컴퓨터 그래픽스 과제
- 실습 2 레포트 -

과목명: 컴퓨터그래픽스
담당 교수: 김동준 교수님
제출일: 2022. 11. 15
전공: 정보융합학부
학번: 2020204097
이름: 윤가영

1. Lighting and Shading

Lighting은 빛과 표면 사이의 상호작용을 하며 랜더링 시에 색을 결정한다. 즉 빛과 노말벡터, 그리고 물질의 속성으로 해당 지점의 색을 계산한다. 그 대상이 꼭짓점일수도, 픽셀일수도 있다. Shading은 각 픽셀마다 색을 계산하며 interpolation으로 픽셀의 정보를 채운다. Lighting과 다르게 그 대상이 꼭짓점 색일수도, 노말벡터일수도 있다. Shading은 Lighting의 단위에 따라 세 가지 종류로 구분된다.

(1) Flat Shading

Flat shading은 삼각형 단위로 색을 계산한다. 연산 속도가 빠르지만 평평하고 비현실적인 외관을 만든다. 그렇기 때문에 보통 도형의 해상도를 보고 싶을 때 사용한다.

(2) Gouraud Shading (Per-Vertex Lighting)

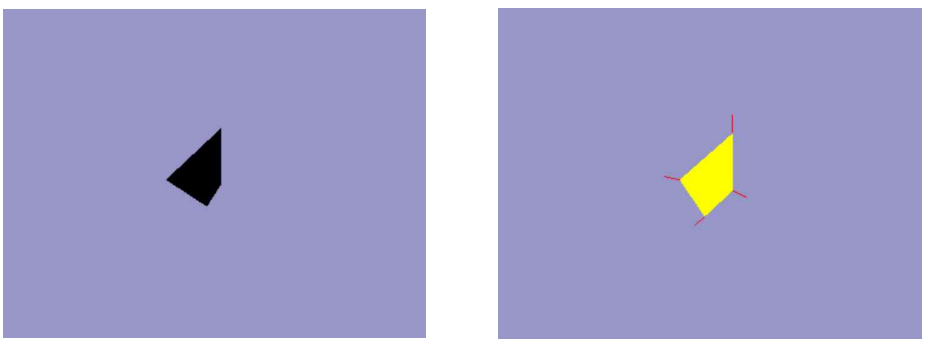
Gouraud shading은 per-vertex lighting이 단위이기 때문에 색을 한 꼭짓점마다 계산한다. 즉, 삼각형 안에 있는 모든 조각에 꼭짓점 단위로 색을 채운다. 우선 Lighting이 꼭짓점마다 계산하여 색을 결정하고, 그 후 Shading이 결정된 색으로 interpolation하여 표면을 결정한다. 그렇기 때문에 연산 속도가 빠르지만, 여전히 평평하고 비현실적인 하이라이트를 가진다.

(3) Phong Shading (Per-Pixel Lighting)

Phong shading은 per-pixel lighting이 단위기에 픽셀 단위로 계산한다. 빛 연산을 수행하기 위해 모든 조각에 꼭짓점마다 있는 노말벡터를 interpolation 해야 하는 필요성이 있다. 때문에 Shading 중에서 가장 부드럽고 자연스럽다. 간혹 pixel shader가 아닌 fragment shader라고 부를 때가 있는데, 그것은 OpenGL에서 부르는 용어로 지칭한다. pixel shader는 Direct3D 일때의 용어다.

2. Normal 설정

빛의 반사는 법선이 있어야 생긴다. 자동으로 외적을 통해 꼭짓점의 법선을 구해주는 함수로 computeVertexNormals()가 있다. 법선벡터를 구한 것을 화면으로 보여주기 위해서는 VertexNormalsHelper() 함수를 사용하면 된다. Orbitcontrol처럼 import하는 것이 중요하다.



▲ 법선벡터를 넣지 않은 것 / Helper까지 추가로 넣은 것

3. Geometry 설정

도형은 BufferGeometry를 통하여 생성한다. 우선 사면체의 꼭짓점을 지정하고, 각 꼭짓점의 법선과 꼭짓점의 순서를 기술한다.

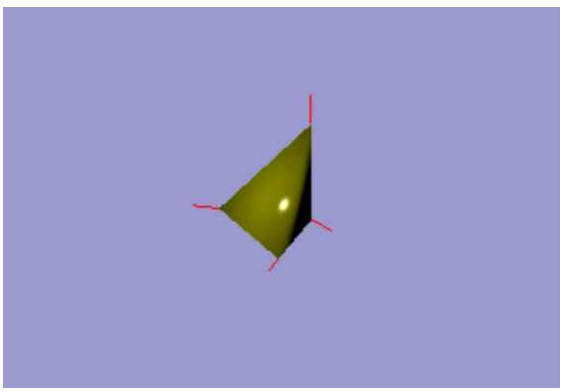
```
// geometry setting
const points = [
    10, 0, 0,
    0, 10, 0,
    0, 0, 10,
    0, 0, 0,
];
const normals = [
  1, 0, 0,
  1, 0, 0,
  1, 0, 0,
  1, 0, 0,
];


const triIndices = [
    1, 0, 3,        2, 1, 3,
    3, 0, 2,        1, 2, 0
]

const geometry = new THREE.BufferGeometry();
const pointsArray = new Float32Array(points);
const normalsArray = new Float32Array(normals);
geometry.setAttribute('position', new THREE.BufferAttribute( pointsArray, 3 ));
geometry.setIndex(triIndices);
```

4. Phong Material 설정

Lighting을 하기 위해서는 물질의 속성을 바꿔야 한다. 이때 'MeshBasicMaterial' 이 아닌 'MeshPhongMaterial' 을 사용한다. Phong Material의 기본적인 옵션에는 Flatshading과 Shiniess가 있다.

```
// material setting
const material = new THREE.MeshPhongMaterial( { color: 0xffff00, wireframe: false, flatShading: false, shininess: 300 } );
```

Flatshading은 물질이 평평한 음영으로 랜더링 되는지의 여부를 결정한다. 기본값으로 false가 되어있다. True로 하면 삼각형 단위로 설정되기 때문에 옆면이 그라데이션처럼 보이는 것을 방지할 수 있다. Shininess는 하이라이트의 정도를 설정한다. 값이 클수록 하이라이트가 선명해지며 기본값은 30으로 고정되어 있다. 현재 색은 0xffff00인 노란색으로 설정되어 있는데, 이는 반사할 색이 노란색이 된다.



▲ Phong Material까지 설정한 Geometry

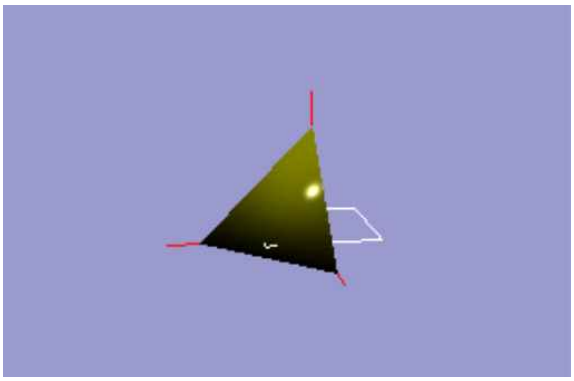
5. Light 설정

THREE.js에서는 기본적으로 'DirentionalLight'을 쓴다. 디폴트가 -y축으로 빛이 향해있기 때문에 위에서 빛이 쬐어지는 식으로 생각하면 된다.

```
const myLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
myLight.position.set(0, 1, 0);
myLight.target = myMesh2;
myScene.add(myLight);

const lightHelper = new THREE.DirectionalLightHelper( myLight, 5 );
myScene.add(lightHelper);
```

빛을 설정할 때, 기본값으로 빛의 색과 강도/세기를 입력받는다. 색은 기본값으로 하얀색이 되어있으며 강도는 1로 고정되어 있다. 위치는 화면상에 있는 Object3D와 동일하게 설정된다. 기본적으로 (0, 1, 0)으로 고정되어 있기에 빛은 위에서 아래로 비추는 형식이다.

또한 THREE.js는 빛의 효과를 시각화해주는 'DirectionalLightHelper' 기능을 제공한다. 이 함수는 평면과 빛의 위치와 방향을 나타내는 선으로 구성되어 있다.



▲ Light까지 설정한 Scene

6. Shadow 설정

빛을 'DirectionalLight'으로 해주었기 때문에, 그림자 또한 'DirectionalLightShadow' 함수를 사용하면 된다.

```
const myLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
myLight.position.set(20, 20, 20);
myLight.target = myMesh;
myLight.castShadow = true;

myLight.shadow.camera.near = 1;
myLight.shadow.camera.far = 500;
myLight.shadow.mapSize.width = 1024;
myLight.shadow.mapSize.height = 1024;
```

먼저 빛에 대한 그림자 속성을 정의해야 한다. 그림자 카메라의 near plane은 0.5가, far plane은 500으로 기본값이 설정되어 있다. 맵 사이즈 또한 가로와 세로 모두 512로 기본적으로 설정되어 있다.

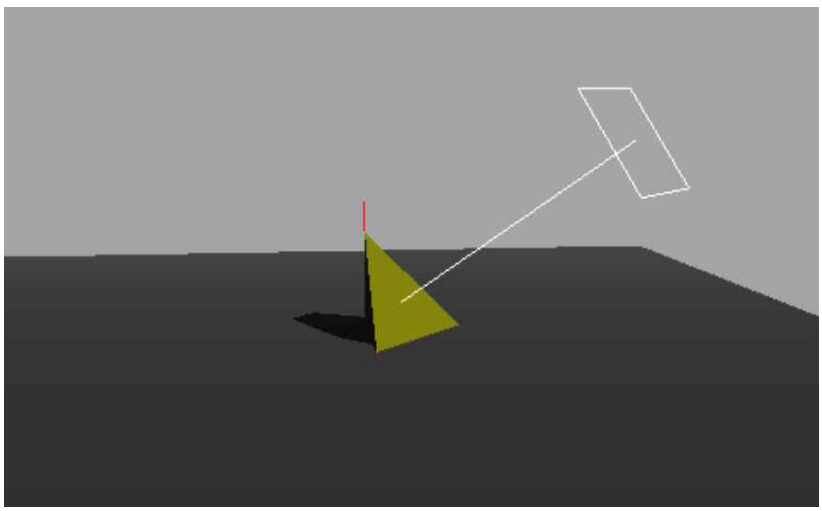
빛 뿐만 아니라 도형에도 그림자 속성을 정의해야 한다. 다음과 같이 적용하면 된다.
myMesh.castShadow = true; myMesh.receiveShadow = false;

마지막으로 그림자를 받을 평면을 추가한다.

```
const mesh = new THREE.Mesh( new THREE.PlaneGeometry( 100, 100 ), new THREE.MeshPhongMaterial( { color: 0x999999, depthWrite: false } ) );
mesh.rotation.x = - Math.PI / 2;
mesh.receiveShadow = true;
myScene.add( mesh );
```

```
const renderer = new THREE.WebGLRenderer();
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFShadowMap;
```

추가적으로 랜더러에 다음과 같이 설정할 수 있다. 'shadowMap.enabled' 는 쉐도우 맵에서 가져온 것을 처리하겠다는 것이고, 타입을 'THREE.PXFShadowMap' 으로 정의하면 쉐도우 맵의 계단현상을 줄일 수 있다.



▲ Shadow까지 설정한 Scene