Environment Reflection/Refraction and Fog

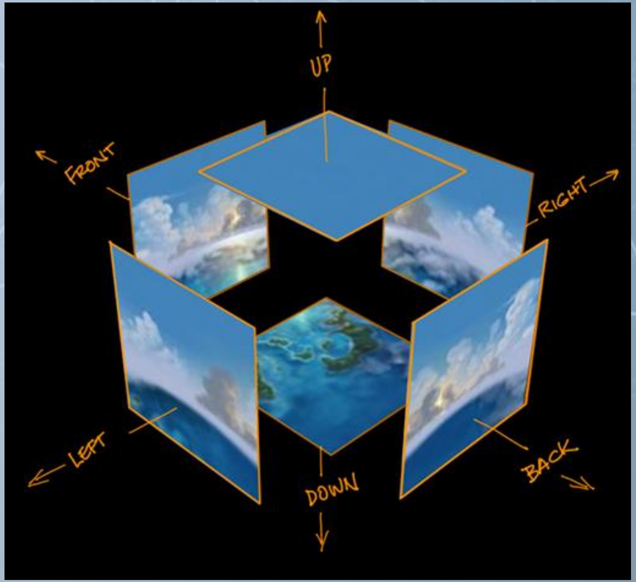
CSU0021: Computer Graphics

Cubic Environment Mapping (Cubemap)

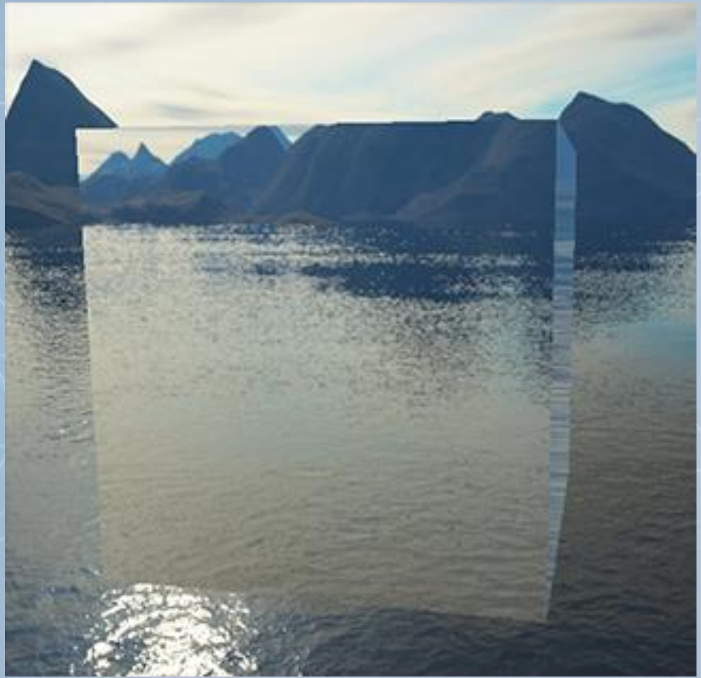
- Applications
  - Skybox (last topic)
  - **Environment refraction**
  - **Environment reflection**
  - Dynamic reflection (next topic)
- Fog?
  - Fog implementation is nothing about cubemap

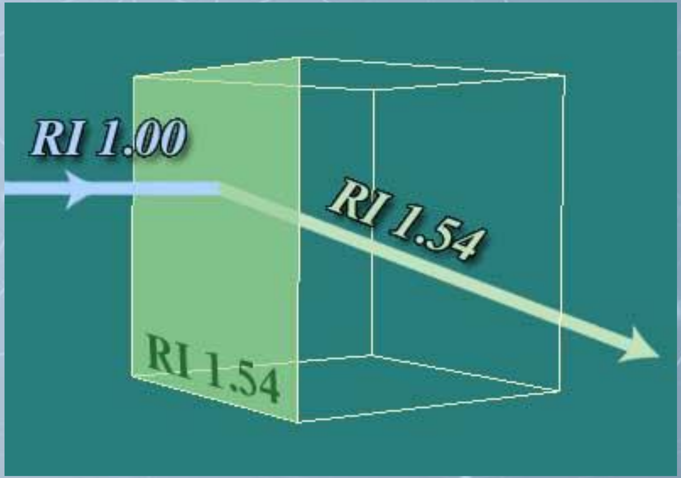
Skybox

- The background comes from cube map images

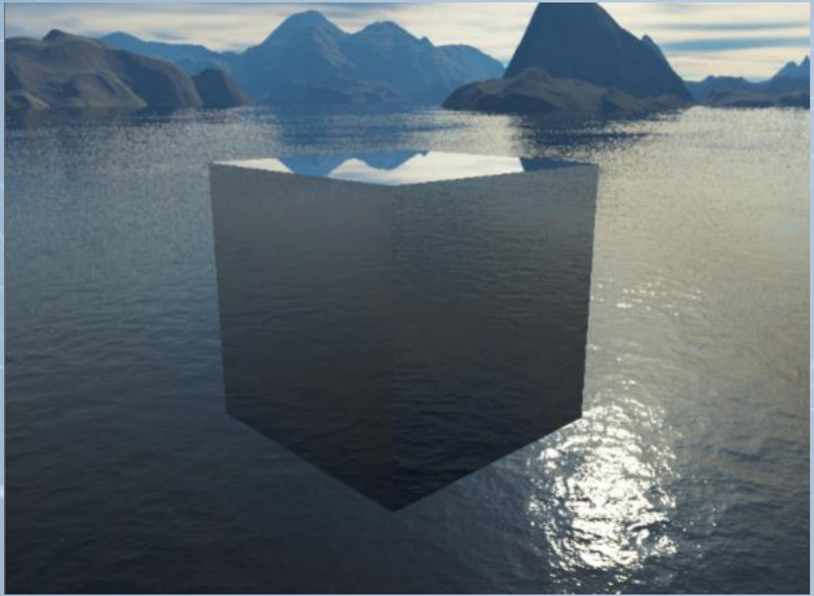


Environment Refraction





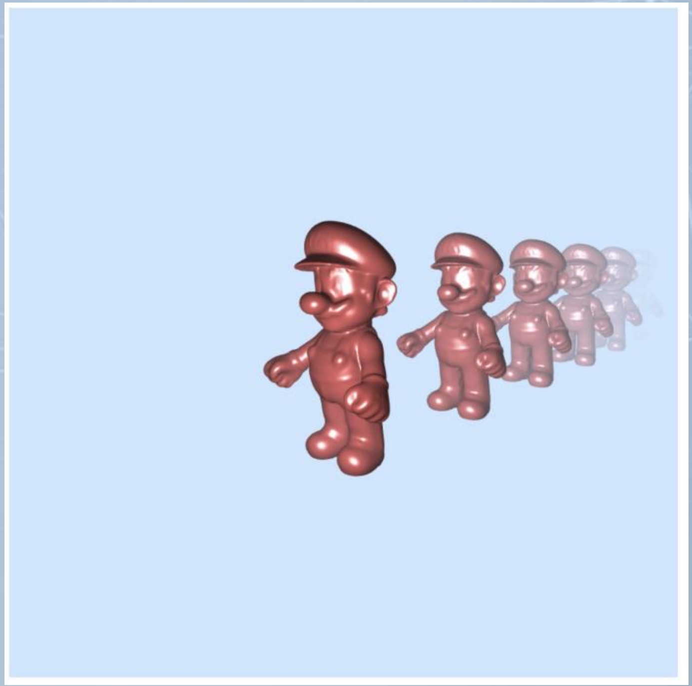
Environment Reflection



Dynamic Reflection

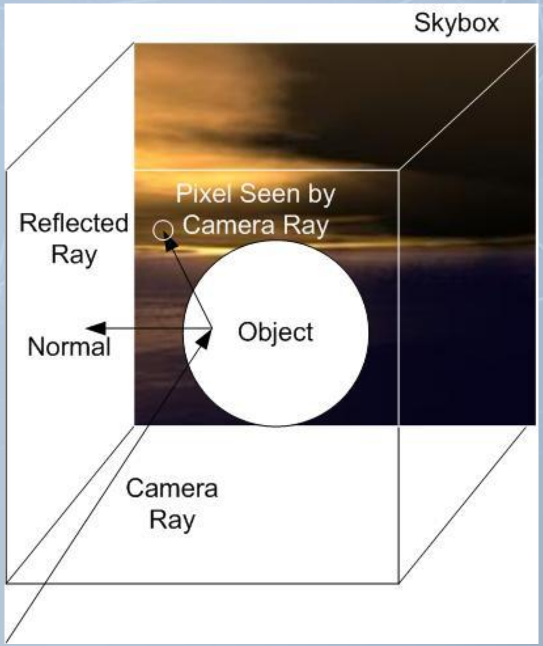
Fog

- Fog implementation does not use cube map



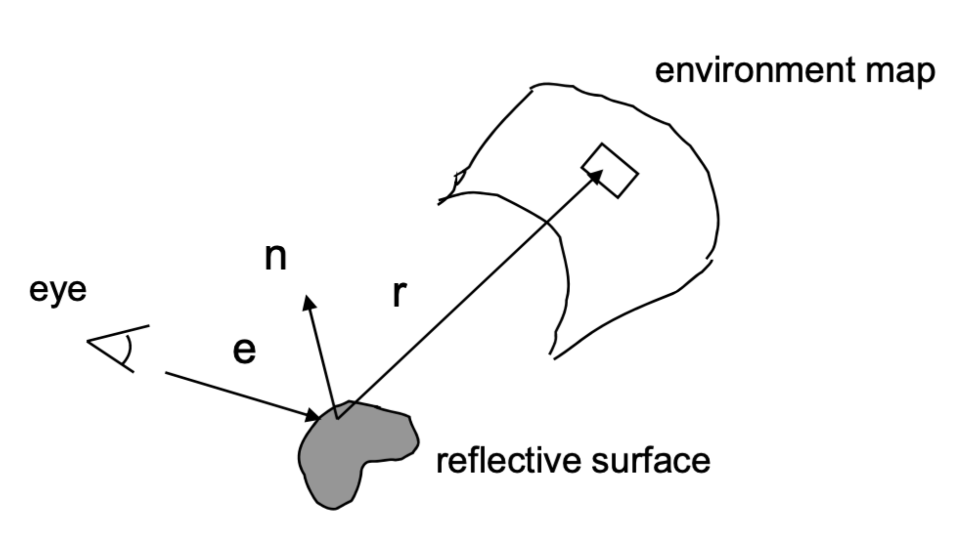
Environment Map Reflection

- A cheap way to implement reflection
- But, it does not reflect any 3D model in the scene



Basic Idea

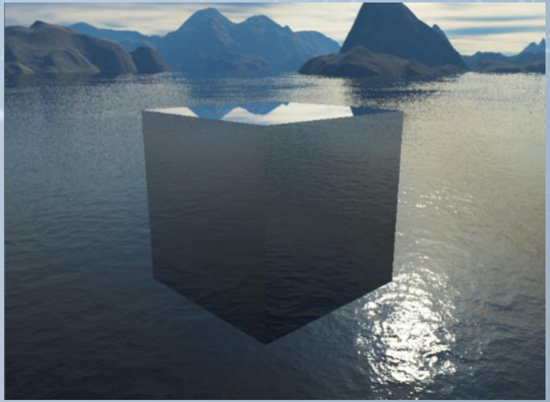
- Assuming the environment is far away and the object does not reflect itself and other objects
  - The reflection at a point can be solely decided by the reflection vector



Basic Steps

- Load images to create an environment cube map texture
- For each pixel on a reflective object, get its normal vector
- Compute the reflection vector based on the eye position and the normal vector
- Use the reflection vector to compute an index into the environment cube map texture
- Use the corresponding texel to color the pixel

- Note: you can have the environment reflection on an object without the environment background in the scene



Example (Ex11-1)

- A reflective cube

- Files
  - cube.obj
  - cuon-matrix.js
  - index.html
  - neg-x.jpg
  - neg-y.jpg
  - neg-z.jpg
  - pos-x.jpg
  - pos-y.jpg
  - pos-z.jpg
  - WebGL.js

Example (Ex11-1)

- Shaders in WebGL.js

Draw the reflective cube

```
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_ViewPosition;
    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(textureCube(u_envCubeMap, R).rgb, 1.0);
    }
`;
```

Calculate the vertex position in clip space

Calculate the vertex position in world space

Transform normal vector to world space

```
var VSHADER_SOURCE_ENVCUBE = `
  attribute vec4 a_Position;
  varying vec4 v_Position;
  void main() {
    v_Position = a_Position;
    gl_Position = a_Position;
  }
`;

var FSHADER_SOURCE_ENVCUBE = `
  precision mediump float;
  uniform samplerCube u_envCubeMap;
  uniform mat4 u_viewDirectionProjectionInverse;
  varying vec4 v_Position;
  void main() {
    vec4 t = u_viewDirectionProjectionInverse * v_Position;
    gl_FragColor = textureCube(u_envCubeMap, normalize(t.xyz / t.w));
  }
`;
```

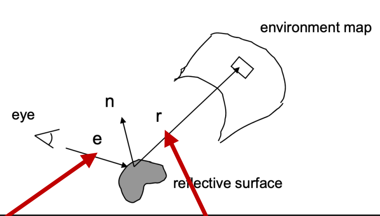
Shader to draw the background quad (same as Ex10-3)

Example (Ex11-1)

- Shaders in WebGL.js

Draw the reflective cube

```
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_ViewPosition;
    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(textureCube(u_envCubeMap, R).rgb, 1.0);
    }
`;
```

Calculate the reflective vector

Calculate the vector from an object point to eye

Use the reflective vector to look up color from the cubemap to color the cube

```
var VSHADER_SOURCE_ENVCUBE = `
  attribute vec4 a_Position;
  varying vec4 v_Position;
  void main() {
    v_Position = a_Position;
    gl_Position = a_Position;
  }
`;

var FSHADER_SOURCE_ENVCUBE = `
  precision mediump float;
  uniform samplerCube u_envCubeMap;
  uniform mat4 u_viewDirectionProjectionInverse;
  varying vec4 v_Position;
  void main() {
    vec4 t = u_viewDirectionProjectionInverse * v_Position;
    gl_FragColor = textureCube(u_envCubeMap, normalize(t.xyz / t.w));
  }
`;
```

Example (Ex11-1)

- main() in WebGL.js

Load the cube map images and create a cubemap texture

```
    cubeMapTex = initCubeTexture("pos-x.jpg", "neg-x.jpg", "pos-y.jpg", "neg-y.jpg",
                                 "pos-z.jpg", "neg-z.jpg", 512, 512)

    quadObj = initVertexBufferForLaterUse(gl, quad);

    gl.enable(gl.DEPTH_TEST);

    draw();//draw it once before mouse move

    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 += 0.25;
      draw();
      requestAnimationFrame(tick);
    }
    tick();
}
```

Make the cube rotate

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

    var quad = new Float32Array(
      [
        -1, -1, 1,
         1, -1, 1,
        -1,  1, 1,
        -1,  1, 1,
         1, -1, 1,
         1,  1, 1
      ]); //just a quad

    programEnvCube = compileShader(gl, VSHADER_SOURCE_ENVCUBE, FSHADER_SOURCE_ENVCUBE);
    programEnvCube.a_Position = gl.getAttribLocation(programEnvCube, 'a_Position');
    programEnvCube.u_envCubeMap = gl.getUniformLocation(programEnvCube, 'u_envCubeMap');
    programEnvCube.u_viewDirectionProjectionInverse = gl.getUniformLocation(programEnvCube, 'u_viewDirectionProjectionInverse');

    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_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');
    program.u_envCubeMap = gl.getUniformLocation(program, 'u_envCubeMap');

    response = await fetch('cube.obj');
    text = await response.text();
    obj = parseOBJ(text);

    for( let i=0; i < obj.geometries.length; i ++ ){
      let o = initVertexBufferForLaterUse(gl,
                                          obj.geometries[i].data.position,
                                          obj.geometries[i].data.normal,
                                          obj.geometries[i].data.texcoord);
      objComponents.push(o);
    }
```

Background quad vertices

Compile two shaders

Example (Ex11-1)

- draw() in WebGL.js

Just call shader to draw the cube

```
//Draw the reflective cube
gl.useProgram(program);
gl.depthFunc(gl.LESS);
//model Matrix (part of the mvp matrix)
var modelMatrix = new Matrix4();
modelMatrix.setScale(objScale, objScale, objScale);
modelMatrix.rotate(rotateAngle, 1, 1, 1); //make the cube rotate
//mvp: projection * view * model matrix
var mvpMatrix = new Matrix4();
mvpMatrix.set(projMatrix).multiply(viewMatrix).multiply(modelMatrix);

//normal matrix
var normalMatrix = new Matrix4();
normalMatrix.setInverseOf(modelMatrix);
normalMatrix.transpose();

gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
gl.uniform1i(program.u_envCubeMap, 0);

gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMapTex);

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

Make the cube keep rotating

This shader also need the environment cube map texture

Prepare matrices and draw the background quad (almost same as Ex-10-3)

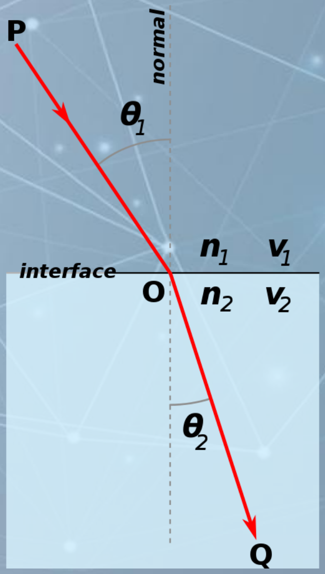
```
function draw(){
  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4, 0.4, 0.4, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  //rotate the camera view direction
  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);

  var viewMatrix = new Matrix4();
  var projMatrix = new Matrix4();
  projMatrix.setPerspective(60, 1, 1, 15);
  viewMatrix.setLookAt(cameraX, cameraY, cameraZ,
                       cameraX + newViewDir.elements[0],
                       cameraY + newViewDir.elements[1],
                       cameraZ + newViewDir.elements[2],
                       0, 1, 0);
  var viewMatrixRotationOnly = new Matrix4();
  viewMatrixRotationOnly.set(viewMatrix);
  viewMatrixRotationOnly.elements[12] = 0; //ignore translation
  viewMatrixRotationOnly.elements[13] = 0;
  viewMatrixRotationOnly.elements[14] = 0;
  var vpFromCameraRotationOnly = new Matrix4();
  vpFromCameraRotationOnly.set(projMatrix).multiply(viewMatrixRotationOnly);
  var vpFromCameraInverse = vpFromCameraRotationOnly.invert();

  //draw the background quad
  gl.useProgram(programEnvCube);
  gl.depthFunc(gl.LEQUAL);
  gl.uniformMatrix4fv(programEnvCube.u_viewDirectionProjectionInverse,
                      false, vpFromCameraInverse.elements);
  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMapTex);
  gl.uniform1i(programEnvCube.u_envCubeMap, 0);
  initAttributeVariable(gl, programEnvCube.a_Position, quadObj.vertexBuffer);
  gl.drawArrays(gl.TRIANGLES, 0, quadObj.numVertices);
```

Try (5mins)

- If you comment the code to draw the background quad in draw() (Line244~Line253), can we still see the reflection on the cube?

Environment Refraction

- What is it?
- Bending of light as it passes from one medium with **refraction index** n_1, to the other medium with **refraction index** n_2
  - Snell's law
    - $\frac{sin\theta_1}{sin\theta_2} = \frac{n_1}{n_2}$
  - List of refractive indices
    - https://en.wikipedia.org/wiki/List_of_refractive_indices
  - Use the refraction index to determine the Incident angle and refractive angle



- Same, neighboring 3D objects do not have impact on environment refraction

Example (Ex11-2)

- A refractive cube

- Files

cube.obj
cuon-matrix.js
index.html
neg-x.jpg
neg-y.jpg
neg-z.jpg
pos-x.jpg
pos-y.jpg
pos-z.jpg
WebGL.js

Example (Ex11-2)

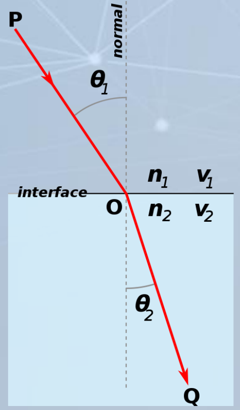
- Ex11-2 and Ex11-1 are almost the same, the only differences is this small segment of code in this shader
  - The shader to draw the cube

```
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_ViewPosition;
    uniform samplerCube u_envCubeMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    void main(){
      float ratio = 1.00 / 1.1; //glass
      vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
      vec3 normal = normalize(v_Normal);
      vec3 R = refract(-V, normal, ratio);
      gl_FragColor = vec4(textureCube(u_envCubeMap, R).rgb, 1.0);
    }
`;
```

Example (Ex11-2)

- refract(I, N, eta)
  - https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/refract.xhtml
  - I: incident vector
  - N: normal vector
  - eta: ratio of indices of refraction

$$\frac{sin\theta_1}{sin\theta_2} = \boxed{\frac{n_1}{n_2}}$$

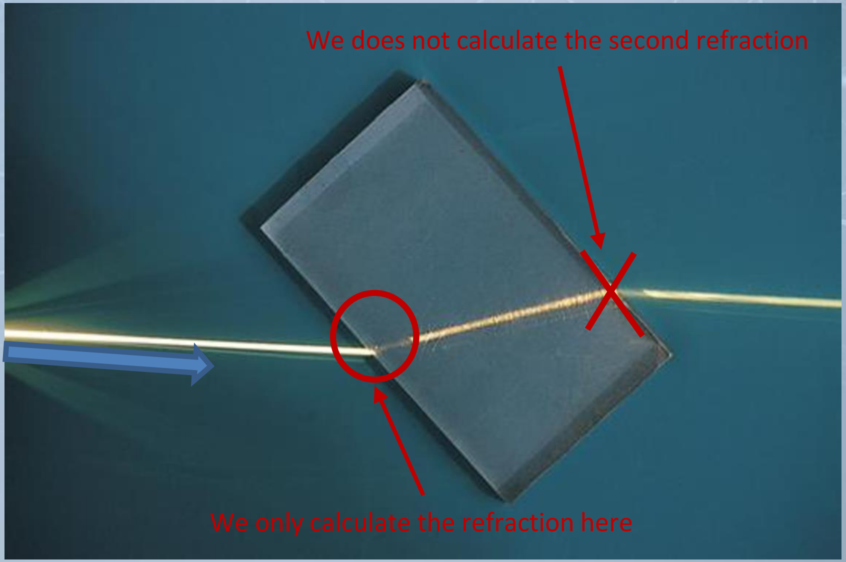



```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;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

var FSHADER_SOURCE = `
    precision mediump float;
    uniform vec3 u_ViewPosition;
    uniform samplerCube u_envCubeMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    void main(){
      float ratio = 1.00 / 1.1; //glass
      vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
      vec3 normal = normalize(v_Normal);
      vec3 R = refract(-V, normal, ratio);
      gl_FragColor = vec4(textureCube(u_envCubeMap, R).rgb, 1.0);
    }
`;
```

We assume the material of the cube is glass (refractive index is 1.1)

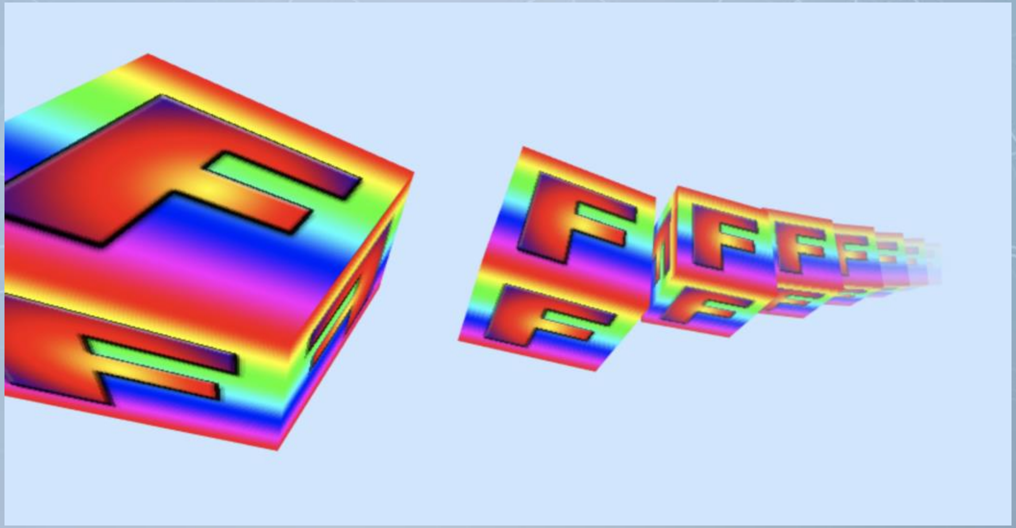
Try (5mins)

- Download the code and run
- Is this refraction implementation is good enough?
  - We only calculate the refraction once



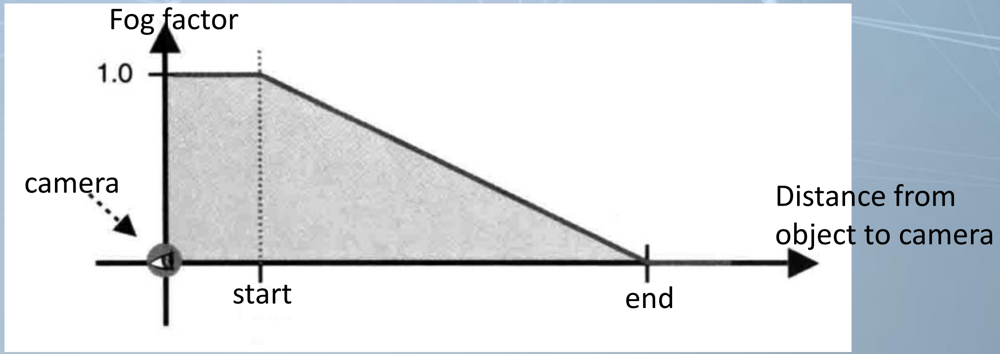
Fog

- Idea: if the object is further away from the camera, it should be blurred more
- We do not need cube map and texture for fog implementation



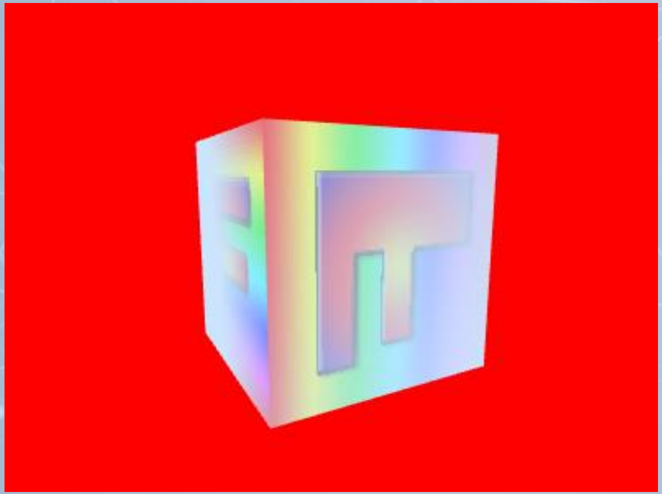
Linear Fog

- Linear Fog: a simple version of fog implementation
- We use "fog factor" to control how much we should blur the object
- Fog factor is determined by the distance from the object to the camera
  - **fogFactor = (end – distance(camera, objPoint)) / (end - start)**
    - start: the closest distance we want the fog effect
    - end: the farthest distance we still can see the object (a little bit)
  - **color = (fogFactor * objColor) + ((1-fogFactor) * fogColor)**



Fog Color

- Key: Fog color should be the same as the background color



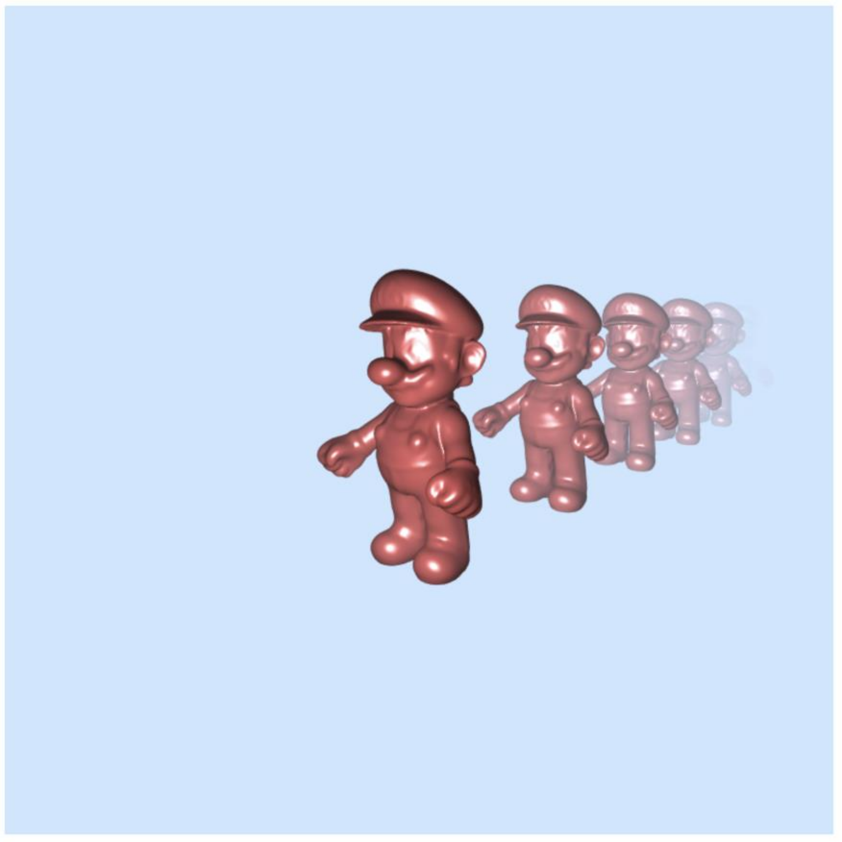
Background color and
fog color are different

Background color and
fog color are the same

Example (Ex11-3)

- Put marios in fog

- Files
  - cuon-matrix.js
  - index.html
  - mario.obj
  - WebGL.js



Example (Ex11-3)

- draw() in WebGL.js
  - draw six marios

```
functio any v(){
    gl.clearColor(0.8, 0.9, 1.0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    for( let depth = 0; depth< 6; depth++){
      //model Matrix (part of the mvp matrix)
      modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
      modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
      modelMatrix.rotate(-30, 0, 1, 0);
      modelMatrix.translate(0, 0, -3 * depth);
      modelMatrix.scale(objScale, objScale, objScale);
      //mvp: projection * view * model matrix
      mvpMatrix.setPerspective(70, 1, 1, 100);
      mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
      mvpMatrix.multiply(modelMatrix);

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

      gl.uniform3f(program.u_LightPosition, 0, 0, 3);
      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, 1.0, 0.4, 0.4);
      gl.uniform3f(program.u_EyePos, cameraX, cameraY, cameraZ);

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

Example (Ex11-3)

- shaders in WebGL.js

Calculate the distance between an object point to the camera in vertex shader

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

```
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 float u_shininess;
    uniform vec3 u_Color;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying float v_Dist;
    void main(){
        // let ambient and diffuse color are u_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        // 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);
            // V: the vector, point to viewer
            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;
        }

        vec3 illumination = ambient + diffuse + specular;

        //fog
        vec3 fogColor = vec3(0.8, 0.9, 1.0);
        vec2 fogDist = vec2(5.0, 20.0);
        float fogFactor = clamp((fogDist.y - v_Dist)/(fogDist.y - fogDist.x), 0.0, 1.0);

        gl_FragColor = vec4( fogColor * (1.0 - fogFactor) + illumination * fogFactor, 1.0 );
    }
`;
```

Example (Ex11-3)

- shaders in WebGL.js

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

```
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 float u_shininess;
    uniform vec3 u_Color;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying float v_Dist;
    void main(){
        // let ambient and diffuse color are u_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        // 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);
            // V: the vector, point to viewer
            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;
        }

        vec3 illumination = ambient + diffuse + specular;

        //fog
        vec3 fogColor = vec3(0.8, 0.9, 1.0);
        vec2 fogDist = vec2(5.0, 20.0);
        float fogFactor = clamp((fogDist.y - v_Dist)/(fogDist.y - fogDist.x), 0.0, 1.0);

        gl_FragColor = vec4( fogColor * (1.0 - fogFactor) + illumination * fogFactor, 1.0 );
    }
`;
```

Regular illumination code and store the result in the variable “illumination”

Example (Ex11-3)

- shaders in WebGL.js

Define fog color here
(this is the same as the background color we use)

"start" and "end" in the fog factor calculation equation

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

```
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 float u_shininess;
    uniform vec3 u_Color;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying float v_Dist;
    void main(){
        // let ambient and diffuse color are u_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        // 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);
            // V: the vector, point to viewer
            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;
        }

        vec3 illumination = ambient + diffuse + specular;

        //fog
        vec3 fogColor = vec3(0.8, 0.9, 1.0);
        vec2 fogDist = vec2(5.0, 20.0);
        float fogFactor = clamp((fogDist.y - v_Dist)/(fogDist.y - fogDist.x), 0.0, 1.0);

        gl_FragColor = vec4( fogColor * (1.0 - fogFactor) + illumination * fogFactor, 1.0 );
    }
`;
```

Example (Ex11-3)

- shaders in WebGL.js

Calculate the “fogFactor“, and clamp it to 0 ~ 1

mix the fog color and object color

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

```
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 float u_shininess;
    uniform vec3 u_Color;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying float v_Dist;
    void main(){
        // let ambient and diffuse color are u_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        // 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);
            // V: the vector, point to viewer
            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;
        }

        vec3 illumination = ambient + diffuse + specular;

        //fog
        vec3 fogColor = vec3(0.8, 0.9, 1.0);
        vec2 fogDist = vec2(5.0, 20.0);
        float fogFactor = clamp((fogDist.y - v_Dist)/(fogDist.y - fogDist.x), 0.0, 1.0);

        gl_FragColor = vec4( fogColor * (1.0 - fogFactor) + illumination * fogFactor, 1.0 );
    }
`;
```