컴퓨터 그래픽스 과제 3

2018204058 김민교

1. 환경맵 설정

```
96   //texture map setting
97   const textureLoader = new THREE.TextureLoader();
98   const textureEquirec = textureLoader.load("../myenv.jpg");
99   textureEquirec.mapping = THREE.EquirectangularReflectionMapping;
100  scene.background = textureEquirec;
```

환경맵을 EquirectangularReflectionMapping 으로 설정했다.

2. Three JS에서 전달해줄 변수 선언

```
103  const material = new THREE.RawShaderMaterial({
104    uniforms: {
105      time: { value: 1.0 },
106      envMap: { value: textureEquirec },
107      modelMatrix: { value: 0 },
108      normalMatrix: { value: 0 },
109      lightPosition: { value: new THREE.Vector3(30, 30, 30) },
110      lightColor: { value: new THREE.Color(0xff00ff) },
111      ambientColor: { value: new THREE.Color(0x404040) },
112      materialColor: { value: new THREE.Color(0x40ffff) },
113      cameraPosition: { value: 0 },
114      shininess: { value: 20 },
115    },
116    vertexShader: document.getElementById("vertexShader").textContent,
117    fragmentShader: document.getElementById("fragmentShader").textContent,
118    side: THREE.DoubleSide,
119    transparent: true,
120  });
121
122  material.envMap = textureEquirec;
```

shader programing에 보낼 변수 목록 :

envMap : 환경맵

modelMatrix : cube의 모델 트랜스폼 매트릭스

normalMatrix : cube 모델 스페이스에서 노말들을 월드 스페이스로 변환하는 매트릭스

lightPosition : 월드 상의 포인트 라이트 위치

lightColor : 라이트의 색깔

ambientColor : 앰비언트 컬러

materialColor : 큐브의 매터리얼 컬러

cameraPosition : 카메라 위치

shininess : 물질이 매끄러운 정도

여기서 modelMatrix, normalMatrix, cameraPosition은 rendering 될 때마다 바뀌는 값이다. 그래서 render될 때마다 변한 값을 shader 프로그램에 보내주어야 한다.

3. Rendering 할 때마다 변수 보내기

```
140   // render a scene using a camera before drawing the next frame on the screen
141   renderer.setAnimationLoop(() => {
142     //----------------- cube ModelMatrix -----------------
143     let cubeMatrix = cube.matrixWorld.clone();
144     //  큐브 모델 매트릭스를 GPU에 보내자~
145     cube.material.uniforms.modelMatrix.value = cubeMatrix.clone();
146
147     //----------------- 노말 트랜스폼 매트릭스를 구하자. -----------------
148     // 1. 모델 매트릭스에 역행렬 취하고  2. 트랜스포즈 하기
149     let normalMatrix = cubeMatrix.clone().invert().transpose();
150     // 노말 변환 매트릭스를 GPU에 보내자~
151     cube.material.uniforms.normalMatrix.value = normalMatrix.clone();
152
153     //----------------- 카메라 위치를 구하자. -----------------
154     let cameraPosition = new THREE.Vector3();
155     // ----------------- 카메라의 월드 스페이스의 위치 구하기-----------------
156     camera.getWorldPosition(cameraPosition);
157     // ----------------- 카메라 월드 상 위치를 GPU에 보내자~
158     cube.material.uniforms.cameraPosition.value = cameraPosition.clone();
159
160     renderer.render(scene, camera);
161   });
```

line 143~ 145: Cube의 월드 매트릭스를 구하고, 이것을 GPU로 보낸다!

line 147 ~ 151 : Cube의 월드 매트릭스의 역행렬을 구하고 트랜스포즈를 취해서 노말 변환 매트릭스를 구한다. 이 것을 렌더링 할 때마다 GPU에 보낸다.

line 154 ~ 158 : 카메라의 월드에서의 위치를 구한다. 이것을 렌더링 할 때마다 GPU로 보낸다.

4. Vertex & Fragment Shader

- Vertex Shader

```
        <script id="vertexShader" type="x-shader/x-vertex">

Line 1      precision mediump float;
Line 2      precision mediump int;
Line 3
Line 4      uniform mat4 modelViewMatrix;
Line 5      uniform mat4 projectionMatrix;
Line 6      // Assignment TO DO //
Line 7      // get your model matrix
Line 8
Line 9      uniform mat4 modelMatrix; // 모델 트랜스폼 매트릭스
Line 10     uniform mat4 normalMatrix; // 노말 트랜스폼 시키는 매트릭스
Line 11
Line 12     attribute vec3 position;
Line 13     attribute vec3 normal;
Line 14
Line 15     varying vec3 vPosition;
Line 16     varying vec3 vNormal; // 이거를 fragmentshader 에 전달해주어야함.
Line 17
Line 18     void main()    {
Line 19         // Assignment TO DO //
Line 20         // choose your lighting space (world or camera space) : world
Line 21
Line 22         // transform position and normal accordingly
Line 23
Line 24         // 포지션 월드 스페이스로
Line 25         vPosition = (modelMatrix*vec4(position,1.0)).xyz;
Line 26
Line 27         //노말 월드스페이스로
Line 28         vNormal= (normalMatrix * vec4(normal,0.0)).xyz;
Line 29
Line 30         gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
Line 31
Line 32     }
Line 33     </script>
```

Line 9~10 : uniform 한정자로 mat4타입의 modelMatrix와 normalMatrix를 선언했다. uniform 한정자는 전역 셰이더 변수이다. 사용자가 셰이더 프로그램에 전달하는 값이다. 읽을 수만 있다. modelMatrix와 normalMatrix는 JavaScript에서 전달해주는 값이다.

Line 12 ~ 13: attribute 한정자로 vec3타입의 position, normal를 선언했다. 이 값은 Vertex의 attribute값이다.

Line 15 ~16: varying 한정자로 vec3타입의 vPosition, vNormal를 선언했다. varying 한정자는 fragment shader에 보내는 출력값을 나타낸다. 이 값들을 Rasterizer가 보간하고 fragment shader에 보낸다!

Line 25 : JavaScript에서 보낸 modelMatrix 값으로 모델 스페이스에 있는 position 값을 월드 스페이

스로 변환한다. vPosition에 변환한 위치 값을 넣는다.

Line 28 : JavaScript에서 보낸 normalMatrix 값으로 모델 스페이스에 있는 normal 값을 월드 스페이스로 변환한다. vNormal에 변환한 노말 값을 넣는다.

- FragmentShader

```
<script id="fragmentShader" type="x-shader/x-fragment">
#define X_PI 3.14159265358979323846

precision mediump float;
precision mediump int;

uniform sampler2D envMap;

// Assignment TO DO //

 //-- get Point Light position--
uniform vec3 lightPosition;

//-- get Point Light color--
uniform vec3 lightColor;

//-- get Amient Light color--
uniform vec3 ambientColor;

//-- get Camera position--
uniform vec3 cameraPosition;

//-- get Material color--
uniform vec3 materialColor;

//-- get Shininess--
uniform float shininess;

// Assignment TO DO //
// get necessary attributes (interpolated)
varying vec3 vPosition;
varying vec3 vNormal;

void main()      {
    // compute Phong Lighting

    //-- Normal 벡터--
    vec3 N = normalize(vNormal);

    //-- Light 벡터-- : 빛 위치 - 표면
    vec3 L = normalize(lightPosition-vPosition) ;

    //-- View 벡터-- : 카메라 위치 - 표면
    vec3 V = normalize(cameraPosition-vPosition);

    //-- Reflector 벡터-- :
    vec3 R = 2.0 * dot(N,L)*N - L;
```

```
Line 48         //-- 환경맵의 텍스처를 맵핑하기 위한 뷰의 반사 벡터
Line 49         vec3 R_V = 2.0*(dot(N,V))*N-V;
Line 50
Line 51         //-- 환경맵의 텍스처 좌표를 구하자잉~ (강의교안 참조 Blinn/Newell Latitude Mapping)
Line 52         float u =( atan(R_V.x/R_V.z) + X_PI )/ (2.0* X_PI);
Line 53         float v =( asin(R_V.y) + X_PI/2.0 )/X_PI;
Line 54
Line 55         //-- 텍스처 좌표의 컬러
Line 56         vec3 colorEnv = texture2D( envMap, vec2(u, v) ).rgb;
Line 57
Line 58         //compute the diffuse term
Line 59         float diffuse = max(dot(L, N), 0.0);
Line 60         vec3 diffuseTerm = diffuse*vec3(materialColor.x * lightColor.x , materialColor.y *
Line 61 lightColor.y, materialColor.z * lightColor.z);
Line 62
Line 63         //compute the specular term
Line 64
Line 65         float specular =pow(max(dot(R_V,L),0.0),shininess);
Line 66         vec3 specularTerm = specular*vec3(materialColor.x*colorEnv.x,
Line 67 materialColor.y*colorEnv.y,materialColor.z*colorEnv.z);
Line 68
Line 69         //compute the ambient term
Line 70         vec3 ambientTerm = vec3(materialColor.x * ambientColor.x , materialColor.y *
Line 71 ambientColor.y, materialColor.z * ambientColor.z);
Line 72
Line 73         //-- 최종 색깔 결정 : diffuse Term + specularTerm + ambientTerm
Line 74         vec3 final_color = diffuseTerm+ambientTerm+specularTerm;
Line 75
Line 76         gl_FragColor.rgb = final_color;
Line 77         gl_FragColor.a = 1.0;
Line 78     }
      </script>
```

Line 6 : JavaScript에서 전달해주는 환경맵

Line 7~22 : JavaScript에서 전달한 값들을 받는다.

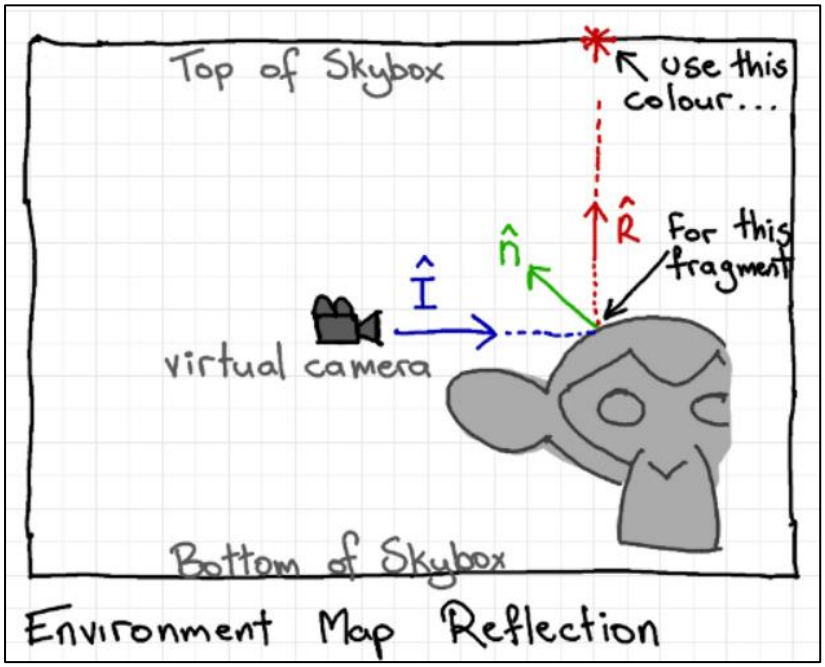
Line 37 : vNormal을 노말라이즈드 한다. 퐁라이팅 계산할 때, 노말라이즈된 벡터로 계산해야되기 때문이다.

Line 40 : LightVector를 구한다. LightVector는 표면에서 빛으로 가는 방향이다. 이 벡터 또한 노말라이즈 한다.

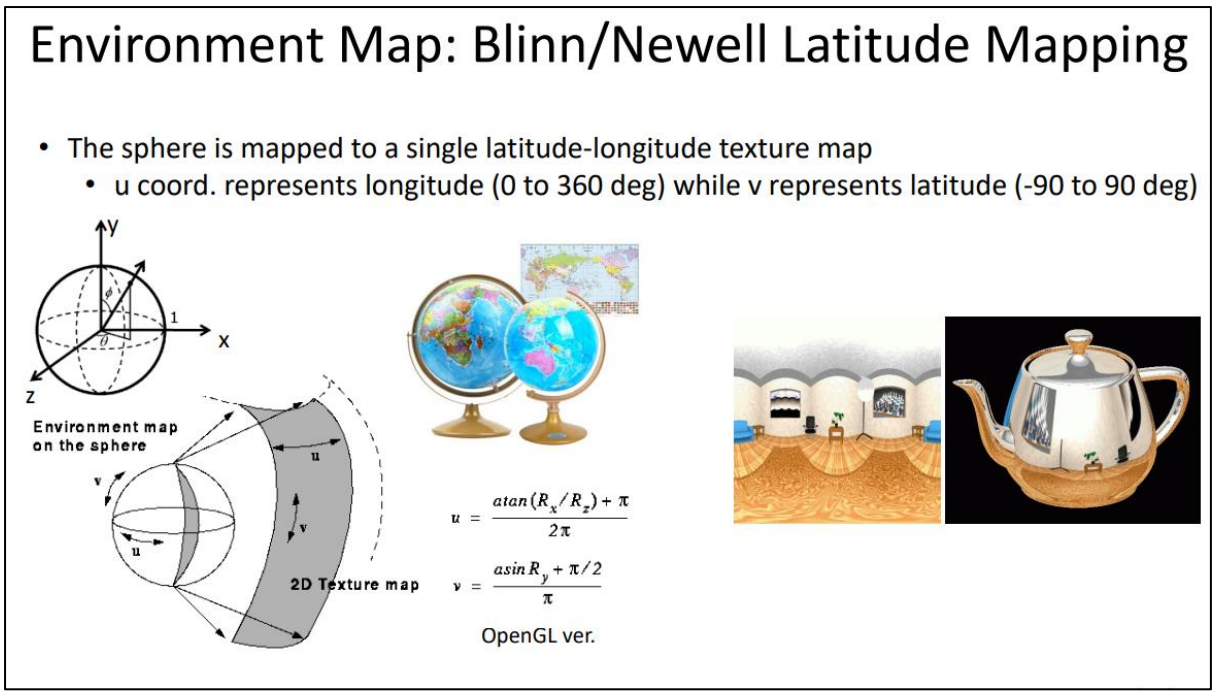
Line 43 : View Vector를 구한다. ViewVector는 표면에서 카메라로 가는 방향이다. 이 벡터 또한 노말라이즈한다.

Line 46 : Reflection Vector를 구한다. 이것은 빛의 입사에 따른 반사되는 벡터이다

Line 49 :R_V는 뷰벡터의 반사 벡터이다. 이 벡터가 쭉 직진해서 환경맵에 닿는다. 이 때, 닿은 텍스처 u,v 좌표의 컬러가 Reflector Light의 Color가 될 것 이다.



Line 52 ~ Line 53 : Blinn/Newell Latitude Mapping 방식을 이용해 texture의 u,v값을 구한다.



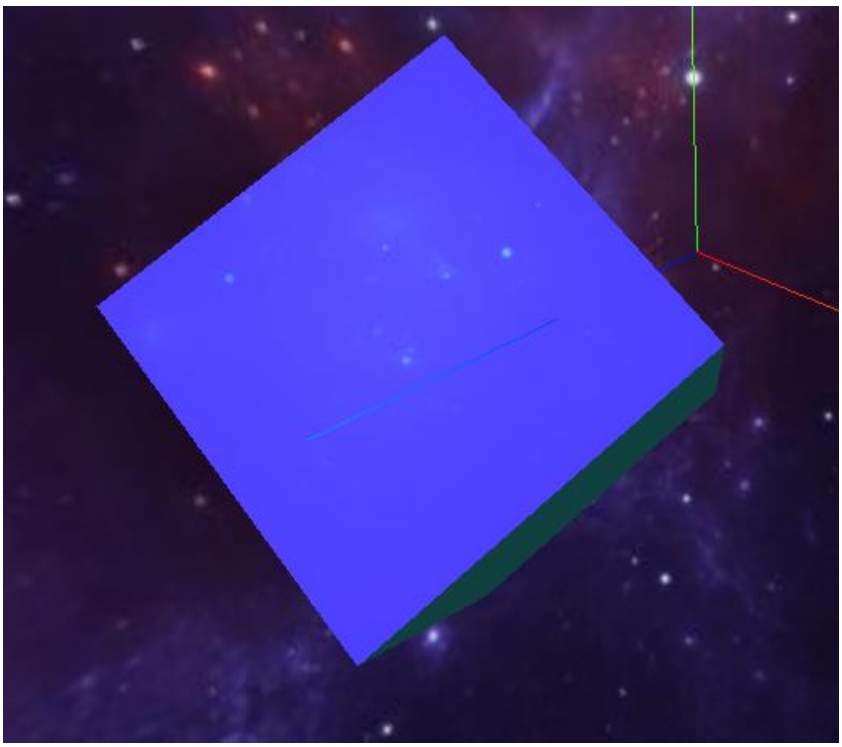
line 56 : u,v 를 이용해 texture의 컬러를 뽑아낸다.

line 66~67 : 환경맵에 맵핑되는 컬러를 ReflectorLightColor로 지정하여 specular term을 구한다.

최종적으로 diffuse Term, SpecularTemr, AmbientTerm을 전부 구해서 최종 색깔을 얻는다.

5. 결과

- Shininess : 30



- Shininess : 200