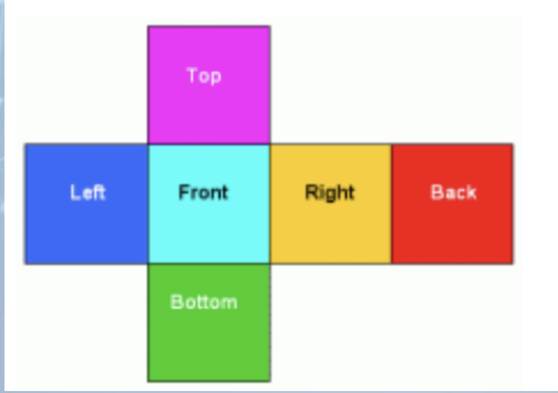
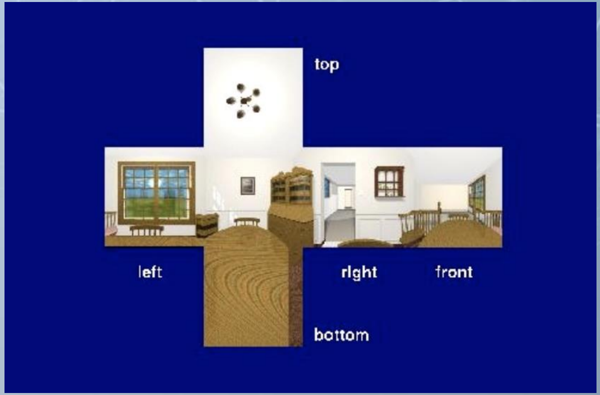
Cubemap

CSU0021: Computer Graphics

Cubic Environment Mapping (Cubemap)

- Introduced by Nate Green 1986
- Place the camera in the center of the environment and project it to 6 sides of a cube
- Cube map consists of six 2D images



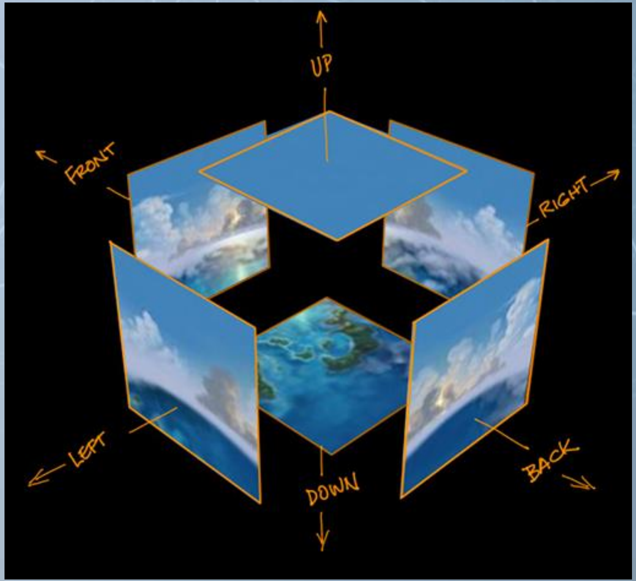


Cubic Environment Mapping (Cubemap)

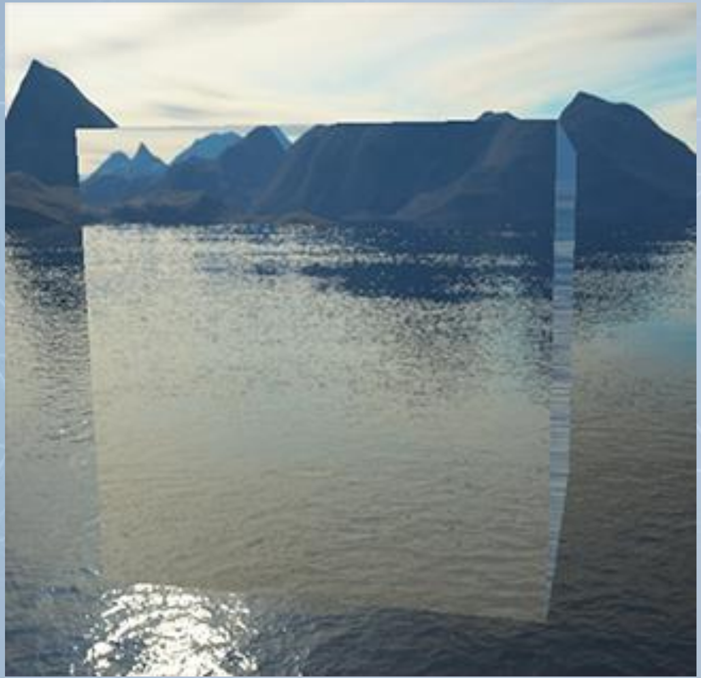
- Applications
  - **Skybox**
  - Environment refraction
  - Environment reflection
  - Dynamic reflection

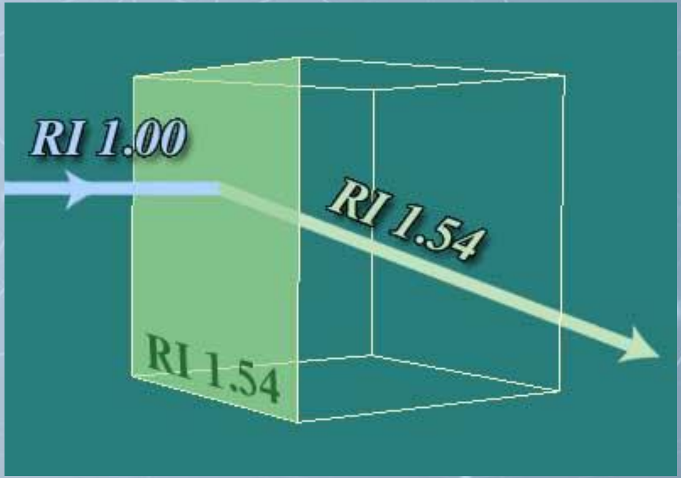
Skybox

- The background comes from cube map images

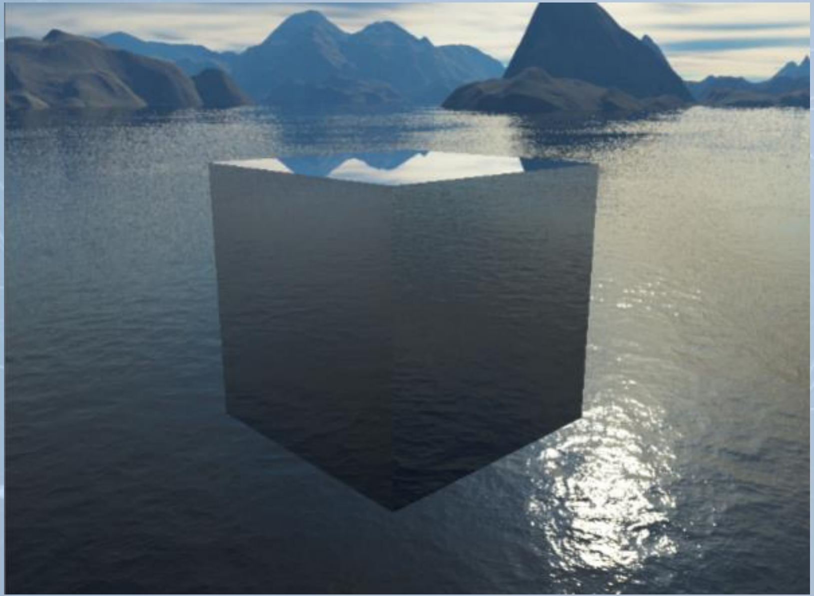


Environment Refraction





Environment Reflection



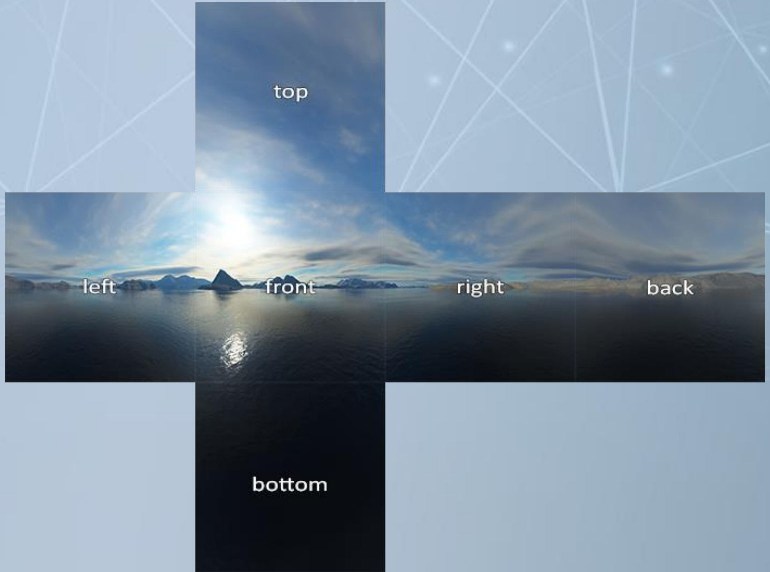
Dynamic Reflection

Skybox

- A large cube that encompass the entire scene
- Contains 6 images of a surrounding environment
- Let users/players the illusion that the environment he is in is much larger than it is
- Examples of skybox in video games: mountains, clouds or sky

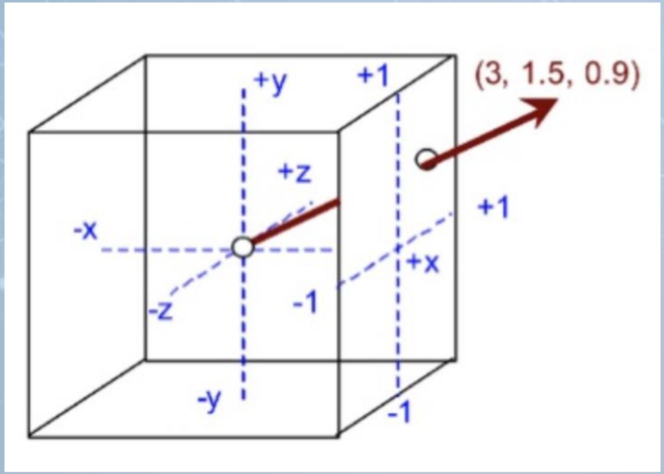
Cubemap

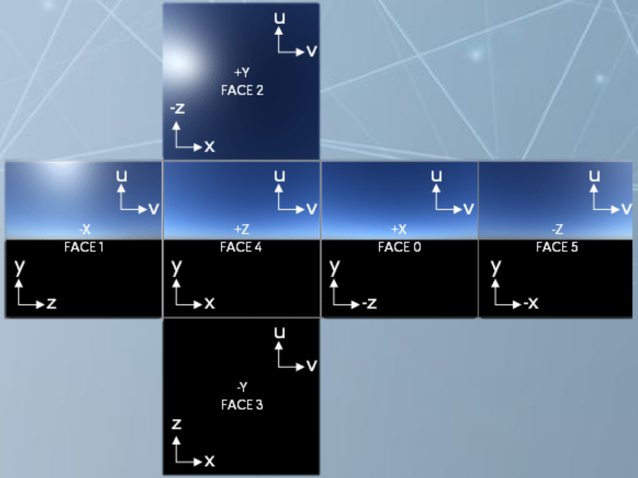
- The skypbox comes from a cubemap
- Cubemap consists of 6 images
- Load the images and use them to render the background



Cubemap

- Cubemap is supported by WebGL
- After the 6 images (cubemap) is loaded, WebGL has a special texture, cubemap texture, to handle it
- You can use a vec3 to indicate a location and access a color on the cubemap
  - Between (+1, +1, +1) and (-1, -1, -1)



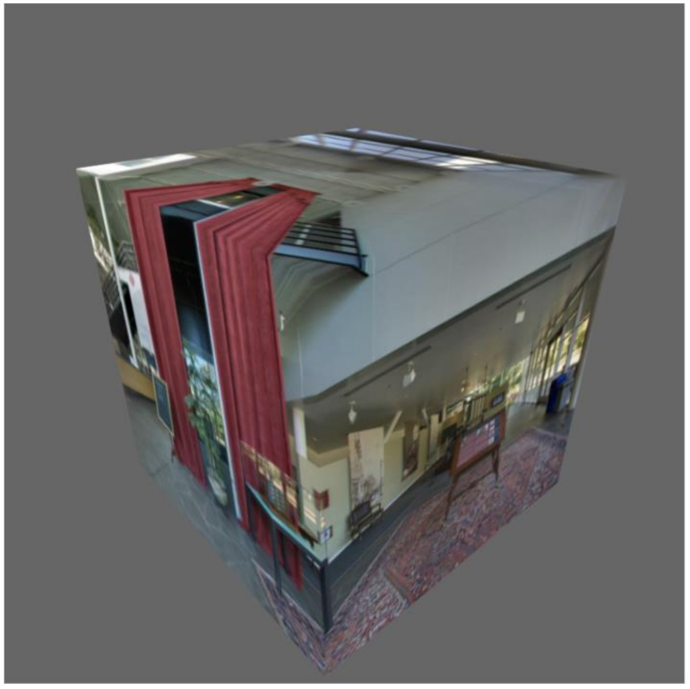


Example (Ex10-1)

- Load a cubemap and use it to color a 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 (Ex10-1)

- initCubeTexture() in WebGL.js
- Very similar to initalize a 2D texture
- Steps
  - Create a texture buffer
  - Bind the buffer to “gl.TEXTURE_CUBE_MAP”
  - Assign a loaded image to the appropriate target (ex: gl.TEXTURE_CUBE_MAP_POSITIVE_Y)
  - Configurate the cubemap
    - gl.texParameteri()

In main()

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

```
function initCubeTexture(posXName, negXName, posYName, negYName,
                         posZName, negZName, imgWidth, imgHeight)
{
  var texture = gl.createTexture();

  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      fName: posXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      fName: negXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      fName: posYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      fName: negYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      fName: posZName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      fName: negZName,
    },
  ];
  faceInfos.forEach((faceInfo) => {
    const {target, fName} = faceInfo;
    // setup each face so it's immediately renderable (avoid error message)
    gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
    gl.texImage2D(target, 0, gl.RGBA, imgWidth, imgHeight, 0,
                  gl.RGBA, gl.UNSIGNED_BYTE, null);

    var image = new Image();
    image.onload = function(){
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    };
    image.src = fName;
  });

  return texture;
}
```

Example (Ex10-1)

- initCubeTexture() in WebGL.js
- Very similar to initalize a 2D texture
- Steps
  - **Create a texture buffer**
  - Bind the buffer to “gl.TEXTURE_CUBE_MAP”
  - Assign a loaded image to the appropriate target (ex: gl.TEXTURE_CUBE_MAP_POSITIVE_Y)
  - Configurate the cubemap
    - gl.texParameteri()

In main()

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

```
function initCubeTexture(posXName, negXName, posYName, negYName,
                         posZName, negZName, imgWidth, imgHeight)
{
  var texture = gl.createTexture();

  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      fName: posXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      fName: negXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      fName: posYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      fName: negYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      fName: posZName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      fName: negZName,
    },
  ];
  faceInfos.forEach((faceInfo) => {
    const {target, fName} = faceInfo;
    // setup each face so it's immediately renderable (avoid error message)
    gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
    gl.texImage2D(target, 0, gl.RGBA, imgWidth, imgHeight, 0,
                  gl.RGBA, gl.UNSIGNED_BYTE, null);

    var image = new Image();
    image.onload = function(){
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    };
    image.src = fName;
  });

  return texture;
}
```

Example (Ex10-1)

- initCubeTexture() in WebGL.js
- Very similar to initalize a 2D texture
- Steps
  - Create a texture buffer
  - **Bind the buffer to "gl.TEXTURE_CUBE_MAP"**
  - Assign a loaded image to the appropriate target (ex: gl.TEXTURE_CUBE_MAP_POSITIVE_Y)
  - Configurate the cubemap
    - gl.texParameteri()

In main()

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

```
function initCubeTexture(posXName, negXName, posYName, negYName,
                         posZName, negZName, imgWidth, imgHeight)
{
  var texture = gl.createTexture();

  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      fName: posXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      fName: negXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      fName: posYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      fName: negYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      fName: posZName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      fName: negZName,
    },
  ];
  faceInfos.forEach((faceInfo) => {
    const {target, fName} = faceInfo;
    // setup each face so it's immediately renderable (avoid error message)
    gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
    gl.texImage2D(target, 0, gl.RGBA, imgWidth, imgHeight, 0,
                  gl.RGBA, gl.UNSIGNED_BYTE, null);

    var image = new Image();
    image.onload = function(){
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    };
    image.src = fName;
  });

  return texture;
}
```

Example (Ex10-1)

- initCubeTexture() in WebGL.js
- Very similar to initalize a 2D texture
- Steps
  - Create a texture buffer
  - Bind the buffer to "gl.TEXTURE_CUBE_MAP"
  - **Assign a loaded image to the appropriate target (ex: gl.TEXTURE_CUBE_MAP_POSITIVE_Y)**
  - Configurate the cubemap
    - gl.texParameteri()

In main()

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

```
function initCubeTexture(posXName, negXName, posYName, negYName,
                         posZName, negZName, imgWidth, imgHeight)
{
  var texture = gl.createTexture();

  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      fName: posXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      fName: negXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      fName: posYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      fName: negYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      fName: posZName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      fName: negZName,
    },
  ];
  faceInfos.forEach((faceInfo) => {
    const {target, fName} = faceInfo;
    // setup each face so it's immediately renderable (avoid error message)
    gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
    gl.texImage2D(target, 0, gl.RGBA, imgWidth, imgHeight, 0,
                  gl.RGBA, gl.UNSIGNED_BYTE, null);

    var image = new Image();
    image.onload = function(){
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    };
    image.src = fName;
  });

  return texture;
}
```

Example (Ex10-1)

- initCubeTexture() in WebGL.js
- Very similar to initalize a 2D texture
- Steps
  - Create a texture buffer
  - Bind the buffer to "gl.TEXTURE_CUBE_MAP"
  - Assign a loaded image to the appropriate target (ex: gl.TEXTURE_CUBE_MAP_POSITIVE_Y)
  - **Configurate the cubemap**
    - **gl.texParameteri()**

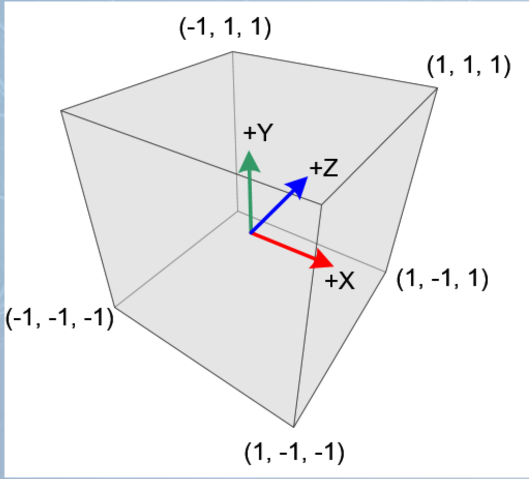
In main()

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

```
function initCubeTexture(posXName, negXName, posYName, negYName,
                         posZName, negZName, imgWidth, imgHeight)
{
  var texture = gl.createTexture();

  const faceInfos = [
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_X,
      fName: posXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
      fName: negXName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
      fName: posYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
      fName: negYName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
      fName: posZName,
    },
    {
      target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
      fName: negZName,
    },
  ];
  faceInfos.forEach((faceInfo) => {
    const {target, fName} = faceInfo;
    // setup each face so it's immediately renderable (avoid error message)
    gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
    gl.texImage2D(target, 0, gl.RGBA, imgWidth, imgHeight, 0,
                  gl.RGBA, gl.UNSIGNED_BYTE, null);

    var image = new Image();
    image.onload = function(){
      gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
      gl.texImage2D(target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    };
    image.src = fName;
  });

  return texture;
}
```

Example (Ex10-1)

- main() in WebGL.js
- Load a cube from obj




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

    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);
      cubeObj.push(o);
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_envCubeMap = gl.getUniformLocation(program, 'u_envCubeMap');

    gl.useProgram(program);

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

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

Example (Ex10-1)

- Pass the cube map texture to shader
- Similar to passing 2D texture
  - But the target is "gl.TEXTURE_CUBE_MAP"

```
function draw(){
  gl.useProgram(program);
  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);

  var mvpFromCamera = new Matrix4();
  // //model Matrix (part of the mvp matrix)
  let modelMatrix = new Matrix4();
  modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 2.0, 2.0);
  modelMatrix.multiply(cubeMdlMatrix);
  // //mvp: projection * view * model matrix
  mvpFromCamera.setPerspective(60, 1, 1, 15);
  mvpFromCamera.lookAt(cameraX, cameraY, cameraZ,   0, 0, 0, 0, 1, 0);
  mvpFromCamera.multiply(modelMatrix);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMapTex);
  gl.uniform1i(program.u_envCubeMap, 0);

  //cube
  gl.useProgram(program);
  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);
  for( let i=0; i < cubeObj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, cubeObj[i].vertexBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, cubeObj[i].numVertices);
  }
}
```

Example (Ex10-1)

- Shader in WebGL.js

- Variable type to represent the cubemap texture: “samplerCube”

- Access color from the cube map: textureCube()

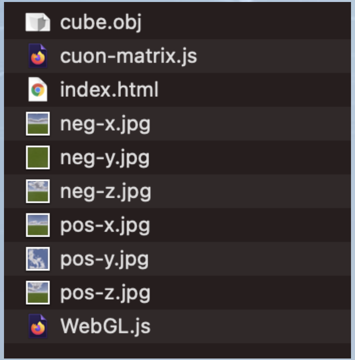
```
var VSHADER_SOURCE = `
  attribute vec4 a_Position;
  varying vec4 v_TexCoord;
  uniform mat4 u_MvpMatrix;
  void main() {
    gl_Position = u_MvpMatrix * a_Position;
    v_TexCoord = a_Position;
  }
`;

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

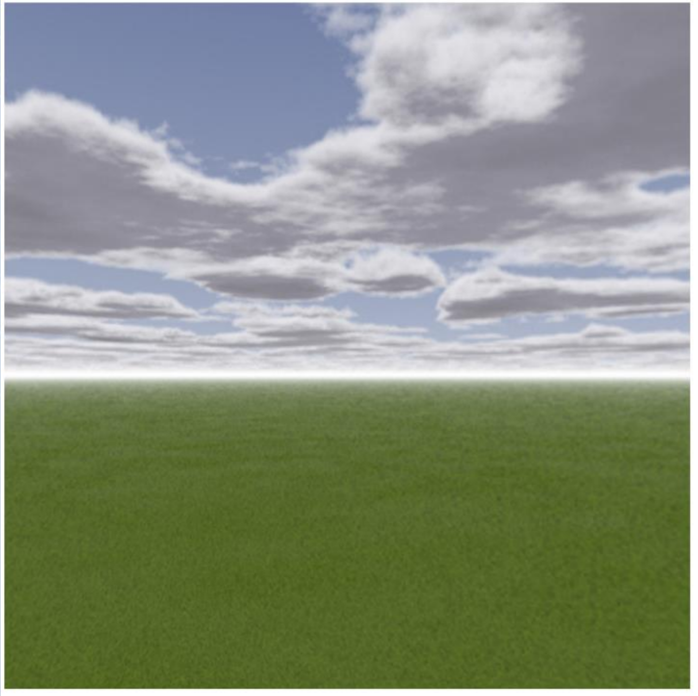
Try and Think (5mins)

- Download the code and run
- Make sure you know how to load cube map image, pass the cube map texture to shader and access color from the cube map texture
- If you move the camera into the cube, what you see?
  - Set cameraX, cameraY, cameraZ to 0, 0 and 0.001, respectively

Example (Ex10-2)

- Instead of using model matrix to rotate a object, we want move the camera (player) and rotate the view of the camera in the scene
  - Move the camera position if users press ‘w’ or ‘s’ (move forward or backward)
  - change the view direction if users use mouse to drag and move
- Files





Example (Ex10-2)

- I have one more variable set for view direction

```
var mouseLastX, mouseLastY;
var mouseDragging = false;
var angleX = 0, angleY = 0;
var gl, canvas;
var modelMatrix;
var nVertex;
var cameraX = 0, cameraY = 0, cameraZ = 0;
var cameraDirX = 0, cameraDirY = 0, cameraDirZ = 1;
var cubeObj = [];
var cubeMapTex;
```

Example (Ex10-2)

- draw() in WebGL.js

- **Now, we rotate the view direction by the information stored in “angleX” and “angleY”**

- Then, use the camera position and the look-at point (camera position + view direction) to set the view matrix

```
function draw(){
  gl.useProgram(program);
  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);

  var mvpFromCamera = new Matrix4();
  // //model Matrix (part of the mvp matrix)
  let modelMatrix = new Matrix4();
  modelMatrix.setScale(2.0, 2.0, 2.0);
  // //mvp: projection * view * model matrix
  mvpFromCamera.setPerspective(60, 1, 1, 15);
  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);
  mvpFromCamera.lookAt(cameraX, cameraY, cameraZ,
                       cameraX + newViewDir.elements[0],
                       cameraY + newViewDir.elements[1],
                       cameraZ + newViewDir.elements[2],
                       0, 1, 0);
  mvpFromCamera.multiply(modelMatrix);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMapTex);
  gl.uniform1i(program.u_envCubeMap, 0);

  //cube
  gl.useProgram(program);
  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);
  for( let i=0; i < cubeObj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, cubeObj[i].vertexBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, cubeObj[i].numVertices);
  }
}
```

Example (Ex10-2)

- draw() in WebGL.js
- Now, we rotate the view direction by the information stored in “angleX” and “angleY”
- **Then, use the camera position and the look-at point (camera position + view direction) to set the view matrix**

```
function draw(){
  gl.useProgram(program);
  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);

  var mvpFromCamera = new Matrix4();
  // //model Matrix (part of the mvp matrix)
  let modelMatrix = new Matrix4();
  modelMatrix.setScale(2.0, 2.0, 2.0);
  // //mvp: projection * view * model matrix
  mvpFromCamera.setPerspective(60, 1, 1, 15);
  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);
  mvpFromCamera.lookAt(cameraX, cameraY, cameraZ,
                        cameraX + newViewDir.elements[0],
                        cameraY + newViewDir.elements[1],
                        cameraZ + newViewDir.elements[2],
                        0, 1, 0);
  mvpFromCamera.multiply(modelMatrix);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, cubeMapTex);
  gl.uniform1i(program.u_envCubeMap, 0);

  //cube
  gl.useProgram(program);
  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);
  for( let i=0; i < cubeObj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, cubeObj[i].vertexBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, cubeObj[i].numVertices);
  }
}
```

Example (Ex10-2)

- keydown() in WebGL.js
- Add a keyboard event to move the camera position
- ‘w’: move forward along the view direction
- ‘s’: move backward along the view direction

```
function keydown(ev){
  //implment keydown event here
  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);

  if(ev.key == 'w'){
      cameraX += (newViewDir.elements[0] * 0.1);
      cameraY += (newViewDir.elements[1] * 0.1);
      cameraZ += (newViewDir.elements[2] * 0.1);
  }
  else if(ev.key == 's'){
    cameraX -= (newViewDir.elements[0] * 0.1);
    cameraY -= (newViewDir.elements[1] * 0.1);
    cameraZ -= (newViewDir.elements[2] * 0.1);
  }
  draw();
}
```

Try and Think (5mins)

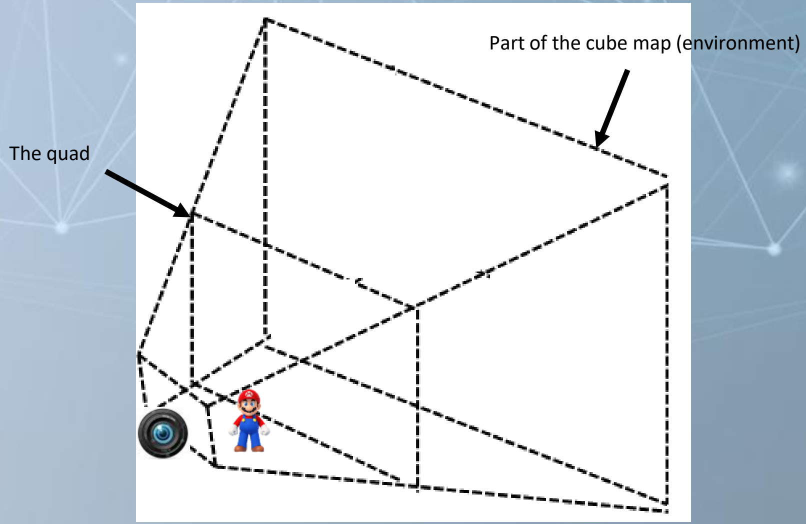
- Download and run the code
- Keep moving, do you go out the skybox?
- We usually want to keep staying in the skybox, how to solve this problem?

Example (Ex10-3)

- We should never go out of the skybox
  - The environment cube supposes to stay at infinite distance
- Instead of coloring an “real cube”, draw a quad which covers the whole camera view
  - Put the quad behind all the objects you want to render
  - Project the cube map to the quad
    - Only the rotation of the view has impact on this projection (do not consider translation)

- 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 (Ex10-3)

- In main() WebGL.js
- We define a quad and the coordinate here is in "clip space"
  - We will NOT transform it in vertex shader
  - Z of this quad is 1:
    - we want the quad always behind all objects
    - Any object with z grater than 1 will be clipped out

```
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');

    quadObj = initVertexBufferForLaterUse(gl, quad);

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

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

Example (Ex10-3)

- Shaders in WebGL.js
- The quad vertices is already in clip space, so we direct assign it to gl_Position
- We also pass the quad vertices coordinates to fragment shader

```
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 (Ex10-3)

- Shaders in WebGL.js
- To project the environment cube map to the quad, we should look up the color from the cube map to color the quad
- This look-up behavior depends on which direction the user is looking at (about the view matrix)
- We only know the coordinates on the quad (v_Position) and the coordinate is defined in the clip space
- To look up the color from the cube map, we need the direction vector defined in world space (of course, environment is in the world space)
- We know:
  - $C_{clip} = M_{proj} * M_{view} * C_{world}$
  - $\left(M_{proj} * M_{view}\right)^{-1} * C_{clip} = C_{world}$

```
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 (Ex10-3)

- draw() in WebGL.js
- Prepare $\left(M_{proj} * M_{view}\right)^{-1}$ and pass it into the shader

```
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);

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

  //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);
}
```

Example (Ex10-3)

- draw() in WebGL.js
- Although the player (camera) can move, we want to ignore the camera translation when draw the environment
  - We assume the environment cube is at the infinite distance.
  - The local camera move (translation) will not affect what the camera can see about the environment
- We ignore the view translation by setting m_{12}, m_{13}, m_{14} ($\begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$)to 0
  - Because m_{12}, m_{13}, m_{14} in view matrix are responsible for camera translation

```
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);

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

  //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);
}
```

Example (Ex10-3)

- draw() in WebGL.js
- gl.depthFunc()
  - Set what z value is possible to pass the depth buffer (in depth test)
  - By default, it is gl.depthFunc(gl.LESS), the value is smaller than current buffer value is possible to pass
  - We set the z of the quad at 1 and we want it is possible to pass the depth test. (the initial depth buffer value is 1)
    - gl.depthFunc(gl.LEQUAL): less ang equal
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/depthFunc

```
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);

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

  //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 and Think (5mins)

- Download and run Ex10-3
- It looks like you never move, when you press “w” and “s”. However, you move.
- I use console.log() to output the camera position (x, y, z in world space) in console when you press “w” or “s”. You can check.
- Think why you don’t see any change even if the camera move?
- And, is this what we want?
  - The answer is YES.
  - The environment you can see should not change if you only move forward or backward (no camera rotation) because the environment background suppose to stay at infinite distance
  - If you add objects around you, you will realize you actually move