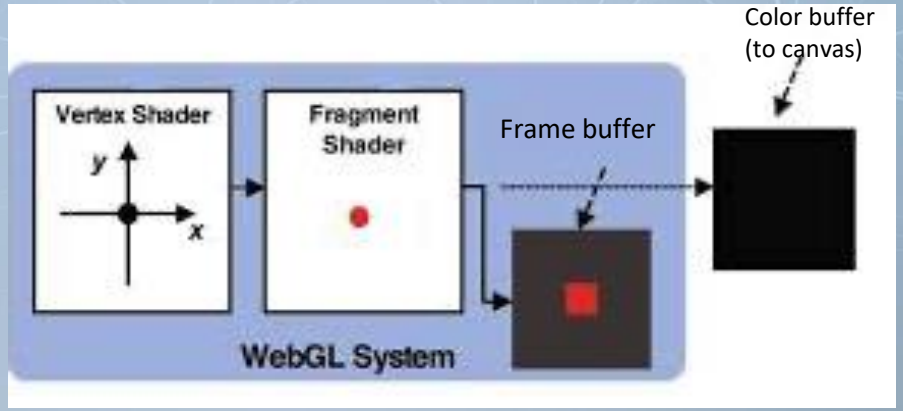
Frame Buffer & Shadow

CSU0021: Computer Graphics

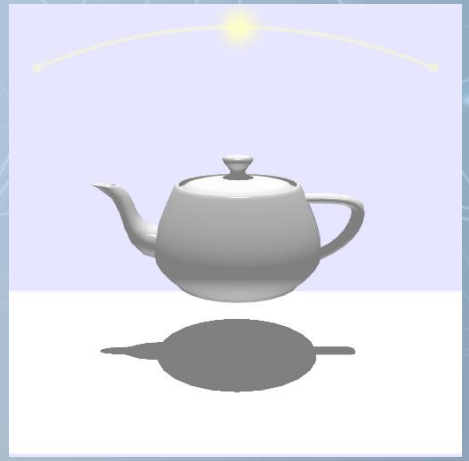
Frame Buffer

- Replace color buffer or depth buffer
  - By default, WebGL draws on color buffer
- We can also draw on **frame buffer**
  - Offscreen rendering. Why?



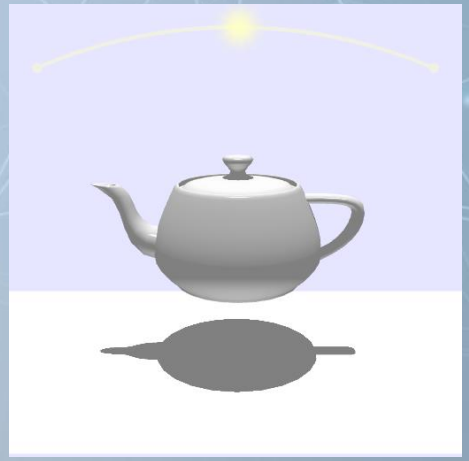
Applications of Offscreen Rendering

- 3D rendering in another 3D rendering
- Shadow …

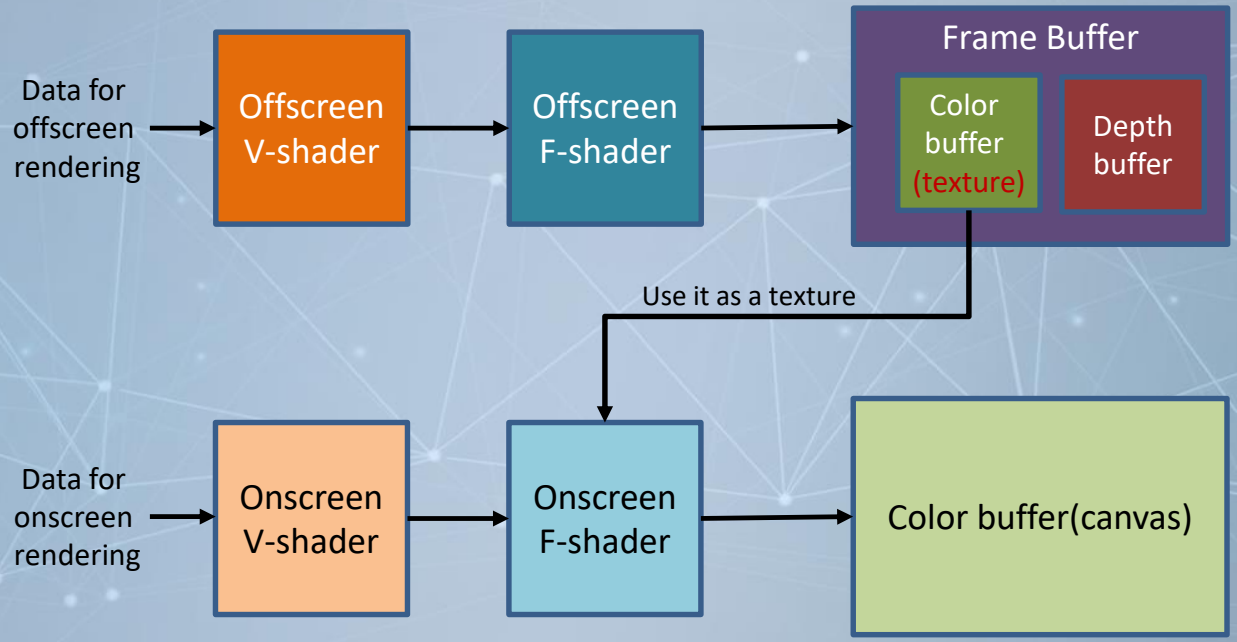


Applications of Offscreen Rendering

- At least two pass rendering
  - The first pass: offscreen rendering to a frame buffer
  - The second pass: use the information in the frame buffer to render the final image and show it on canvas

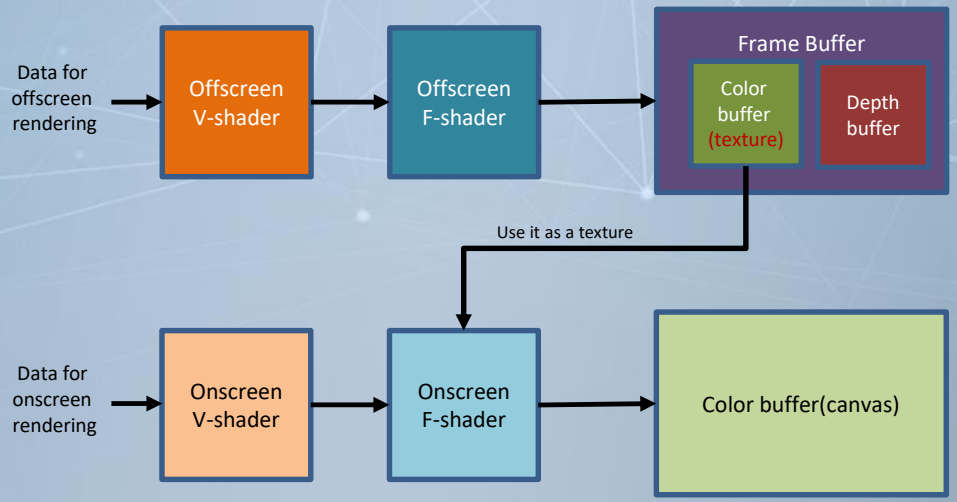


Big Picture of Use of Frame Buffer



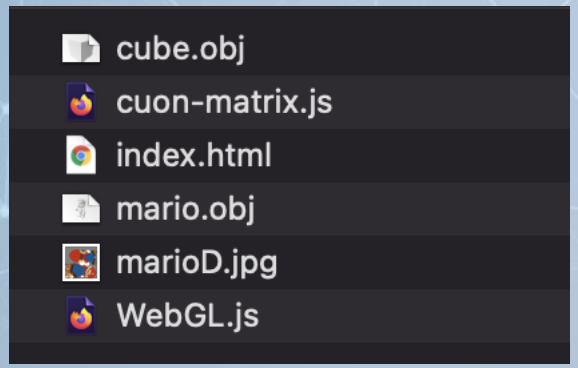
Steps of Use of Frame Buffer

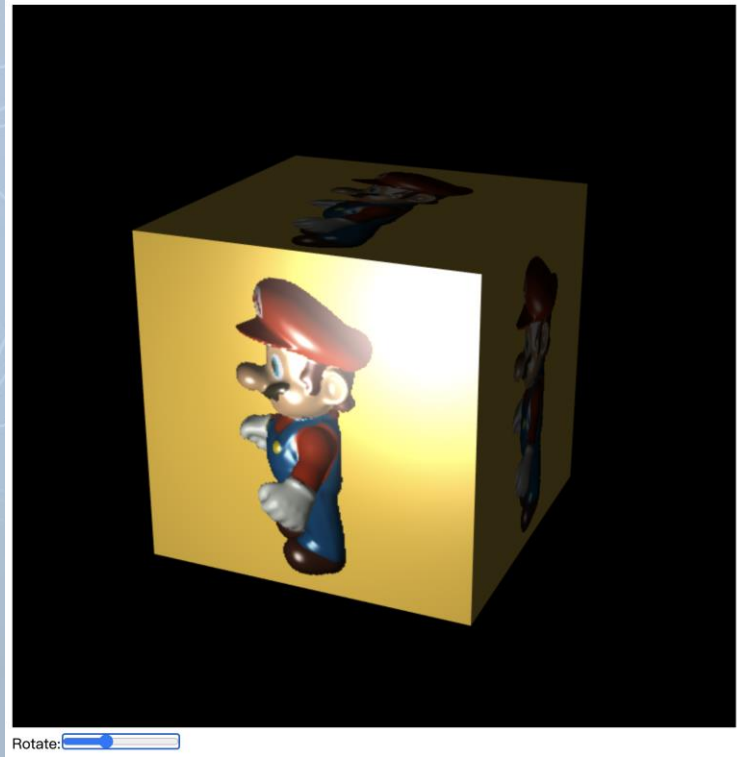
- Create and setup a frame buffer
- Switch the destination of rendering to the created frame buffer
- Call off-screen shader to draw to the frame buffer
- Switch the destination of the rendering back to the default buffer
- Pass the texture of the created frame buffer to the on-screen shader
- Call the on-screen shader to draw an image frame



Example (Ex09-1)

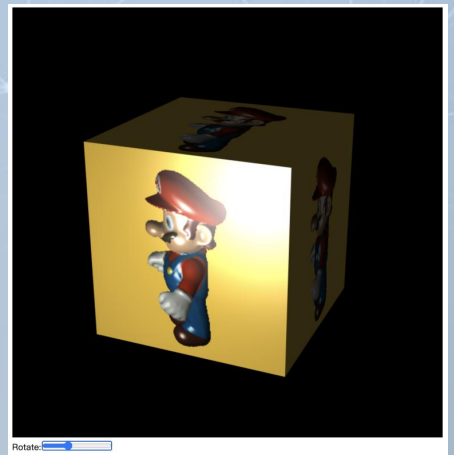
- When users drag the slider, the mario on the cube rotates (the cube does not move)
- Files:





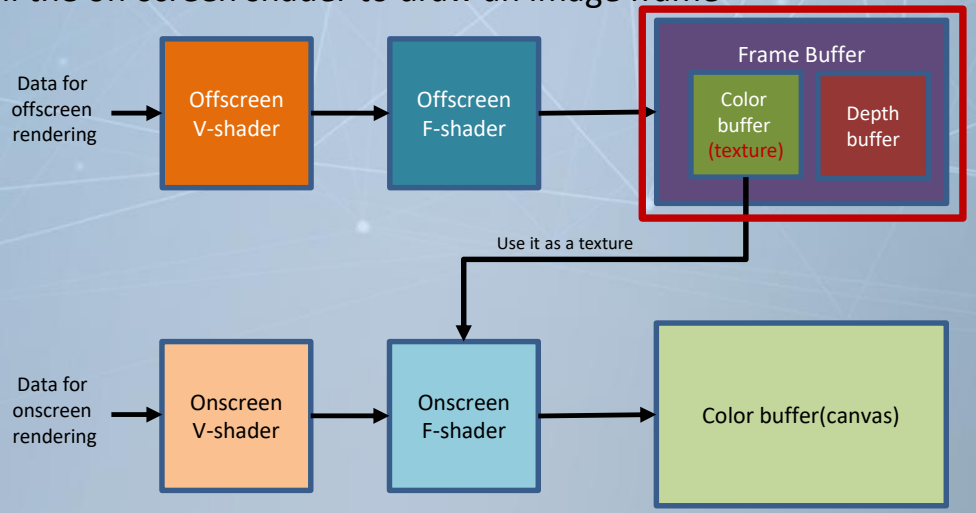
Example (Ex09-1)

- Idea of this example
  - Pass the mario model to an off-screen shader and render
  - The result of the off-screen rendering is stored in a texture
  - Pass the texture and the cube to an on-screen shader
  - Render the cube and map the texture to the cube faces



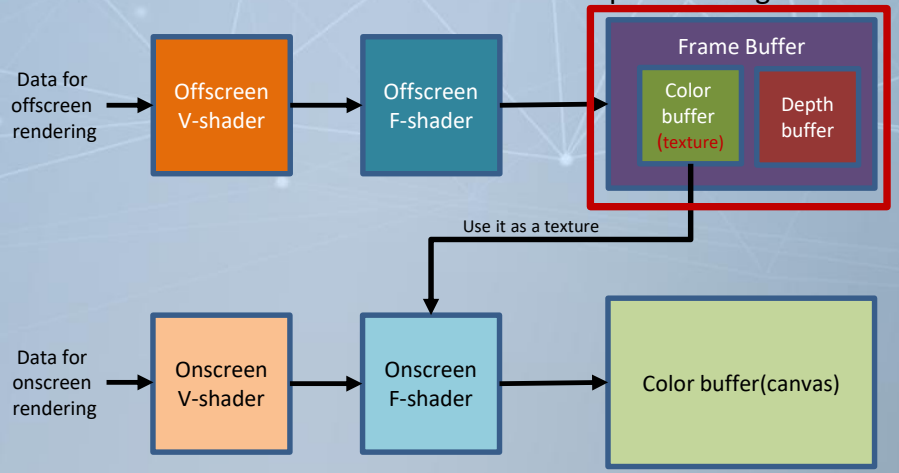
Example (Ex09-1): Steps of Use of Frame Buffer

- **Create and setup a frame buffer**
- Switch the destination of rendering to the created frame buffer
- Call off screen shader to draw to the frame buffer
- Switch the destination of the rendering back to the default color buffer
- Pass the texture of the created frame buffer to the on-screen shader
- Call the on-screen shader to draw an image frame



Example (Ex09-1): Steps of Frame Buffer Creation

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer: multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()



Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- create a frame buffer: gl.createFramebuffer()
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/createFramebuffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - **Create a frame buffer: gl.createFramebuffer()**
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- create a texture buffer as the color buffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);      size to render (2048, 2048)
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - **Create a texture buffer as a color buffer:**
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- Create a render buffer as a depth buffer
  - gl.createRenderbuffer()
- https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/createRenderbuffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - **Create a render buffer as a depth buffer: gl.createRenderbuffer()**
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- Configurate the render buffer
  - gl.bindRenderbuffer()
    - Bind the buffer we just created to gl.RENDERBUFFER
    - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/bindRenderbuffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - **Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()**
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- Configurate the render buffer
  - gl.renderbufferStorage(target, internalformat, width, height)
    - target: should be gl.RENDERBUFFER
    - internalformat: functionality and size per unit of this buffer
    - Width, height: width and height of this buffer
    - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/renderbufferStorage

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

size (2048, 2048)

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - **Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()**
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- Before starting to setup the frame buffer, we should bind the created from buffer to gl.FRAMEBUFFER using gl.bindFramebuffer()
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/bindFramebuffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - **Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()**
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- set the texture to be color buffer of the frame buffer
- gl.framebufferTexture2D(target, attachment, textarget, texture, level)
  - target: gl.FRAMEBUFFER
  - attachment: gl.COLOR_ATTACHMENT0 means linking to color buffer
  - textarget: gl.TEXTURE_2D or gl.TEXTURE_CUBE
  - texture: the texture
  - level: set 0
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/framebufferTexture2D

```
fbo = initFrameBuffer(gl);
```
in main()

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - **Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()**
  - Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- Set the render buffer as the depth buffer of the frame buffer
- gl.framebufferRenderbuffer(target, attachment, renderbuffertarget, renderbuffer)
  - target: gl.FRAMEBUFFER in this case
  - attachment: gl.DEPTH_ATTACHMENT to set this buffer as the depth buffer
  - renderbuffertarget: gl.RENDERBUFFER in this case
  - renderbuffer: the buffer
  - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/framebufferRenderbuffer

```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - **Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer**

Example (Ex09-1): Steps of Frame Buffer Creation

- initFrameBufffer() in WebGL.js
- framebuffer.texture = texture
  - It is easy for us to access the texture if we put the reference of the texture and the framebuffer together
- return frameBuffer

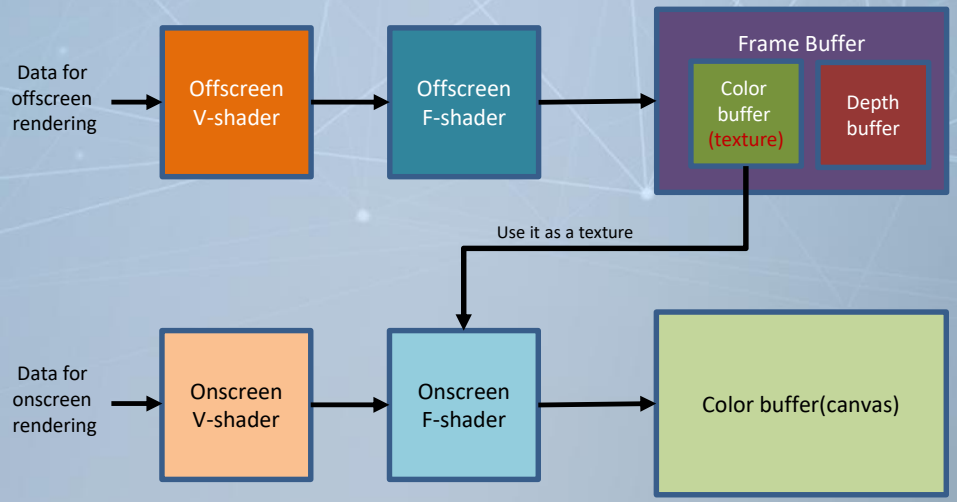
```
fbo = initFrameBuffer(gl);
```
in main()

```
function initFrameBuffer(gl){
  //create and set up a texture object as the color buffer
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, offScreenWidth, offScreenHeigh
                0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);


  //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;
  return frameBuffer;
}
```

- Create and setup a frame buffer: (5 steps)
  - Create a frame buffer: gl.createFramebuffer()
  - Create a texture buffer as a color buffer:
    multiple steps. The same as how we create a texture buffer before
  - Create a render buffer as a depth buffer: gl.createRenderbuffer()
  - Configurate the render buffer: gl.bindRenderbuffer(), gl.renderbufferStorage()
  - Bind the texture buffer to the frame buffer as a color buffer: gl.framebufferTexture2D()
  - **Bind the render buffer to the frame buffer as a depth buffer: gl.framebufferRenderbuffer()**

Example (Ex09-1): Switch Rendering Target to New Frame Buffer

- Create and setup a frame buffer
- **Switch the destination of rendering to the created frame buffer**
- Call off screen shader to draw to the frame buffer
- Switch the destination of the rendering back to the default color buffer
- Pass the texture of the created frame buffer to the on-screen shader
- Call the on-screen shader to draw an image frame



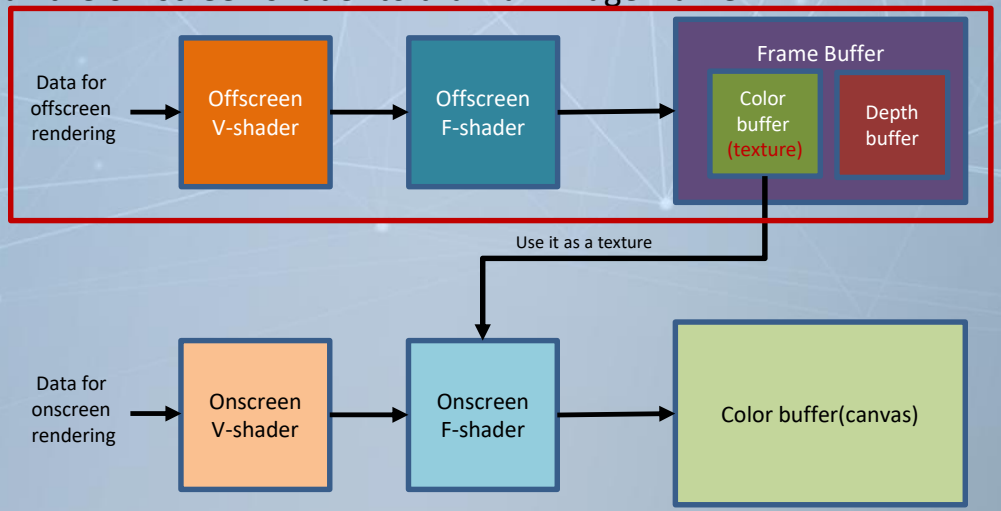
Example (Ex09-1): Switch Rendering Target to New Frame Buffer

- draw() in WebGL.js
- Set the viewport before drawing if your off-screen and on-screen size are different

```
function draw(){
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  drawOffScreen();
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  gl.viewport(0, 0, canvas.width, canvas.height);
  drawOnScreen();
}
```

Example (Ex09-1): Call Off Screen Shader

- Create and setup a frame buffer
- Switch the destination of rendering to the created frame buffer
- **Call off-screen shader to draw to the frame buffer**
- Switch the destination of the rendering back to the default color buffer
- Pass the texture of the created frame buffer to the on-screen shader
- Call the on-screen shader to draw an image frame



Example (Ex09-1): Switch Rendering Target to New Frame Buffer

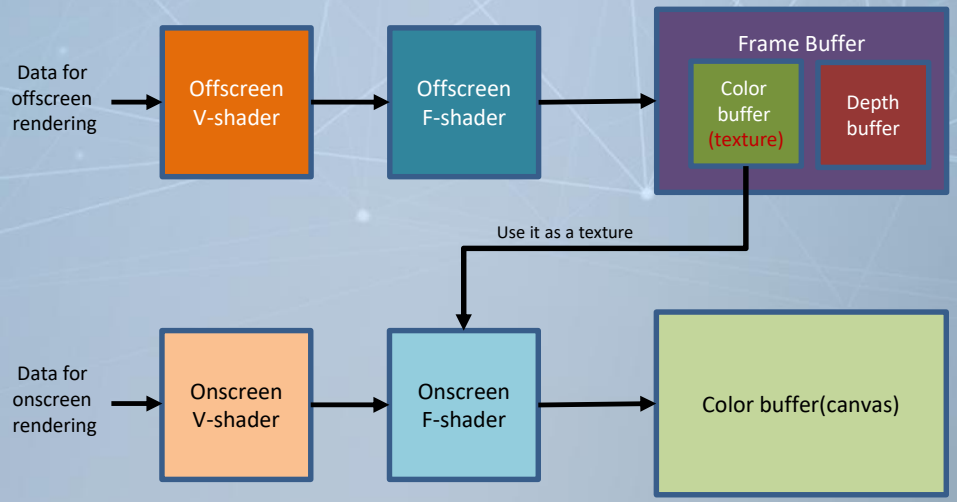
- drawOffScreen() in WebGL.js
- drawOffScreen() is essentially the same as normal draw procedure

```
function draw(){
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  drawOffScreen();
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  gl.viewport(0, 0, canvas.width, canvas.height);
  drawOnScreen();
}
```

```
function drawOffScreen(){
  gl.clearColor(1.0, 0.8, 0.2, 1.0);

  //model Matrix (part of the mvp matrix)
  modelMatrix.setRotate(rotateAngle, 0, 1, 0);
  modelMatrix.scale(objScale, objScale, objScale);
  //mvp: projection * view * model matrix
  mvpMatrix.setPerspective(30, 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.uniform1i(program.u_Sampler0, 0);

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

  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textures["marioTex"]);

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

Example (Ex09-1): Switch Back to Normal Rendering Target

- Create and setup a frame buffer
- Switch the destination of rendering to the created frame buffer
- Call off screen shader to draw to the frame buffer
- **Switch the destination of the rendering back to the default color buffer**
- Pass the texture of the created frame buffer to the on-screen shader
- Call the on-screen shader to draw an image frame



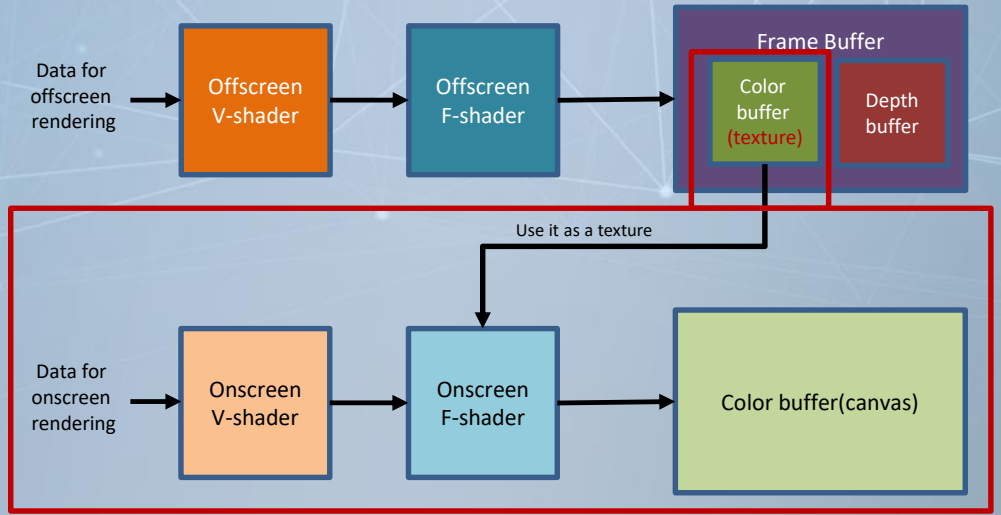
Example (Ex09-1): Switch Rendering Target to New Frame Buffer

- draw() in WebGL.js
- To switch the rendering target back to normal(default) buffer, just call gl.bindFramebuffer() and set the second parameter to **null**

```
function draw(){
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  drawOffScreen();
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  gl.viewport(0, 0, canvas.width, canvas.height);
  drawOnScreen();
}
```

Example (Ex09-1): Switch Back to Normal Rendering Target

- Create and setup a frame buffer
- Switch the destination of rendering to the created frame buffer
- Call off screen shader to draw to the frame buffer
- Switch the destination of the rendering back to the default color buffer
- **Pass the texture of the created frame buffer to the on-screen shader**
- **Call the on-screen shader to draw an image frame**



Example (Ex09-1): Pass the Texture from Off-screen Shader to On-Screen Shader and Render

- drawOnScreen() in WebGL.js
- Just get the texture object in our “fbo” and assign to a texture unit
- Call the on-screen shader to render

```
function draw(){
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  drawOffScreen();
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  gl.viewport(0, 0, canvas.width, canvas.height);
  drawOnScreen();
}
```

```
function drawOnScreen(){
    gl.clearColor(0,0,0,1);

    //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.scale(1.0, 1.0, 1.0);
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 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.uniform1i(program.u_Sampler0, 0);

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

    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, fbo.texture);

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

Example (Ex09-1): Shader

- Shader?
  - The on- and off-screen are the same one in this example.
  - And, they are the same as the shader we have to render a scene with a texture image

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Normal;
    attribute vec2 a_TexCoord;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_Normal = normalize(vec3(u_normalMatrix * a_Normal));
        v_TexCoord = a_TexCoord;
    }
`;
```

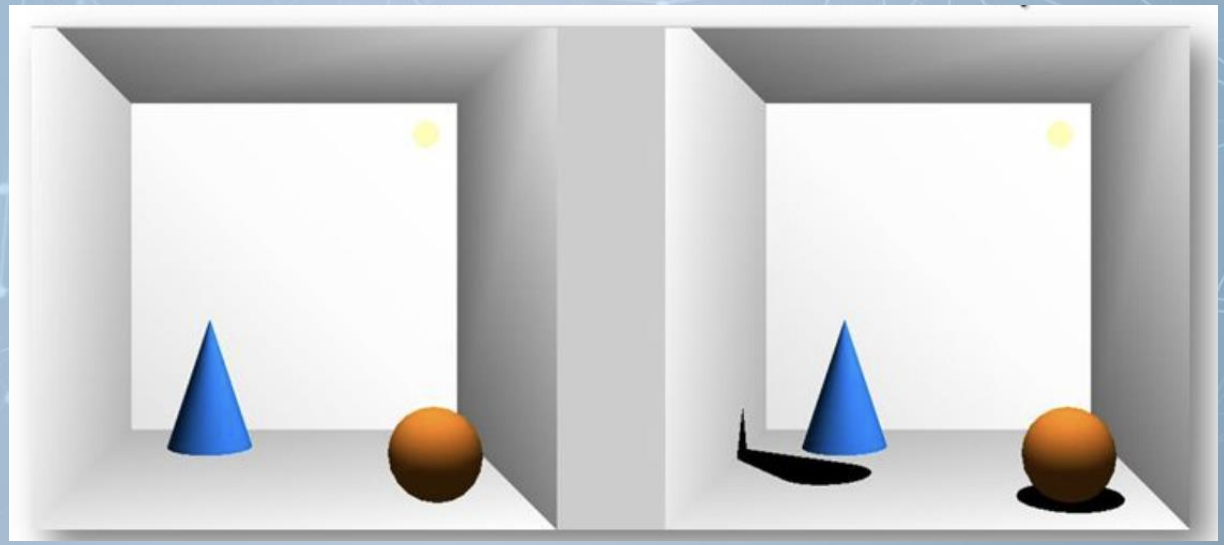
```
var FSHADER_SOURCE = `
    precision mediump float;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform float u_shininess;
    uniform sampler2D u_Sampler0;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    void main(){
        // let ambient and diffuse color are u_Color
        // (you can also input them from ouside and make them different)
        vec3 texColor = texture2D( u_Sampler0, v_TexCoord ).rgb;
        vec3 ambientLightColor = texColor;
        vec3 diffuseLightColor = texColor;
        // 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;
        }

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

Try and Think (5mins)

- The use of the frame buffer is quite complicated.
- Run the code.
- Read the code and make sure you somewhat know what it is going on.

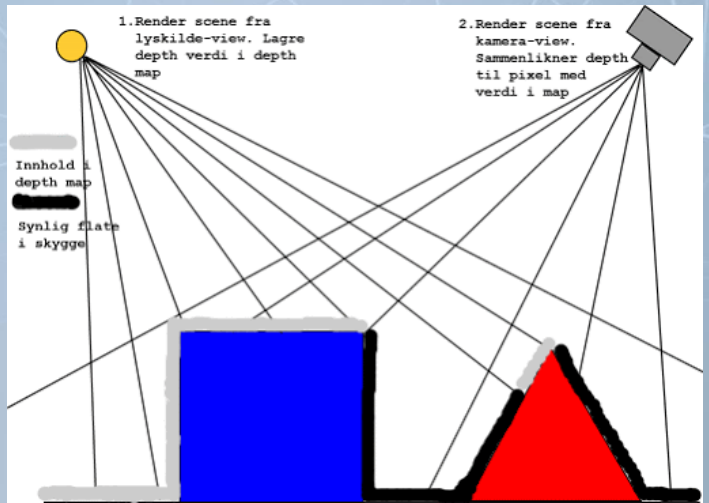
Shadow

- Shadow gives users more clues about the relative position of objects



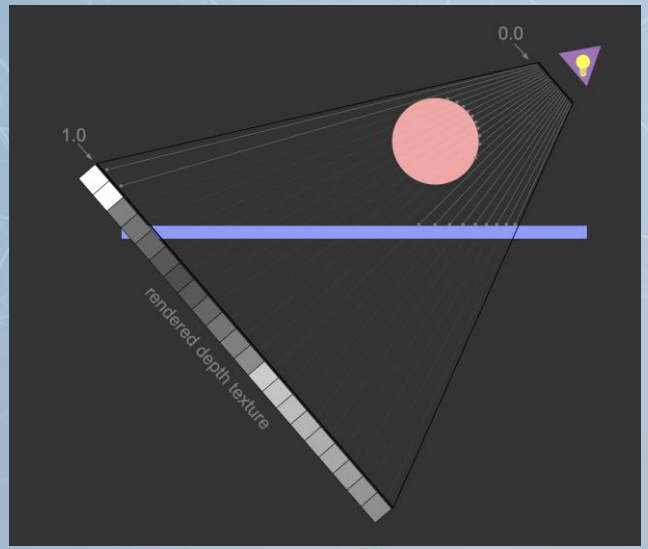
Idea of Shadow Rendering

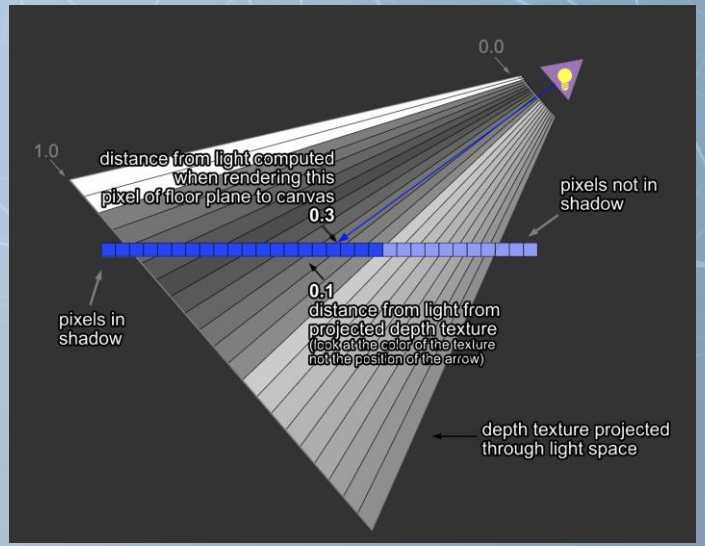
- Sun/light cannot see the shadow
  - Before determining a fragment color in fragment shader check whether this one is the closest object point to the light in the scene



2 Major Steps of Shadow Rendering

- We have to really write two pair of shaders now
- Off-screen rendering (to a frame buffer) first
  - Set the camera position at the light position
  - When rendering, put the depth to gl_FragColor instead of color information
- On-screen rendering
  - Pass the depth information (texture of the frame buffer) produced by the off-screen rendering to on-screen rendering shader
  - The fragment shader of the on-screen rendering can look up the depth information to determine a fragment is under the shadow or not

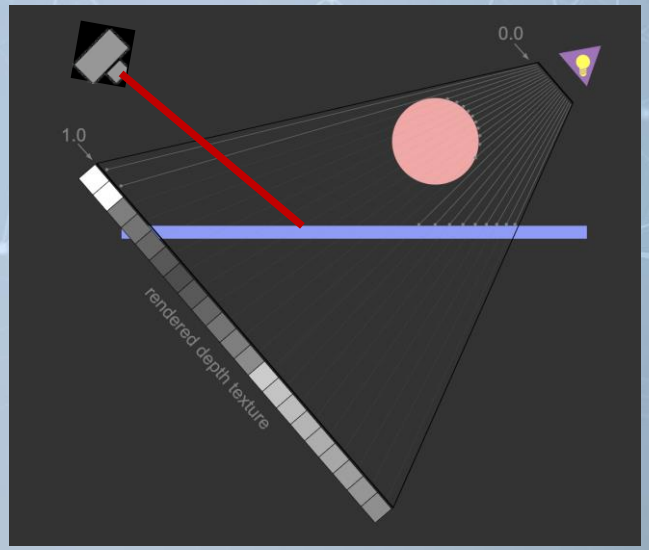




https://webglfundamentals.org/webgl/lessons/webgl-shadows.html

Space Transformation between View of the Camera and Light

- Major question: if you are going to rendering a fragment in the fragment shader of on-screen rendering, which texture coordinate you should use to access the texture in the frame buffer to get the correct depth information



Space Transformation between the On- and Off-Screen Rendering

- Major question: if you are going to rendering a fragment in the fragment shader of on-screen rendering, which texture coordinate you should use to access the texture in the frame buffer to get the correct depth information

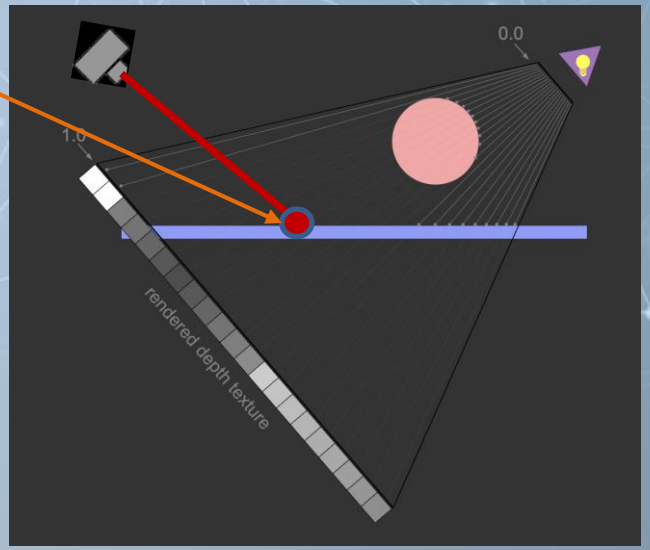
On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility



Space Transformation between the On- and Off-Screen Rendering

- Major question: if you are going to rendering a fragment in the fragment shader of on-screen rendering, which texture coordinate you should use to access the texture in the frame buffer to get the correct depth information

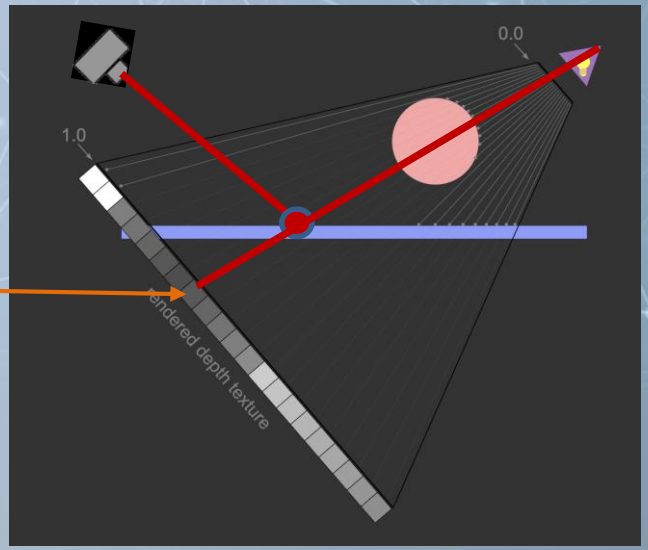
On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility



Space Transformation between the On- and Off-Screen Rendering

- Major question: if you are going to rendering a fragment in the fragment shader of on-screen rendering, which texture coordinate you should use to access the texture in the frame buffer to get the correct depth information

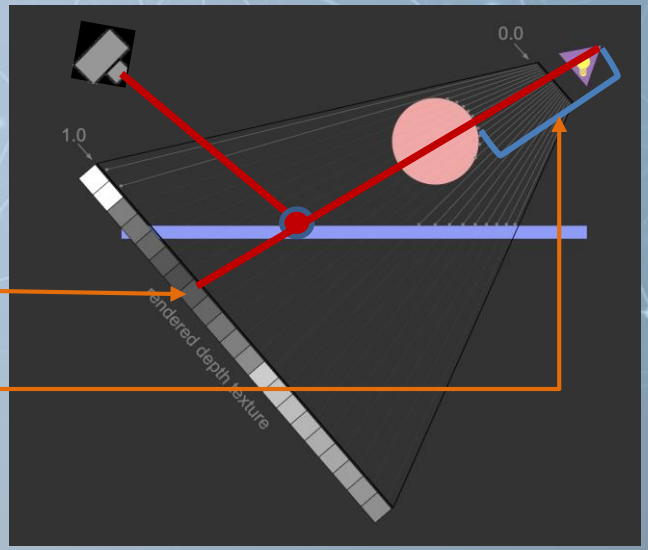
On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility



Space Transformation between the On- and Off-Screen Rendering

- Major question: if you are going to rendering a fragment in the fragment shader of on-screen rendering, which texture coordinate you should use to access the texture in the frame buffer to get the correct depth information

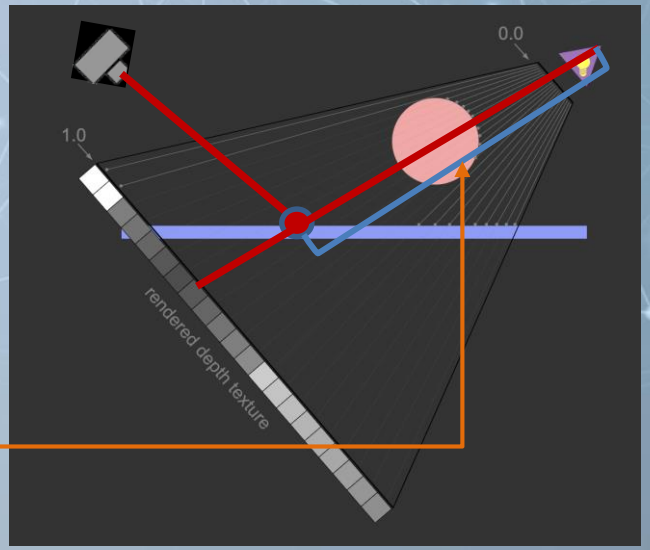
On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

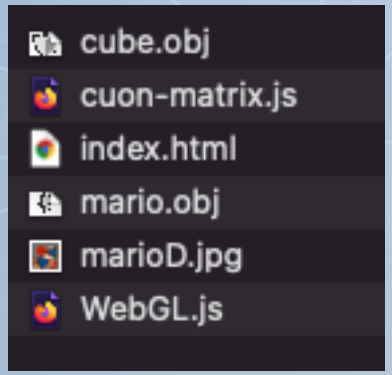
4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

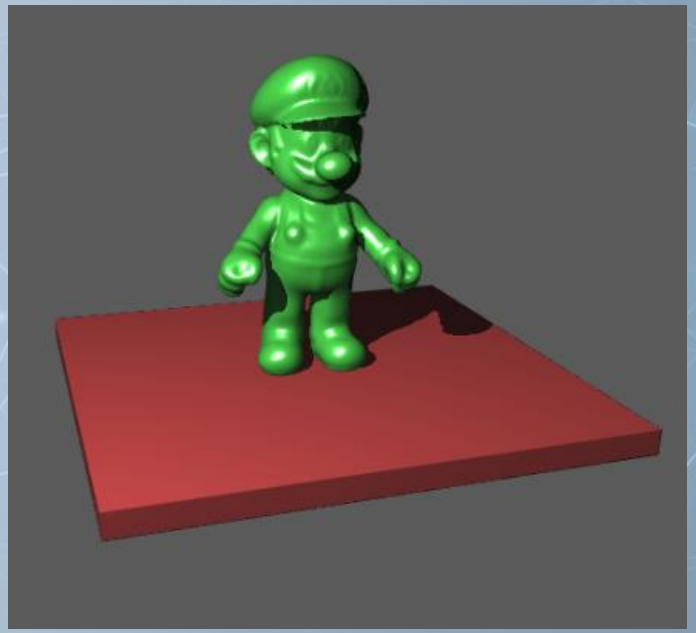


Example (Ex09-2)

- Add shadow to the scene with a mario and a plane

- Files:





Example (Ex09-2)

- main() in WebGL.js
- We have two shaders now
  - We have to compile both of them

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

    //setup shaders and prepare shader variables
    shadowProgram = compileShader(gl, VSHADER_SHADOW_SOURCE, FSHADER_SHADOW_SOURCE);
    shadowProgram.a_Position = gl.getAttribLocation(shadowProgram, 'a_Position');
    shadowProgram.u_MvpMatrix = gl.getUniformLocation(shadowProgram, 'u_MvpMatrix');

    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_MvpMatrixOfLight = gl.getUniformLocation(program, 'u_MvpMatrixOfLight');
    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_shininess = gl.getUniformLocation(program, 'u_shininess');
    program.u_Sampler0 = gl.getUniformLocation(program, "u_Sampler0");
    program.u_ShadowMap = gl.getUniformLocation(program, "u_ShadowMap");
    program.u_Color = gl.getUniformLocation(program, 'u_Color');
```

Example (Ex09-2)

- draw() in WebGL.js
- Firstly,
  - we active shadowProgram
  - set the created frame buffer (fbo) as the destination of this off-screen rendering (for depth info.)
  - Draw the plane (cube), then mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Firstly,
  - we active shadowProgram
  - set the created frame buffer (fbo) as the destination of this off-screen rendering (for depth info.)
  - Draw the plane (cube), then mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Firstly,
  - we active shadowProgram
  - set the created frame buffer (fbo) as the destination of this off-screen rendering (for depth info.)
  - Draw the plane (cube), then mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Firstly,
  - we active shadowProgram
  - set the created frame buffer (fbo) as the destination of this off-screen rendering (for depth info.)
  - Draw the plane (cube), then mario
- The function “drawOffScreen()” returns the mvpMatrix (model, view, projection matrix) of the object. We will use it for on-screen rendering

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Then,
  - Active the shadow for normal rendering
  - Set the destination of rendering to the default buffer
  - Call drawOneObjectOnScreen() to render the cube and mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Then,
  - Active the shadow for normal rendering
  - Set the destination of rendering to the default buffer
  - Call drawOneObjectOnScreen() to render the cube and mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- draw() in WebGL.js
- Then,
  - Active the shadow for normal rendering
  - Set the destination of rendering to the default buffer
  - Call drawOneObjectOnScreen() to render the cube and mario

```
function draw(){
  ///// off scree shadow
  gl.useProgram(shadowProgram);
  gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
  gl.viewport(0, 0, offScreenWidth, offScreenHeight);
  gl.clearColor(0.0, 0.0, 0.0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);
  //cube
  let cubeMdlMatrix = new Matrix4();
  cubeMdlMatrix.setScale(2.0, 0.1, 2.0);
  let cubeMvpFromLight = drawOffScreen(cubeObj, cubeMdlMatrix);
  //mario
  let marioMdlMatrix = new Matrix4();
  marioMdlMatrix.setTranslate(0.0, 1.4, 0.0);
  marioMdlMatrix.scale(0.02,0.02,0.02);
  let marioMvpFromLight = drawOffScreen(marioObj, marioMdlMatrix);

  ///// on scree rendering
  gl.useProgram(program);
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
  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);
  //cube
  drawOneObjectOnScreen(cubeObj, cubeMdlMatrix, cubeMvpFromLight, 1.0, 0.4, 0.4);
  //mario
  drawOneObjectOnScreen(marioObj, marioMdlMatrix, marioMvpFromLight, 0.4, 1.0, 0.4);
}
```

Example (Ex09-2)

- drawOffScreen() in WebGL.js
- Return mvpFromLight because we need it for on-screen rendering

```
function drawOffScreen(obj, mdlMatrix){
  var mvpFromLight = new Matrix4();
  //model Matrix (part of the mvp matrix)
  let modelMatrix = new Matrix4();
  modelMatrix.setRotate(angleY, 0, 1, 0);
  modelMatrix.rotate(angleX, 0, 1, 0);
  modelMatrix.multiply(mdlMatrix);
  //mvp: projection * view * model matrix
  mvpFromLight.setPerspective(70, offScreenWidth/offScreenHeight, 1, 15);
  mvpFromLight.lookAt(lightX, lightY, lightZ, 0, 0, 0, 0, 1, 0);
  mvpFromLight.multiply(modelMatrix);

  gl.uniformMatrix4fv(shadowProgram.u_MvpMatrix, false, mvpFromLight.elements);

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

  return mvpFromLight;
}
```

Example (Ex09-2)

- drawOnScreen() in WebGL.js
- Pass the texture which contain the depth information to on-screen rendering shader

```
function drawOneObjectOnScreen(obj, mdlMatrix, mvpFromLight, colorR, colorG, colorB){
  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
  modelMatrix.multiply(mdlMatrix);
  //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);

  //normal matrix
  let normalMatrix = new Matrix4();
  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.uniform1i(program.u_ShadowMap, 0);
  gl.uniform3f(program.u_Color, colorR, colorG, colorB);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpFromCamera.elements);
  gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
  gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);
  gl.uniformMatrix4fv(program.u_MvpMatrixOfLight, false, mvpFromLight.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, fbo.texture);

  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);
  }
}
```

Example (Ex09-2)

- Shaders to calculate depth from light
- **gl_FragCoord:**
  - gl_FragCoord is automatically calculated from gl_Position (clips pace)
  - gl_FragCoord.xy is in window device coordinate system (pixels)
    - We have introduced the calculation of gl_FragCoord.xy in the topic “space transformation”
  - gl_FragCoord.z is between 0 to 1 (depth)
    - gl_FragCoord.z = (gl_Position.z/gl_Position.w) / 2 + 0.5

```
var VSHADER_SHADOW_SOURCE = `
    attribute vec4 a_Position;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_ViewMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_ProjMatrix;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
    }
`;

var FSHADER_SHADOW_SOURCE = `
    precision mediump float;
    void main(){
      ////////** LOW precision depth implementation **/////
      gl_FragColor = vec4(gl_FragCoord.z, 0.0, 0.0, 1.0);
    }
`;
```

Example (Ex09-2)

- Vertex shaders for on-screen rendering
- a_Position: coordinate in object space
- v_PositionFromLight: coordinate in clip space from light

On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    uniform mat4 u_ProjMatrixFromLight;
    uniform mat4 u_MvpMatrixOfLight;
    varying vec4 v_PositionFromLight
    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));
        v_PositionFromLight = u_MvpMatrixOfLight * a_Position; //for shadow
    }
`;
```

Example (Ex09-2)

- Fragment shaders for on-screen rendering

On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

```
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;
    uniform sampler2D u_ShadowMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying vec4 v_PositionFromLight;
    const float deMachThreshold = 0.005; //0.001 if having high precision depth
    void main(){
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        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;
        }

        //**** shadow
        vec3 shadowCoord = (v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;
        vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);
        ////////******* LOW precision depth implementation *******//////////
        float depth = rgbaDepth.r;
        float visibility = (shadowCoord.z > depth + deMachThreshold) ? 0.3 : 1.0;

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

Example (Ex09-2)

- Fragment shaders for on-screen rendering

On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

```
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;
    uniform sampler2D u_ShadowMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying vec4 v_PositionFromLight;
    const float deMachThreshold = 0.005; //0.001 if having high precision depth
    void main(){
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        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;
        }

        //***** shadow
        vec3 shadowCoord = (v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;
        vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);
        ////////******* LOW precision depth implementation *******//////////
        float depth = rgbaDepth.r;
        float visibility = (shadowCoord.z > depth + deMachThreshold) ? 0.3 : 1.0;

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

Example (Ex09-2)

- Fragment shaders for on-screen rendering

On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

```
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;
    uniform sampler2D u_ShadowMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying vec4 v_PositionFromLight;
    const float deMachThreshold = 0.005; //0.001 if having high precision depth
    void main(){
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        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;
        }

        //***** shadow
        vec3 shadowCoord = (v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;
        vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);
        ////////******* LOW precision depth implementation *******//////////
        float depth = rgbaDepth.r;
        float visibility = (shadowCoord.z > depth + deMachThreshold) ? 0.3 : 1.0;

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

Example (Ex09-2)

- Fragment shader for on-screen rendering

On-screen shader:

1. Calculate the coordinate of the clip space from LIGHT of the object point
   - coordLightClipSpace= mvpMatrixFromLight * a_Position
   - coordLightClipSpace is a homogenous coordinate.
   - So, coordLightClipSpace = coordLightClipSpace.xyz/ coordLightClipSpace.w

2. Valid range of XY plane in clip space is -1 to +1. Texture coordinate is defined in between 0 – 1
   1. texCoordS = coordLightClipSpace.x/2 + 0.5
   2. texCoordT = coordLightClipSpace.y/2 + 0.5

3. Use [texCoordS, texCoordT] to access the texture and get the depth of the closest object point

4. Depth from the object point to light**: coordLightClipSpace.z**
   - Compare coordLightClipSpace.z and the depth of the closest object point to determine the visibility

```
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;
    uniform sampler2D u_ShadowMap;
    varying vec3 v_Normal;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying vec4 v_PositionFromLight;
    const float deMachThreshold = 0.005; //0.001 if having high precision depth
    void main(){
        vec3 ambientLightColor = u_Color;
        vec3 diffuseLightColor = u_Color;
        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;
        }

        //***** shadow
        vec3 shadowCoord = (v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;
        vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);
        ////////******* LOW precision depth implementation *******//////////
        float depth = rgbaDepth.r;
        float visibility = (shadowCoord.z > depth + deMachThreshold) ? 0.3 : 1.0;

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

Try and Think (5 mins)

- This implementation for shadow is not perfect. But it is a good practice for frame buffer
  - Rotate the scene, you will find out some imperfect results
- What if you set “deMachThreshold” to 0? Why do we need it?
- I comment this in shaders “Low precision implementation”. Can you guess What I mean?