap

- so far we've been studying realtime computer graphics
  - based on the *polygon pipeline* which can be processed very quickly by graphics hardware
- we've used OpenGL – low-level graphics library
  - WebGL / OpenGL 2 – programmable pipeline
    - user can specify shaders, giving control over notions of materials, lights, geometry and the mechanisms for determining final geometry and appearance
- possibilities and shortcomings of this approach
  - fast
  - many photorealistic effects can only be approximated (reflection, refraction, shadows) and/or handled in limited ways

Coming Up

- after fall break
  - higher level tools
    - three.js (3D scene graph API)
    - Blender ("3D creation suite" for modeling, rigging, animation, rendering, and more)
- rest of the semester
  - animation techniques
  - advanced topics
    - particle systems (modeling, animation)
    - raytracing, radiosity (rendering)