컴퓨터 그래픽스 과제
- 실습 1 레포트 -

과목명: 컴퓨터그래픽스
담당 교수: 김동준 교수님
제출일: 2022. 10. 10
전공: 정보융합학부
학번: 20204097
이름: 윤가영

1. Three.js에서의 Graphics Pipeline Transforms

레스터 변환을 기본으로 한 그래픽스 파이프 라인은 연산이 4단계로 진행된다. Application, Geometry, Rasterization, 그리고 Display다. 이중 Geometry 단계에서 4개의 변환이 일어난다. 순서대로 **Model transform, View transform, Projection transform, View port transform**이다. 그리고 각 단계마다 변환을 도와주는 Three.js 도구 클래스가 존재한다.

(1) Model Transform

모델 트랜스폼은 흔히 월드 트렌스폼이라 더 불러진다. 모델 트랜스폼은 그래픽스 파이프라인에서 부르는 명칭이기 때문이다. 모델 트랜스폼은 객체 스페이스를 월드 스페이스로 변환할 때 이루어지는 단계이므로, 기본적으로 객체가 가진 공간이 필요하다. 이것은 디자이너가 결정하며 빛과 카메라 또한 객체가 가능하다.

(2) View Transform

뷰 트랜스폼은 월드 스페이스를 카메라 스페이스로 변환할 때 이루어지는 단계다. 이때 Three.js에서 **Object3D** 클래스가 유용하게 사용된다. Object3D는 월드 스페이스에 정렬된 객체의 방향과 위치를 설정하는데 도움을 준다. 그 뿐만 아니라 카메라 인터페이스까지 제공한다.

(3) Projection Tranform

프로젝션 트랜스폼은 카메라 스페이스를 프로젝션 스페이스로 변환할 때 이루어지는 단계다. 이때 'LookAt' 인터페이스를 통해 연산되며, Three.js에서는 LookAt 인터페이스에서 제공된 **Camera** 클래스가 사용된다. 카메라 스페이스의 단위는 월드 스페이스의 단위와 같으며, 카메라 기준으로 좌표계를 바꿔준다. 이는 증강현실을 할 때 사용된다.

(4) View Port Transform

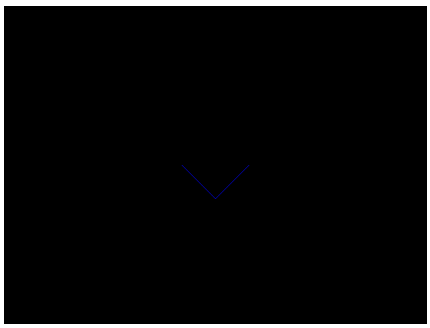
뷰 포트 프랜스폼은 프로젝션 스페이스에서 스크린 스페이스로 변환할 때 이루어지는 단계다. 프로젝션 스페이스에선 두 가지의 투영 방법이 있다. 투시와 직각이 있는데, 이중 투시 투영이 주로 쓰인다. 이때 Three.js 에서는 Object 3D에서 제공된 **PerspectiveCamera** 클래스가 사용된다. 이는 사람이 보는 방식을 묘사한 인터페이스로, 카메라의 종류 또한 프로젝션 스페이스처럼 투시 카메라와 직각 카메라 두 가지로 나뉜다.

최종적으로 디스플레이에 보여질때는 **WebGLRenderer** 클래스가 사용된다. 화면에 창을 띄우는데 .setViewport가 사용되며 구성자로 x, y, width, heightr 네 가지가 있다. x와 y는 오프셋이고 기본값은 (0, 0)이다. width와 height는 지금 그리고자 하는 이미지의 사이즈를 나타낸다.

2. Camara Parameter

(1) 일반적인 카메라 세팅

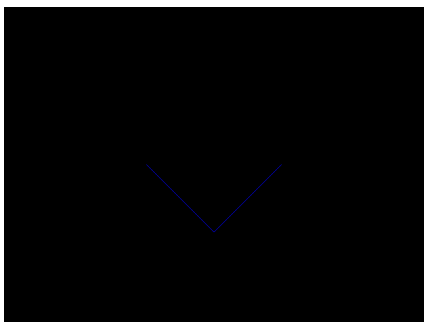
```
// camera setting
const camera = new THREE.PerspectiveCamera( 45, 640.0/480.0, 1, 500 );
camera.position.set( 0, 0, -100);
camera.up.set( 0, -1, 0 );
camera.lookAt( 0, 0, 50 );
```



카메라 세팅을 하기 위해서는 Three.js의 Camera 클래스와 PerspectiveCamera 라는 하위 클래스가 필요하다. Camera 클래스에는 **.positon .up .lookAt** 이 있다. PerspectiveCamera에는 구성자가 **fov, aspect, near, far**로 네 가지가 있다. fov는 수직 혹은 화각을, aspect는 가로 평면과 세로 평면의 비율을 나타내며, 가로/세로 형식으로 수치를 입력한다. near는 카메라의 시점이 시작되는 위치, far는 카메라의 시점이 끝나는 위치를 나타낸다.

(2) 카메라 위치의 z값이 600일 때

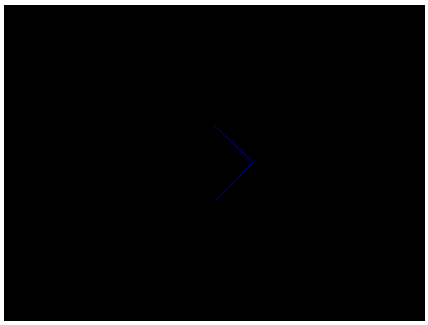
```
// camera setting
const camera = new THREE.PerspectiveCamera( 45, 640.0/480.0, 1, 500 );
camera.position.set( 0, 0, 50 );
camera.up.set( 0, -1, 0 );
camera.lookAt( 0, 0, 50 );
```



그렇기 때문에 fov의 값을 크게 하면 그만큼 물체가 작게 보이고, 반대로 작은 값을 넣으면 물체가 크게 보인다. camera.positon.set은 카메라의 위치를 설정한다. 가령 camera.position.set(10, 10, 10)이라면 각 축에서 10만큼 떨어져 있는 것이다. 현재 카메라의 시점은 1~500로 수치가 정해져 있으므로, 만일 카메라 위치의 z축 값에 600을 넣는다면 물체는 보이지 않는다. 반대로 50을 넣으면 물체가 크게 보인다. 이때, -50을 넣어도 똑같은 결과가 나온다.

(3) 업벡터 세팅

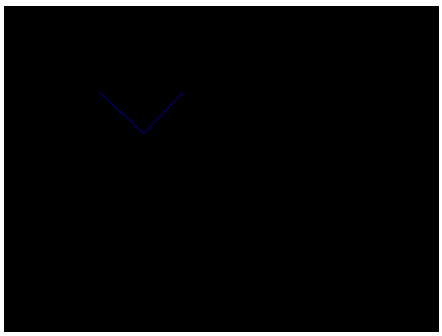
```
// camera setting
const camera = new THREE.PerspectiveCamera( 45, 640.0/480.0, 1, 500 );
camera.position.set( 0, 0, -100 );
camera.up.set( 50, 1, 0 );
camera.lookAt( 0, 0, 50 );
```



camera.up.set은 업벡터로 특정 방향을 가리키며 카메라를 회전한다. 이때 x와 y 값의 부호에 따라 그려지는 것이 다르다. x값이 양수일 경우, 시계방향으로 회전하고 음수일 경우 반시계 방향으로 회전한다. y값이 양수면 위로 뒤집혀진다.

(4) lookAt 세팅

```
// camera setting
const camera = new THREE.PerspectiveCamera( 45, 640.0/480.0, 1, 500 );
camera.position.set( 0, 0, -100 );
camera.up.set( 0, -1, 0 );
camera.lookAt( 20, 20, 1 );
```



camera.lookAt은 카메라가 주시하는 방향으로, x축이 양수면 왼쪽에, 음수면 오른쪽으로 카메라를 주시한다. 이와 비슷하게 y축이 양수면 위에, 음수면 아래에 위치시킨다.

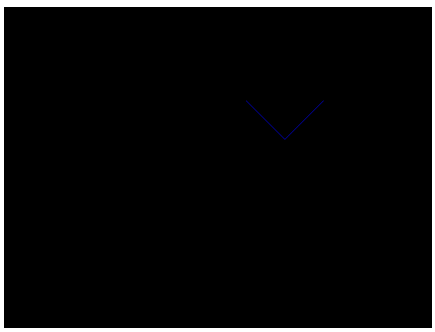
3. view port에 x-offset 값 동작 고찰

```
const renderer = new THREE.WebGLRenderer();
renderer.setSize( 640, 480 );
renderer.setViewport( 320, 0, 640, 480 );

const container = document.getElementById( 'myContainer' );
container.appendChild( renderer.domElement );
```

setViewport는 화면에 보여지는 영역의 위치와 사이즈를 조정하는데, 현재 x값에 화면 크기의 절반인 320을 주었더니 절반만 보여지게 되었다. 이는 도형이 잘려진 것이 아니라, 화면에 보여지는 영역의 x 좌표가 320만큼 이동한 것이다. x와 y의 오프셋 값이 뷰 포트의 왼쪽 하단 모서리 값이라고 생각해도 동일하다.

```
const renderer = new THREE.WebGLRenderer();
renderer.setSize( 640, 480 );
renderer.setViewport( 100, 100, 640, 480 );
```



x와 y에 100의 값을 주었을 때, 각각 왼쪽 하단 모서리에서 100씩 이동한 것을 볼 수 있다.