컴퓨터그래픽스

#중간고사 대체

소프트웨어학과

32183631 이진호

< 목 차 >

GitHub 링크 :

https://github.com/JinHoLeeee/ComputerGraphics_Mid_Project

Ⅰ. 프로그램 설계

프로젝트 목표

Three.js 를 사용하여 3d 모델링을 통해 달을 포함한 태양계를 만든다. 단, 명왕성은 태양계에 더 이상 포함되지 않기 때문에 제외한다.

프로그램 제작 설계

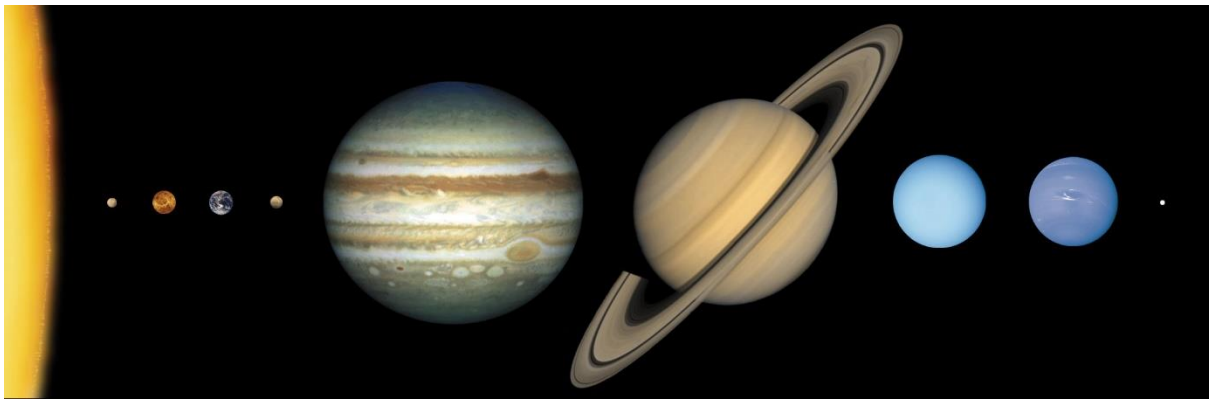
가장 먼저 태양계는 우주에 있기 때문에 scene 의 배경색을 검정색으로 설정하여 우주 공간임을 표현한다.

그 다음 Three.js 에서 제공하는 구 모형인 Sphere Geometry 를 사용하여 scene 에 구를 모델링해본다. 모델링에 성공하면 구 모형을 여러 개 삽입하여 크기와 각 모형 간의 거리를 적당히 조정해본다. 이때 three.js 에서 제공하는 AxisHelper 를 통해 거리 조정을 쉽게 할 수 있다.

다음으로 삽입한 구 모형들에 행성들의 표면을 텍스쳐(material)로 입혀준다. 이때 행성 표면 그림파일을 materia 로 사용하기 위해 three.js 에서 제공하는 MeshPhongMaterial 을 사용한다. 토성과 천왕성 같이 행성에 고리가 필요할 경우 three.js 에서 제공되는 Ring Geometry 를 사용한다.

마지막으로 실제 태양계 행성들의 크기와 거리 비율을 구하여 이를 3d 모델에 적용한다.

아래 그림은 태양계 제작 예상도를 나타낸 것이다. 단, <그림 Ⅰ-1> 에서는 행성 간의 거리 비율을 무시한 것이므로 실제 제작 결과와는 차이가 있다.



<그림 Ⅰ-1> 태양계 제작 예상도

Ⅱ. 제작 과정

소스 코드 설명

```
<script>
    window.onload = function (e){
    var width=window.innerWidth;
    var height=window.innerheight;
    var scene=new THREE.Scene();

    initThree();

    scene.add(new THREE.AmbientLight(0x333333));

    var light=new THREE.DirectionalLight(0xffffff,1);
    light.position.set(5,3,5);
    scene.add(light);
```

<사진 Ⅱ-1> 코드 1

먼저 화면의 크기를 조정하고, three.js 의 scene 을 추가해주었다. 그리고 여러 기능을 담당하는 initThree() 함수를 실행해주었다. 그리고 scene 에 조명을 추가해주었다.

```
var geometry=new THREE.SphereGeometry(1,32,32);//earth radius=1
const loader=new THREE.TextureLoader();
var material=new THREE.MeshPhongMaterial({
map:loader.load('earthmap1k.jpg'),

});

var earthMesh=new THREE.Mesh(geometry,material);
earthMesh.position.z=0;
scene.add(earthMesh);
```

<사진 Ⅱ-2> 코드 2

그리고 구 모델을 new THREE.SphereGeometry 를 이용해 모델링하였다. 이때 첫 번째 매개변수는 구의 반지름 값을 나타내므로 이 모형의 반지름 값은 1 이 된다.

위 <사진 Ⅱ-2> 속 var geometry 는 지구를 담아낼 구의 geometry 를 선언한 것이다. 여기서 material 은 모형의 뼈대 역할을 담당한다. 이 모델에 텍스쳐를 입히기 위해 loader 를 const 로 선언하여 THREE.TextureLoader();를 해주었다. 그리고 지구의 표면 사진을 입히기 위해 material 을 선언하고, 이 material 에 MeshPhongMaterial 을 이용하여 지구 표면 사진인 earthmap1k.jpg 를 로드해주었다. 여기서 material 은 모형의 표면 역할을 담당한다.

다음으로 geomtry 와 material 을 합쳐 지구를 형성해주었다. 이 지구 모형의 위치는 z=0 으로 설정하고 scene 에 추가하였다.

태양과 달, 그리고 다른 행성들의 생성 방식도 위와 같다.

다음 그림은 <사진 Ⅱ-2> 에서 쓰인 지구의 표면 파일이다.

<사진 Ⅱ-3> 지구 표면 그림 파일(jpg)

토성과 천왕성의 고리는 다음과 같은 방법으로 제작하였다.

```
var saturn_ring = new THREE.RingGeometry( 8, 9, 32 );
const saturn_ringloader=new THREE.TextureLoader();
var saturn_ringmaterial = new THREE.MeshPhongMaterial({
map:saturn_ringloader.load('saturn_ring.jpg')
});
var saturn_ringmesh = new THREE.Mesh( saturn_ring, saturn_ringmaterial );
scene.add( saturn_ringmesh );
saturn_ringmesh.position.z=-50;
```

<사진 Ⅱ-4> 토성의 고리

먼저 고리 모양의 모형을 모델링하기 위해 Geometry 를 만들어주었다. 이때 RingGeometry 의 첫 번째 매개변수는 고리의 안쪽 반지름, 두 번째 매개변수는 고리의 바깥쪽 반지름을 의미한다. 이후의 방법은 행성 제작 과정과 같다.

마지막으로 고리 mesh 의 위치를 행성의 위치와 같게 설정하였다.

다음 사진은 처음에 실행하였던 initThree() 함수의 선언 부분이다.

```
function initThree( ) {
        // 브라우저가 WebGL을 지원하는지 체크
        if (WEBGL.isWebGLAvailable( )) {
            console.log('이 브라우저는 WEBGL을 지원합니다.');
        } else {
            console.log('이 브라우저는 WEBGL을 지원하지 않습니다.');
        }

        camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 0.1, 1000);

        let renderer = new THREE.WebGLRenderer({
            antialias: true
        });
        renderer.setSize(window.innerWidth, window.innerHeight);
        renderer.shadowMap.enabled = true;
        renderer.shadowMap.type = THREE.PCFSoftShadowMap;
        document.body.appendChild(renderer.domElement);

        //let axes = new THREE.AxisHelper(10);
        //scene.add(axes);

        camera.position.x = 2;
        camera.position.y = 1;
        camera.position.z = 1;

        controls = new THREE.OrbitControls(camera,renderer.domElement);
        controls.rotateSpeed = 1.0;
        controls.zoomSpeed = 1.2;
        controls.panSpeed = 0.8;
        controls.minDistance = 5;
        controls.maxDistance = 100;

        function animate( ) {
            requestAnimationFrame(animate);

            renderer.render(scene, camera);
            controls.update( );
        }

        animate( );
    }

}
script>
```

<사진 Ⅱ-5> initThree() 함수 선언 부분

initThree()함수가 제공하는 기능은 크게 5 가지이다.

① WebGL 지원 여부 체크

WEBGL.isWebGLAvailable()을 통해 사용하는 브라우저의 WebGL을 지원 여부를 콘솔로 띄워준다.

② Camera 설정

Three.js 에서 제공하는 PerspectiveCamera 를 통해 camera 를 생성하고, 이후에 카메라의 위치를 camera.position 을 통해 설정한다.

③ 렌더링

WEBGLrenderer 를 통해 새 render 를 구현한다. 그리고 사이즈와 shadowMap 을 설정하고, 이를 domElement 에 append 시켜준다.

④ 마우스 드래그를 통한 시점 이동 지원

OrbitControls.js 파일을 이용하여 사용자가 마우스를 드래그하여 자유자재로 scene 내의 시점을 이동하도록 도와주는 부분이다. controls 변수에 three.js 의 orbitcontrols 를 담아주고, 이 controls 변수를 통해 세부 조종 기능을 설정해주었다.

⑤ Animate() 함수 선언과 실행

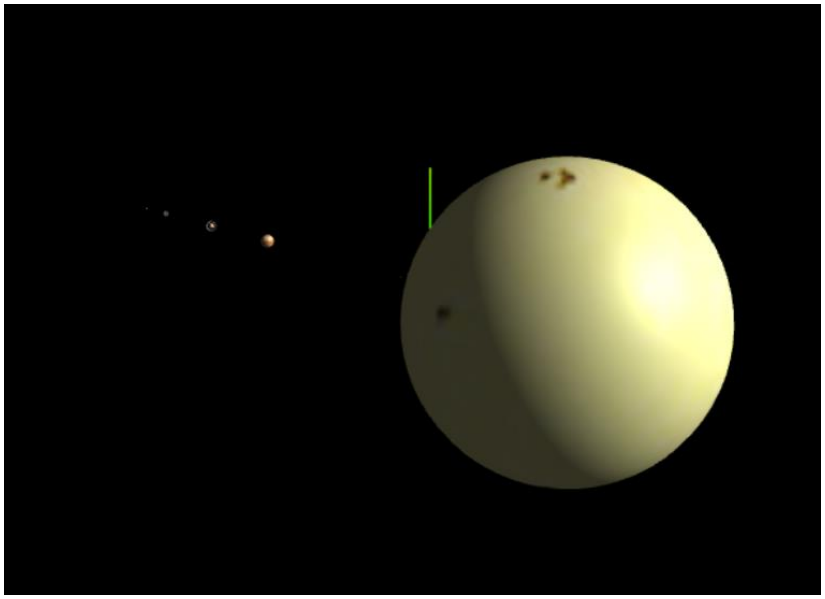
Animation 함수를 통해 renderer 의 렌더함수를 실행하고 위에서 선언했던 변수 controls 에 update()함수를 실행하여 orbitcontrol 이 원활하게 작동하도록 해주었다.

제작 과정

처음 태양계를 구성해본 뒤, 실제 태양계 행성들의 크기와 거리 비율을 적용하여 태양계를 보다 현실감있게 구현하고자 하였다.

수성	태양의 지름x0.0038배	40m
금성	태양의 지름x0.01배	70m
지구	태양의 지름x0.01배	100m
화성	태양의 지름x0.005배	152m
목성	태양의 지름x0.11배	500m
토성	태양의 지름x0.066배	1000m
천왕성	태양의 지름x0.038배	2000m
해왕성	태양의 지름x0.042배	3000m

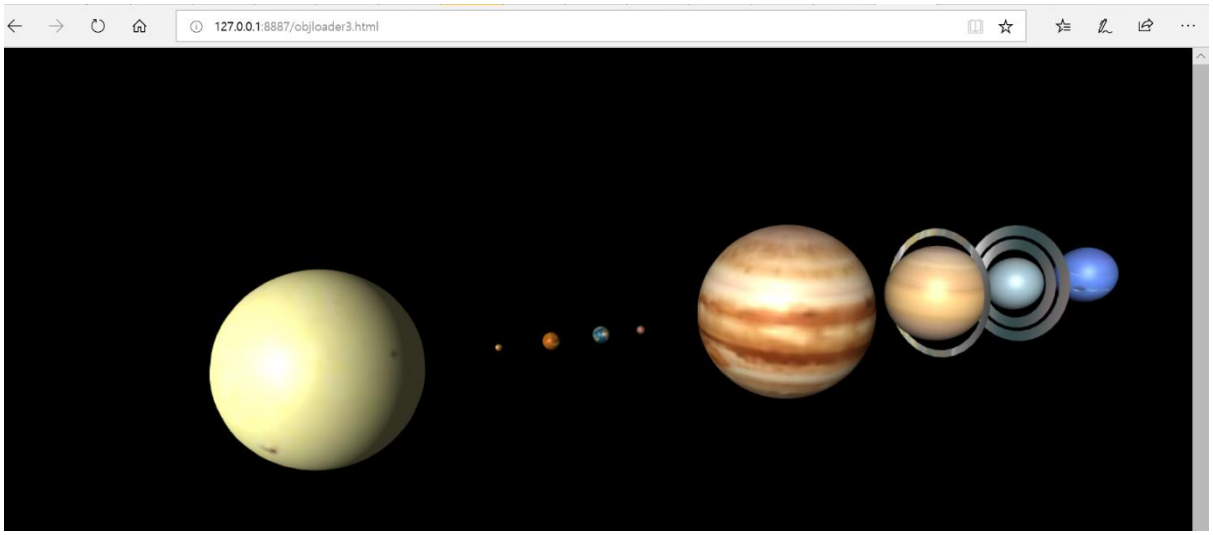
위 표는 태양의 지름 대비 각 행성의 크기 비율과 태양과 지구 사이의 거리가 100m 라고 가정할 때 다른 행성과 태양 사이의 거리를 나타낸 것이다. 초기에는 위 표의 비율을 그대로 적용하여 다음 사진과 같이 구현해보았다.



<사진 Ⅱ-6> 초기 구현 모습

실제 비율을 적용하자 아쉽게도 태양의 크기만 눈에 띄고, 다른 행성은 너무 작아 보이지 않았다. 또, 실제 거리가 워낙 넓은 탓인지 뒤의 목성부터는 화면에 담기지 않았다.

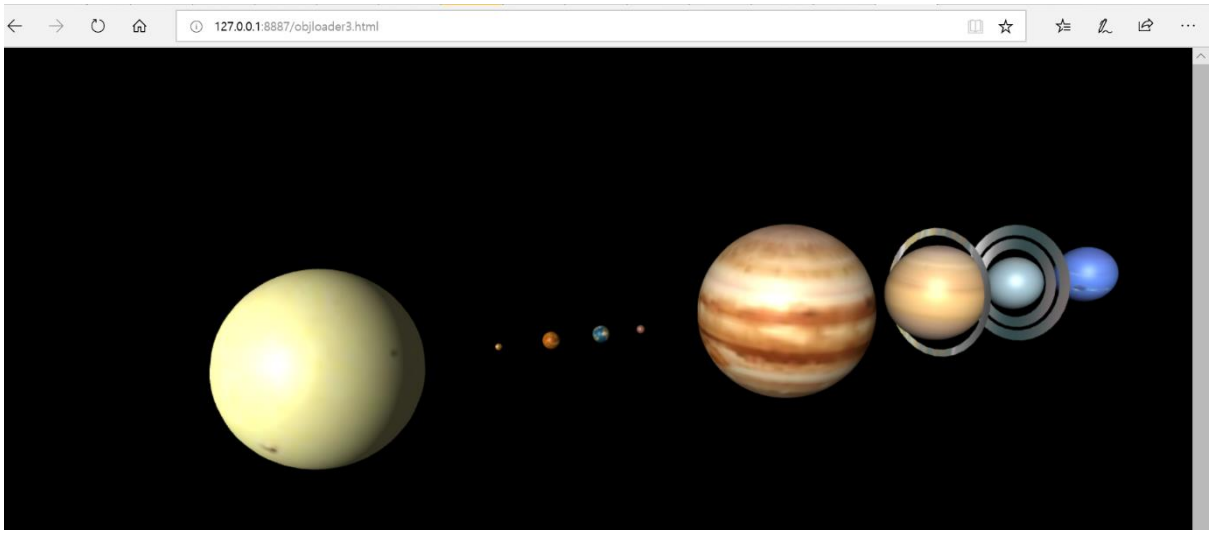
따라서 사용자가 행성의 크기를 한 눈에 가늠할 수 있게끔 행성들의 크기 비율을 그대로 유지하되 반지름을 10 배씩 확대하고, 행성 간 거리가 너무 멀 경우 그 거리를 약 1.5 배 정도씩 좁혀주었다. 그 결과는 다음 사진과 같다.

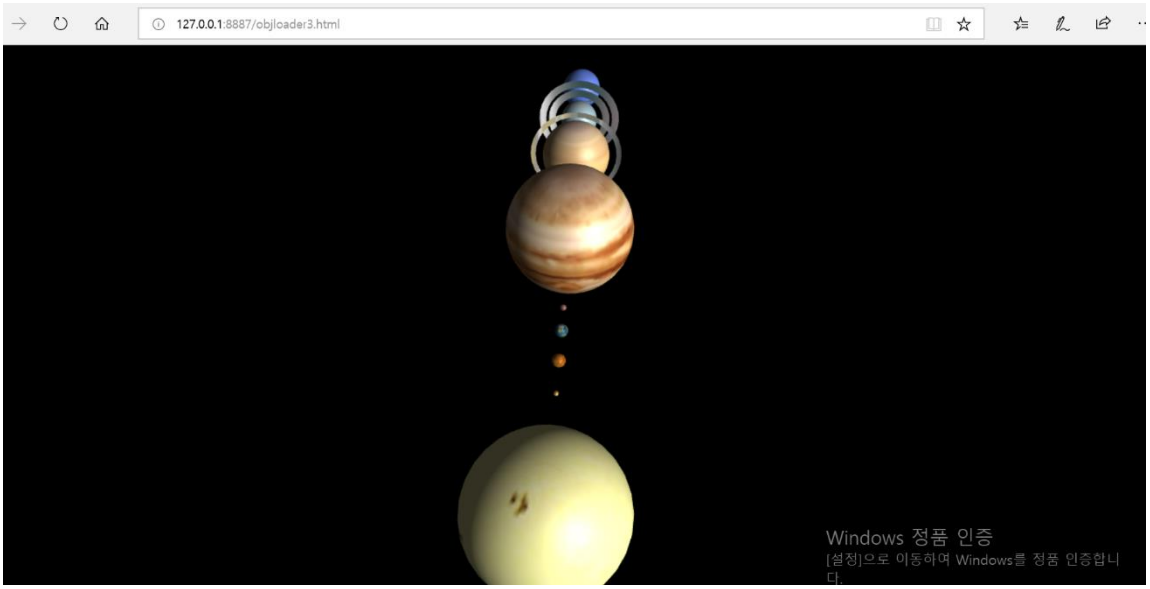


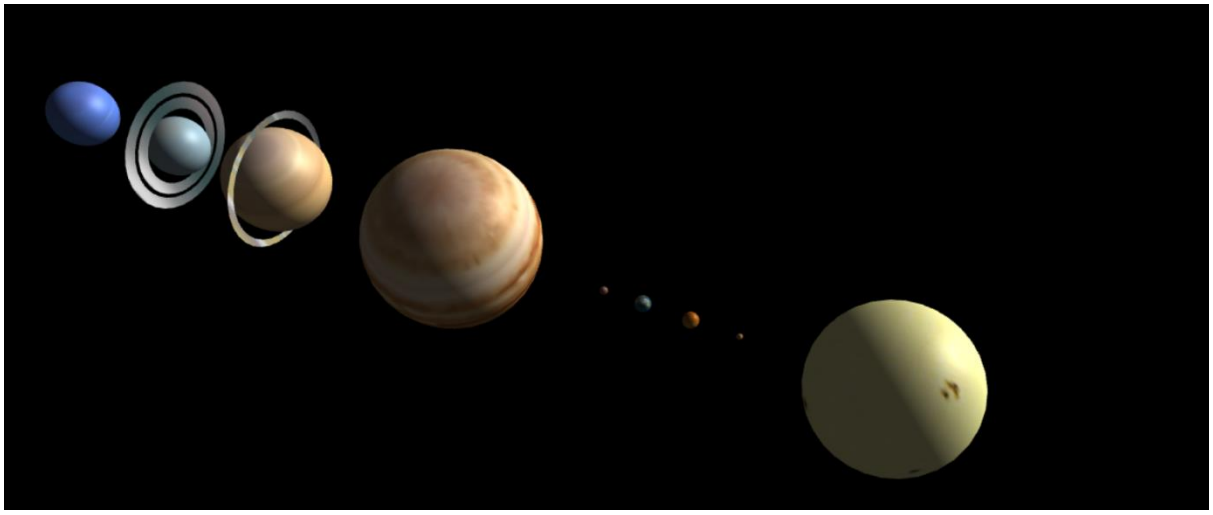
<사진 Ⅱ-7> 수정 후 구현 모습

이전과 다르게 태양계 전체를 화면에 담을 수 있게 되었다. 또한 사용자는 마우스 드래그를 통해 태양계를 자유자재로 구경할 수 있고, 마우스 스크롤을 위아래로 움직여 화면을 확대/축소하며 행성을 더욱 자세하게 들여다볼 수 있다.

Ⅲ. 제작 결과


127.0.0.1:8887/objloader3.html


127.0.0.1:8887/objloader3.html
Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증합니다.



Github 링크 :
https://github.com/JinHoLeeee/ComputerGraphics_Mid_Project