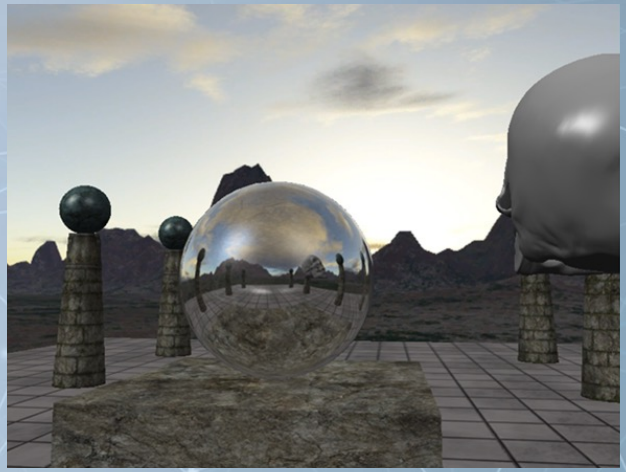
Cube Map Rendering and Dynamic Reflection

CSU0021: Computer Graphics

Cubic Environment Mapping (Cubemap)

- Applications
  - Skybox
  - Environment refraction
  - Environment reflection
  - **Dynamic reflection**
    - **(need online cube map rendering)**



Render Cube Map On-the-fly

- Set camera in your scene to render 6 images to be a cube map texture
  - Determine a camera location to render
  - The field of view must be 90 degrees
  - The aspect ratio must be 1
  - Render for 6 different view directions (and look up vectors)

If camera is at (0,0,0)

```
var ENV_CUBE_LOOK_DIR = [
    [1.0, 0.0, 0.0],    texture_cube_positive_x
    [-1.0, 0.0, 0.0],   texture_cube_negative_x
    [0.0, 1.0, 0.0],    texture_cube_positive_y
    [0.0, -1.0, 0.0],   texture_cube_negative_y
    [0.0, 0.0, 1.0],    texture_cube_positive_z
    [0.0, 0.0, -1.0]    texture_cube_negative_z
];
```

```
var ENV_CUBE_LOOK_UP = [
    [0.0, -1.0, 0.0],
    [0.0, -1.0, 0.0],
    [0.0, 0.0, 1.0],
    [0.0, 0.0, -1.0],
    [0.0, -1.0, 0.0],
    [0.0, -1.0, 0.0]
];
```

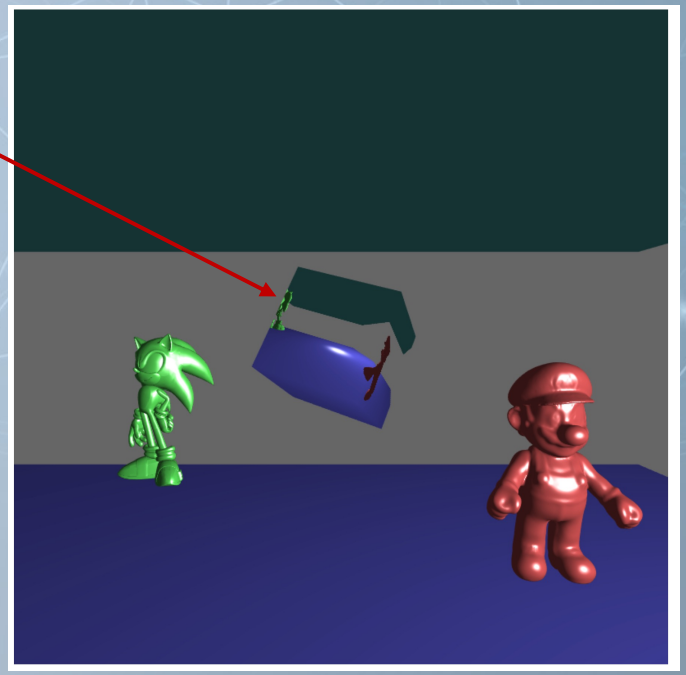
Steps of Cube Map Rendering

- You need FBO and cube map texture

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

Example (Ex12-1)

- Online render a cube map by putting the camera at the center of the cube and map it on a cube
  - **No reflection here (just texture mapping)**
- Files



Example (Ex12-1)

- main() in WebGL.js

The function to create a cube map texture and a frame buffer object for off-screen cube map rendering

```
async function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    cubeObj = await loadOBJtoCreateVBO('cube.obj');
    sonicObj = await loadOBJtoCreateVBO('sonic.obj');
    marioObj = await loadOBJtoCreateVBO('mario.obj');
    quadObj = await loadOBJtoCreateVBO('quad.obj');

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_Normal = gl.getAttribLocation(program, 'a_Normal');
    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_normalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');
    program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');
    program.u_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');
    program.u_Ks = gl.getUniformLocation(program, 'u_Ks');
    program.u_Color = gl.getUniformLocation(program, 'u_Color');
    program.u_shininess = gl.getUniformLocation(program, 'u_shininess');

    programTextureOnCube = compileShader(gl, VSHADER_SOURCE_TEXTURE_ON_CUBE, FSHADER_SOURCE_TEXTURE_ON_CUBE);
    programTextureOnCube.a_Position = gl.getAttribLocation(programTextureOnCube, 'a_Position');
    programTextureOnCube.u_MvpMatrix = gl.getUniformLocation(programTextureOnCube, 'u_MvpMatrix');
    programTextureOnCube.u_envCubeMap = gl.getUniformLocation(programTextureOnCube, 'u_envCubeMap');

    fbo = initFrameBufferForCubemapRendering(gl);

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
    document.onkeydown = function(ev){keydown(ev)};

    var tick = function() {
      rotateAngle += 0.45;
      draw();
      requestAnimationFrame(tick);
    }
    tick();
}
```

Load 3D object models

Compile two shaders

For the cube rotation animation

Example (Ex12-1)

- initFrameBufferForCubemapRendering() in WebGL.js

It basically the steps to create a FBO

But, do not bind any texture to the FBO now

```
function initFrameBufferForCubemapRendering(gl){
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  for (let i = 0; i < 6; i++) {
    gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
                  gl.RGBA, offScreenWidth, offScreenHeight, 0, gl.RGBA,
                  gl.UNSIGNED_BYTE, null);
  }

  //create and setup a render buffer as the depth buffer
  var depthBuffer = gl.createRenderbuffer();
  gl.bindRenderbuffer(gl.RENDERBUFFER, depthBuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
                         offScreenWidth, offScreenHeight);

  //create and setup framebuffer: linke the color and depth buffer to it
  var frameBuffer = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, frameBuffer);
  // gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
  //                         gl.TEXTURE_2D, texture, 0);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
                             gl.RENDERBUFFER, depthBuffer);
  frameBuffer.texture = texture;

  gl.bindFramebuffer(gl.FRAMEBUFFER, null);

  return frameBuffer;
}
```

- Steps
  - **Create a frame buffer object (does not have to bind 2D texture on it for rendering now)**
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

Example (Ex12-1)

- initFrameBufferForCubemapRendering() in WebGL.js

Create a cubemap texture

```
function initFrameBufferForCubemapRendering(gl){
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  for (let i = 0; i < 6; i++) {
    gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
                  gl.RGBA, offScreenWidth, offScreenHeight, 0, gl.RGBA,
                  gl.UNSIGNED_BYTE, null);
  }
```

Just configurate the 6 2D textures, do not bind images

```
  //create and setup a render buffer as the depth buffer
  var depthBuffer = gl.createRenderbuffer();
  gl.bindRenderbuffer(gl.RENDERBUFFER, depthBuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
                         offScreenWidth, offScreenHeight);

  //create and setup framebuffer: linke the color and depth buffer to it
  var frameBuffer = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, frameBuffer);
  // gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
  //                         gl.TEXTURE_2D, texture, 0);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
                             gl.RENDERBUFFER, depthBuffer);
  frameBuffer.texture = texture;

  gl.bindFramebuffer(gl.FRAMEBUFFER, null);

  return frameBuffer;
}
```

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - **Create a cube map texture object (does not bind any image on it now)**
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

Example (Ex12-1)

- initFrameBufferForCubemapRendering() in WebGL.js

This is one way to iterate through all targets of a texture map

```
#define GL_TEXTURE_BINDING_CUBE_MAP        0x8514
#define GL_TEXTURE_CUBE_MAP_POSITIVE_X     0x8515
#define GL_TEXTURE_CUBE_MAP_NEGATIVE_X     0x8516
#define GL_TEXTURE_CUBE_MAP_POSITIVE_Y     0x8517
#define GL_TEXTURE_CUBE_MAP_NEGATIVE_Y     0x8518
#define GL_TEXTURE_CUBE_MAP_POSITIVE_Z     0x8519
#define GL_TEXTURE_CUBE_MAP_NEGATIVE_Z     0x851A
```

```
function initFrameBufferForCubemapRendering(gl){
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  for (let i = 0; i < 6; i++) {
    gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
                  gl.RGBA, offScreenWidth, offScreenHeight, 0, gl.RGBA,
                  gl.UNSIGNED_BYTE, null);
  }

  //create and setup a render buffer as the depth buffer
  var depthBuffer = gl.createRenderbuffer();
  gl.bindRenderbuffer(gl.RENDERBUFFER, depthBuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
                         offScreenWidth, offScreenHeight);

  //create and setup framebuffer: linke the color and depth buffer to it
  var frameBuffer = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, frameBuffer);
  // gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
  //                         gl.TEXTURE_2D, texture, 0);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
                             gl.RENDERBUFFER, depthBuffer);
  frameBuffer.texture = texture;

  gl.bindFramebuffer(gl.FRAMEBUFFER, null);

  return frameBuffer;
}
```

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - **Create a cube map texture object (does not bind any image on it now)**
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

Example (Ex12-1)

- initFrameBufferForCubemapRendering() in WebGL.js

They are 256x256.
”If” we just use this cube map rendering result for reflection. It does not need high quality images on the cubemap texture.

```
function initFrameBufferForCubemapRendering(gl){
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  for (let i = 0; i < 6; i++) {
    gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
                  gl.RGBA, offScreenWidth, offScreenHeight, 0, gl.RGBA,
                  gl.UNSIGNED_BYTE, null);
  }

  //create and setup a render buffer as the depth buffer
  var depthBuffer = gl.createRenderbuffer();
  gl.bindRenderbuffer(gl.RENDERBUFFER, depthBuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
                         offScreenWidth, offScreenHeight);

  //create and setup framebuffer: linke the color and depth buffer to it
  var frameBuffer = gl.createFramebuffer();
  gl.bindFramebuffer(gl.FRAMEBUFFER, frameBuffer);
  // gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
  //                         gl.TEXTURE_2D, texture, 0);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
                             gl.RENDERBUFFER, depthBuffer);
  frameBuffer.texture = texture;

  gl.bindFramebuffer(gl.FRAMEBUFFER, null);

  return frameBuffer;
}
```

Put FBO and the cubemap texture together for easy access later

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - **Create a cube map texture object (does not bind any image on it now)**
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

Example (Ex12-1)

- renderCubMap() in WebGL.js

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - **Bind the fbo as the destination of rendering**
  - **Set the viewport**
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

```
function renderCubeMap(camX, camY, camZ)
{
  var ENV_CUBE_LOOK_DIR = [
      [1.0, 0.0, 0.0],
      [-1.0, 0.0, 0.0],
      [0.0, 1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0]
  ];

  var ENV_CUBE_LOOK_UP = [
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0],
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0]
  ];

  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.4, 0.4, 0.4,1);
  for (var side = 0; side < 6;side++){
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                            gl.TEXTURE_CUBE_MAP_POSITIVE_X+side,
                            fbo.texture, 0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let vpMatrix = new Matrix4();
    vpMatrix.setPerspective(90, 1, 1, 100);
    vpMatrix.lookAt(camX, camY, camZ,
                    camX + ENV_CUBE_LOOK_DIR[side][0],
                    camY + ENV_CUBE_LOOK_DIR[side][1],
                    camZ + ENV_CUBE_LOOK_DIR[side][2],
                    ENV_CUBE_LOOK_UP[side][0],
                    ENV_CUBE_LOOK_UP[side][1],
                    ENV_CUBE_LOOK_UP[side][2]);

    drawRegularObjects(vpMatrix);
  }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Camera center position in world space

Camera view directions

Camera look up vectors

Example (Ex12-1)

- renderCubMap() in WebGL.js

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - **For each camera direction (we have 6)**
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

```
function renderCubeMap(camX, camY, camZ)
{
  var ENV_CUBE_LOOK_DIR = [
      [1.0, 0.0, 0.0],
      [-1.0, 0.0, 0.0],
      [0.0, 1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0]
  ];

  var ENV_CUBE_LOOK_UP = [
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0],
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0]
  ];

  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.4, 0.4, 0.4,1);
  for (var side = 0; side < 6;side++){
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                            gl.TEXTURE_CUBE_MAP_POSITIVE_X+side,
                            fbo.texture, 0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let vpMatrix = new Matrix4();
    vpMatrix.setPerspective(90, 1, 1, 100);
    vpMatrix.lookAt(camX, camY, camZ,
                    camX + ENV_CUBE_LOOK_DIR[side][0],
                    camY + ENV_CUBE_LOOK_DIR[side][1],
                    camZ + ENV_CUBE_LOOK_DIR[side][2],
                    ENV_CUBE_LOOK_UP[side][0],
                    ENV_CUBE_LOOK_UP[side][1],
                    ENV_CUBE_LOOK_UP[side][2]);

    drawRegularObjects(vpMatrix);
  }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Example (Ex12-1)

- renderCubMap() in WebGL.js

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - **Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)**
    - Clear the color buffer and depth buffer
    - Set the proper mvp matrix
    - Call shader to render

```
function renderCubeMap(camX, camY, camZ)
{
  var ENV_CUBE_LOOK_DIR = [
      [1.0, 0.0, 0.0],
      [-1.0, 0.0, 0.0],
      [0.0, 1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0]
  ];

  var ENV_CUBE_LOOK_UP = [
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0],
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0]
  ];

  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.4, 0.4, 0.4,1);
  for (var side = 0; side < 6;side++){
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                            gl.TEXTURE_CUBE_MAP_POSITIVE_X+side,
                            fbo.texture, 0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let vpMatrix = new Matrix4();
    vpMatrix.setPerspective(90, 1, 1, 100);
    vpMatrix.lookAt(camX, camY, camZ,
                    camX + ENV_CUBE_LOOK_DIR[side][0],
                    camY + ENV_CUBE_LOOK_DIR[side][1],
                    camZ + ENV_CUBE_LOOK_DIR[side][2],
                    ENV_CUBE_LOOK_UP[side][0],
                    ENV_CUBE_LOOK_UP[side][1],
                    ENV_CUBE_LOOK_UP[side][2]);

    drawRegularObjects(vpMatrix);
  }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Example (Ex12-1)

- renderCubMap() in WebGL.js

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - **Clear the color buffer and depth buffer**
    - Set the proper mvp matrix
    - Call shader to render

```
function renderCubeMap(camX, camY, camZ)
{
  var ENV_CUBE_LOOK_DIR = [
      [1.0, 0.0, 0.0],
      [-1.0, 0.0, 0.0],
      [0.0, 1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0]
  ];

  var ENV_CUBE_LOOK_UP = [
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0],
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0]
  ];

  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.4, 0.4, 0.4,1);
  for (var side = 0; side < 6;side++){
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                            gl.TEXTURE_CUBE_MAP_POSITIVE_X+side,
                            fbo.texture, 0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let vpMatrix = new Matrix4();
    vpMatrix.setPerspective(90, 1, 1, 100);
    vpMatrix.lookAt(camX, camY, camZ,
                    camX + ENV_CUBE_LOOK_DIR[side][0],
                    camY + ENV_CUBE_LOOK_DIR[side][1],
                    camZ + ENV_CUBE_LOOK_DIR[side][2],
                    ENV_CUBE_LOOK_UP[side][0],
                    ENV_CUBE_LOOK_UP[side][1],
                    ENV_CUBE_LOOK_UP[side][2]);

    drawRegularObjects(vpMatrix);
  }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Example (Ex12-1)

- renderCubMap() in WebGL.js
- **After these steps, we already have a cube map texture ready for use**

- Steps
  - Create a frame buffer object (does not have to bind 2D texture on it for rendering now)
  - Create a cube map texture object (does not bind any image on it now)
  - Bind the fbo as the destination of rendering
  - Set the viewport
  - For each camera direction (we have 6)
    - Bind a face of the cube map texture to the frame buffer as a color buffer (for rendering output)
    - Clear the color buffer and depth buffer
    - **Set the proper mvp matrix**
    - **Call shader to render**

```
function renderCubeMap(camX, camY, camZ)
{
  var ENV_CUBE_LOOK_DIR = [
      [1.0, 0.0, 0.0],
      [-1.0, 0.0, 0.0],
      [0.0, 1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0]
  ];

  var ENV_CUBE_LOOK_UP = [
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0],
      [0.0, 0.0, 1.0],
      [0.0, 0.0, -1.0],
      [0.0, -1.0, 0.0],
      [0.0, -1.0, 0.0]
  ];

  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.4, 0.4, 0.4,1);
  for (var side = 0; side < 6;side++){
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
                            gl.TEXTURE_CUBE_MAP_POSITIVE_X+side,
                            fbo.texture, 0);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    let vpMatrix = new Matrix4();
    vpMatrix.setPerspective(90, 1, 1, 100);
    vpMatrix.lookAt(camX, camY, camZ,
                    camX + ENV_CUBE_LOOK_DIR[side][0],
                    camY + ENV_CUBE_LOOK_DIR[side][1],
                    camZ + ENV_CUBE_LOOK_DIR[side][2],
                    ENV_CUBE_LOOK_UP[side][0],
                    ENV_CUBE_LOOK_UP[side][1],
                    ENV_CUBE_LOOK_UP[side][2]);

    drawRegularObjects(vpMatrix);
  }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

Render the cube map at (0, 0, 0). You probably want to render the cube map from different camera location. If so, you should change this camera position)

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic

  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

For on-screen rendering
Set up viewport and background color.
Clear color and depth buffer and
enable the depth test.

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic

  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

Set up space transformation matrices for rendering

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic

  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

```
function drawRegularObjects(vpMatrix){
  let mdlMatrix = new Matrix4();

  mdlMatrix.setTranslate(-1.0, -2.0, 4.0);
  mdlMatrix.scale(8, 1,8);
  drawOneRegularObject(quadObj, mdlMatrix, vpMatrix, 0.4, 0.4, 1.0);

  mdlMatrix.setTranslate(-1.0, 3.0, 4.0);
  mdlMatrix.scale(8, 1,8);
  drawOneRegularObject(quadObj, mdlMatrix, vpMatrix, 0.4, 1.0, 1.0);

  mdlMatrix.setTranslate(-2.0, -0.5, 2.0);
  mdlMatrix.scale(0.05, 0.05,0.05);
  drawOneRegularObject(sonicObj, mdlMatrix, vpMatrix, 0.4, 1.0, 0.4);

  mdlMatrix.setTranslate(2.5, -0.5, 1.5);
  mdlMatrix.scale(0.02, 0.02,0.02);
  drawOneRegularObject(marioObj, mdlMatrix, vpMatrix, 1.0, 0.4, 0.4);
}
```

```
function drawOneRegularObject(obj, modelMatrix, vpMatrix, colorR, colorG, colorB){
  gl.useProgram(program);
  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniform3f(program.u_LightPosition, lightX, lightY, lightZ);
  gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
  gl.uniform1f(program.u_Ka, 0.2);
  gl.uniform1f(program.u_Kd, 0.7);
  gl.uniform1f(program.u_Ks, 1.0);
  gl.uniform1f(program.u_shininess, 10.0);
  gl.uniform3f(program.u_Color, colorR, colorG, colorB);

  gl.uniformMatrix4fv(program.u_mvpMatrix, false, mvpMatrix.elements);
  gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
  gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, obj[i].vertexBuffer);
    initAttributeVariable(gl, program.a_Normal, obj[i].normalBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

Render all objects (without the cube)

And, this is also called by renderCubeMap() (view matrix are different in renderCubeMap())

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic

  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

```
function drawOneObjectWithCubemapTexture(obj, modelMatrix, vpMatrix){
  gl.useProgram(programTextureOnCube);

  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniformMatrix4fv(programTextureOnCube.u_MvpMatrix, false, mvpMatrix.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, fbo.texture);
  gl.uniform1i(programTextureOnCube.u_envCubeMap, 0);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, programTextureOnCube.a_Position, obj[i].vertexBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic
  Draw the cube and use the rendered cube map texture to color it
  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- draw () in WebGL.js
  - Big picture of rendering a frame

```
function drawOneObjectWithCubemapTexture(obj, modelMatrix, vpMatrix){
  gl.useProgram(programTextureOnCube);
  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniformMatrix4fv(programTextureOnCube.u_MvpMatrix, false, mvpMatrix.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, fbo.texture);
  gl.uniform1i(programTextureOnCube.u_envCubeMap, 0);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, programTextureOnCube.a_Position, obj[i].vertexBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

The shader program to render the cube

Pass the rendered cube map texture to shader

```
function draw(){
  renderCubeMap(0,0,0);//we have bind fbo back to null in function

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ceiling, ground, mario and sonic

  //the cube
  let mdlMatrix = new Matrix4();
  mdlMatrix.setTranslate(0.0, 0.8, 0.0);
  mdlMatrix.scale(0.8, 0.8, 0.8);
  mdlMatrix.rotate(rotateAngle, 1, 1, 1);
  drawOneObjectWithCubemapTexture(cubeObj, mdlMatrix, vpMatrix);
}
```

Example (Ex12-1)

- The first shader in WebGL.js
  - The shader to render all objects except for the cube
  - Just a regular object rendering shader with illumination

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_Normal = normalize(vec3(u_normalMatrix * a_Normal));
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform vec3 u_Color;
    uniform float u_shininess;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    void main(){
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color.rgb;
        vec3 diffuseLightColor = u_Color.rgb;
        // assume white specular light (you can also input it from ouside)
        vec3 specularLightColor = vec3(1.0, 1.0, 1.0);

        vec3 ambient = ambientLightColor * u_Ka;

        vec3 normal = normalize(v_Normal);
        vec3 lightDirection = normalize(u_LightPosition - v_PositionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        vec3 specular = vec3(0.0, 0.0, 0.0);
        if(nDotL > 0.0) {
            vec3 R = reflect(-lightDirection, normal);

            vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
            float specAngle = clamp(dot(R, V), 0.0, 1.0);
            specular = u_Ks * pow(specAngle, u_shininess) * specularLightColor;
        }

        gl_FragColor = vec4( ambient + diffuse + specular, 1.0 );
    }
`;
```

Example (Ex12-1)

- The second shader in WebGL.js
  - Use the rendered cube map texture to color the cube
  - Exactly the same as the shader for cube map texture mapping (Ex10-1)

```
var VSHADER_SOURCE_TEXTURE_ON_CUBE = `
  attribute vec4 a_Position;
  varying vec4 v_TexCoord;
  uniform mat4 u_MvpMatrix;
  void main() {
    gl_Position = u_MvpMatrix * a_Position;
    v_TexCoord = a_Position;
  }
`;

var FSHADER_SOURCE_TEXTURE_ON_CUBE = `
  precision mediump float;
  varying vec4 v_TexCoord;
  uniform samplerCube u_envCubeMap;
  void main() {
    gl_FragColor = textureCube(u_envCubeMap, v_TexCoord.stp);
  }
`;
```

Try and Think (5mins)

- Download and run the code

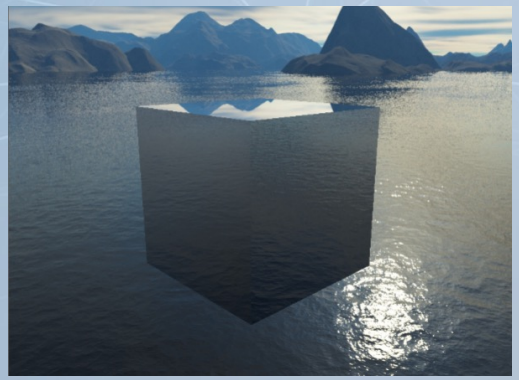
- Make sure you know all steps in this example
  - The idea of this example is not hard
  - But, the implementation is complicated because it almost needs all we have learned this semester

Dynamic Reflection

- Dynamic Reflection
  - Not only reflect the background, but also reflect any 3D object in the scene
- It is straight forward after we know how to create a cube map texture on-the-fly

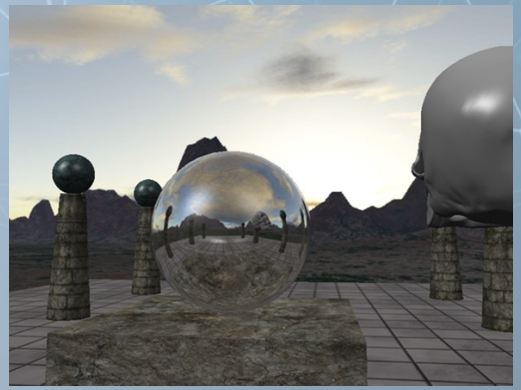
Steps of environment cubemap reflection

1. Load images to create a cube map texture (environment background)
2. Look up this cube map texture to color the reflective object



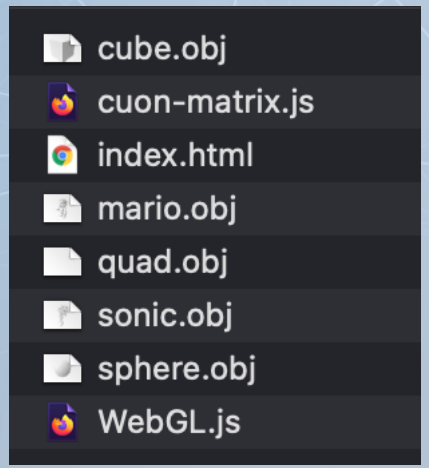
Steps of dynamic reflection

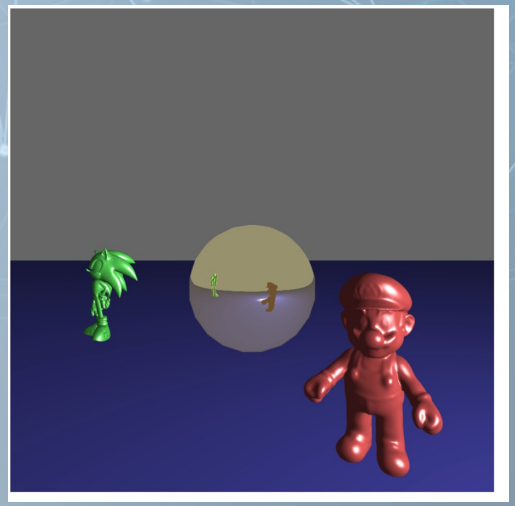
1. Put the camera at center of the reflective object and create the cube map texture on-the-fly
2. Look up this cube map texture to color the reflective object



Example (Ex12-2)

- Dynamic reflection
  - Compare with Ex12-1,
    - we just replace the cube with a sphere, remove the ceiling, make mario and sonic rotate around the sphere **(these all are easy)**
    - Implement reflection on the sphere instead of direct texture mapping
- Files





Example (Ex12-2)

- draw() in WebGL.js

Render the cube map at (0, 0, 0)
because the sphere is at (0, 0, 0)

For on-screen rendering
Set up viewport and background color.
Clear color and depth buffer and enable the depth test.

Set up space transformation matrices to render non-reflective objects

```
function draw(){
  renderCubeMap(0, 0, 0);

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ground, mario, sonic

  //the sphere
  let mdlMatrix = new Matrix4();
  mdlMatrix.setScale(0.5, 0.5, 0.5);
  drawObjectWithDynamicReflection(sphereObj, mdlMatrix, vpMatrix, 0.95, 0.85, 0.4);
}
```

Example (Ex12-2)

- draw() in WebGL.js

```
function draw(){
  renderCubeMap(0, 0, 0);

  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let rotateMatrix = new Matrix4();
  rotateMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  rotateMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  var viewDir= new Vector3([cameraDirX, cameraDirY, cameraDirZ]);
  var newViewDir = rotateMatrix.multiplyVector3(viewDir);
  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  cameraX + newViewDir.elements[0],
                  cameraY + newViewDir.elements[1],
                  cameraZ + newViewDir.elements[2],
                  0, 1, 0);

  drawRegularObjects(vpMatrix);//ground, mario, sonic

  //the sphere
  let mdlMatrix = new Matrix4();
  mdlMatrix.setScale(0.5, 0.5, 0.5);
  drawObjectWithDynamicReflection(sphereObj, mdlMatrix, vpMatrix, 0.95, 0.85, 0.4);
}
```

Render the reflective sphere

Example (Ex12-2)

- shader for reflection rendering () in WebGL.js
  - This shader is almost the same as the shader for environment cubemap reflection

```
var VSHADER_SOURCE_TEXTURE_ON_CUBE = `
  attribute vec4 a_Position;
  attribute vec4 a_Normal;
  uniform mat4 u_MvpMatrix;
  uniform mat4 u_modelMatrix;
  uniform mat4 u_normalMatrix;
  varying vec3 v_Normal;
  varying vec3 v_PositionInWorld;
  void main() {
    gl_Position = u_MvpMatrix * a_Position;
    v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
    v_Normal = normalize(vec3(u_normalMatrix * a_Normal));
  }
`;

var FSHADER_SOURCE_TEXTURE_ON_CUBE = `
  precision mediump float;
  uniform vec3 u_ViewPosition;
  uniform vec3 u_Color;
  uniform samplerCube u_envCubeMap;
  varying vec3 v_Normal;
  varying vec3 v_PositionInWorld;
  void main() {
    vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
    vec3 normal = normalize(v_Normal);
    vec3 R = reflect(-V, normal);
    gl_FragColor = vec4(0.78 * textureCube(u_envCubeMap, R).rgb + 0.3 * u_Color, 1.0);
  }
`;
```

We mix the object color and the reflective color

Try and Think (5mins)

- Download and run the code
- Rendering cube map for every frame is costly. Do we really have to render the cube map texture for every frame if most of the objects in the scene are static
- What if we render a cube map texture then use it on neighboring reflective objects?
- Can we use the same scheme to implement dynamic refraction?