

컴퓨터 그래픽스 과제 2

2018204058 김민교

1. Three JS에서 전달해줄 변수 선언

```
106 const material = new THREE.RawShaderMaterial({
107   uniforms: {
108     time: { value: 1.0 },
109     modelMatrix: { value: 0 },
110     normalMatrix: { value: 0 },
111     lightPosition: { value: new THREE.Vector3(30, 30, 30) },
112     lightColor: { value: new THREE.Color(0xff00ff) },
113     ambientColor: { value: new THREE.Color(0x404040) },
114     materialColor: { value: new THREE.Color(0x40ffff) },
115     cameraPosition: { value: 0 },
116     shininess: { value: 300 },
117   },
118   vertexShader: document.getElementById("vertexShader").textContent,
119   fragmentShader: document.getElementById("fragmentShader").textContent,
120   side: THREE.DoubleSide,
121   transparent: true,
122 });
```

shader programming에 보낼 변수 목록 :

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.

2. Rendering 할 때마다 변수 보내기

```
167 // render a scene using a camera before drawing the next frame on the screen
168 renderer.setAnimationLoop(() => {
169     //----- cube ModelMatrix -----
170     let cubeMatrix = cube.matrixWorld.clone();
171     // 큐브 모델 매트릭스를 GPU에 보내자~
172     cube.material.uniforms.modelMatrix.value = cubeMatrix.clone();
173
174     //----- 노말 트랜스폼 매트릭스를 구하자. -----
175     // 1. 모델 매트릭스에 역행렬 취하고 2. 트랜스포즈 하기
176     let normalMatrix = cubeMatrix.clone().invert().transpose();
177     // 노말 변환 매트릭스를 GPU에 보내자~
178     cube.material.uniforms.normalMatrix.value = normalMatrix.clone();
179
180     //----- 카메라 위치를 구하자. -----
181     let cameraPosition = new THREE.Vector3();
182     // ----- 카메라의 월드 스페이스의 위치 구하기-----
183     camera.getWorldPosition(cameraPosition);
184     // ----- 카메라 월드 상 위치를 GPU에 보내자~
185     cube.material.uniforms.cameraPosition.value = cameraPosition.clone();
186
187     renderer.render(scene, camera);
188 });
```

line 170 ~ 172 : Cube의 월드 매트릭스를 구하고, 이것을 GPU로 보낸다!

line 176 ~ 178 : Cube의 월드 매트릭스의 역행렬을 구하고 트랜스포즈를 취해서 노말 변환 매트릭스를 구한다. 이 것을 렌더링 할 때마다 GPU에 보낸다.

line 181 ~ 185 : 카메라의 월드에서의 위치를 구한다. 이것을 렌더링 할 때마다 GPU로 보낸다.

$$K = (M^{-1})^T$$

3. Vertex & Fragment Shader

- Vertex Shader

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

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

	<code><script id="fragmentShader" type="x-shader/x-fragment"></code>
Line 1	<code>precision mediump float;</code>
Line 2	<code>precision mediump int;</code>
Line 3	
Line 4	<code>uniform float time;</code>
Line 5	<code>// Assignment TO DO //</code>
Line 6	<code>// get Point Light position</code>
Line 7	<code>uniform vec3 lightPosition;</code>
Line 8	
Line 9	<code>// get Point Light color</code>
Line 10	<code>uniform vec3 lightColor;</code>
Line 11	
Line 12	<code>// get Ambient Light color</code>
Line 13	<code>uniform vec3 ambientColor;</code>
Line 14	
Line 15	<code>// get Camera position</code>
Line 16	<code>uniform vec3 cameraPosition;</code>
Line 17	
Line 18	<code>// get Material color</code>
Line 19	<code>uniform vec3 materialColor;</code>
Line 20	
Line 21	<code>//get Shininess</code>
Line 22	<code>uniform float shininess;</code>
Line 23	
Line 24	<code>varying vec3 vPosition;</code>
Line 25	<code>// Assignment TO DO //</code>
Line 26	<code>// get necessary attributes (interpolated)</code>
Line 27	<code>varying vec3 vNormal;</code>
Line 28	
Line 29	<code>void main() {</code>
Line 30	
Line 31	<code> // compute Phong Lighting</code>
Line 32	
Line 33	<code> //-- Normal 벡터--</code>
Line 34	<code> vec3 N = normalize(vNormal);</code>
Line 35	
Line 36	<code> //-- Light 벡터-- : 빛 위치 - 표면</code>
Line 37	<code> vec3 L = normalize(lightPosition-vPosition) ;</code>
Line 38	
Line 39	<code> //-- View 벡터-- : 카메라 위치 - 표면</code>
Line 40	<code> vec3 V = normalize(cameraPosition-vPosition);</code>
Line 41	
Line 42	<code> //-- Reflector 벡터-- :</code>
Line 43	<code> vec3 R = 2.0 * dot(N,L)*N - L;</code>
Line 44	
Line 45	<code> //compute the diffuse term</code>

```

Line 46         float diffuse = max(dot(L, N), 0.0);
Line 47         vec3 diffuseTerm = diffuse*vec3(materialColor.x * lightColor.x , materialColor.y *
Line 48         lightColor.y, materialColor.z * lightColor.z);
Line 49
Line 50         //compute the specular term
Line 51         float specular = pow(max(dot(R,V),0.0),shininess);
Line 52         vec3 specularTerm = specular*vec3(materialColor.x*lightColor.x, materialColor.y
Line 53         *lightColor.y, materialColor.z*lightColor.z);
Line 54
Line 55         //compute the ambient term
Line 56         vec3 ambientTerm = vec3(materialColor.x * ambientColor.x , materialColor.y *
Line 57         ambientColor.y, materialColor.z * ambientColor.z);
Line 58
Line 59         //-- 최종 색깔 결정 : diffuse Term + specularTerm + ambientTerm
Line 60         vec3 final_color = diffuseTerm+ambientTerm+specularTerm;
Line 61
Line 62         gl_FragColor.rgb = final_color;
Line 63         gl_FragColor.a = 1.0;
Line 64
Line 65     }
Line 66     </script>

```

Line 7~22 : JavaScript에서 전달한 값들을 받는다.

Line 34 : vNormal을 노말라이즈드 한다. 풍라이팅 계산할 때, 노말라이즈된 벡터로 계산해야되기 때문이다.

Line37 : LightVector를 구한다. LightVector는 표면에서 빛으로 가는 방향이다. 이 벡터 또한 노말라이즈 한다.

Line 40 : View Vector를 구한다. ViewVector는 표면에서 카메라로 가는 방향이다. 이 벡터 또한 노말라이즈한다.

Line 43 : Reflection Vector를 구한다. 이것은 빛의 입사에 따른 반사되는 벡터이다.

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Direct lighting simplification with Phong lighting model

$$color_{\{R,G,B\}} = \underbrace{m_{\{R,G,B\}}^{ambient} \cdot I_{\{R,G,B\}}^{ambient}}_{\text{ambient}} + \sum_{i=1}^{num_lights} \underbrace{\frac{1}{k_c + k_i d_i + k_q d_i^2}}_{\text{attenuation}} \left(\underbrace{m_{\{R,G,B\}}^{diffuse} \cdot I_{i,\{R,G,B\}}^{diffuse} \cdot \max(L_i \cdot N, 0)}_{\text{diffuse}} + \underbrace{m_{\{R,G,B\}}^{specular} \cdot I_{i,\{R,G,B\}}^{specular} \cdot \max(R_i \cdot V, 0)^{shininess}}_{\text{specular}} \right)$$

LightVector와 노말 벡터로는 디퓨즈 텀을 구할 것이다.

LightVector, 노말 벡터, 뷰벡터, R벡터로는 스페큘러 텀을 구할 것이다.

Line 46 ~ 48 : diffuse Term을 구한다

Line 51~54 : Specular Term을 구한다.

Line 56 ~57 : ambient Term을 구한다.

Line 60 : final color는 diffuse Term + Specular Term + ambient Term을 구한 값이다.

Line 62 ~ 63 :gl_FragColor로 색칠해질 값을 설정할 수 있다. gl_FragColor.rgb를 final 컬러로 대체한다. gl_FragColor.a는 투명도인데, 1로 설정하여 투명함이 없게 한다.

4. 결과

