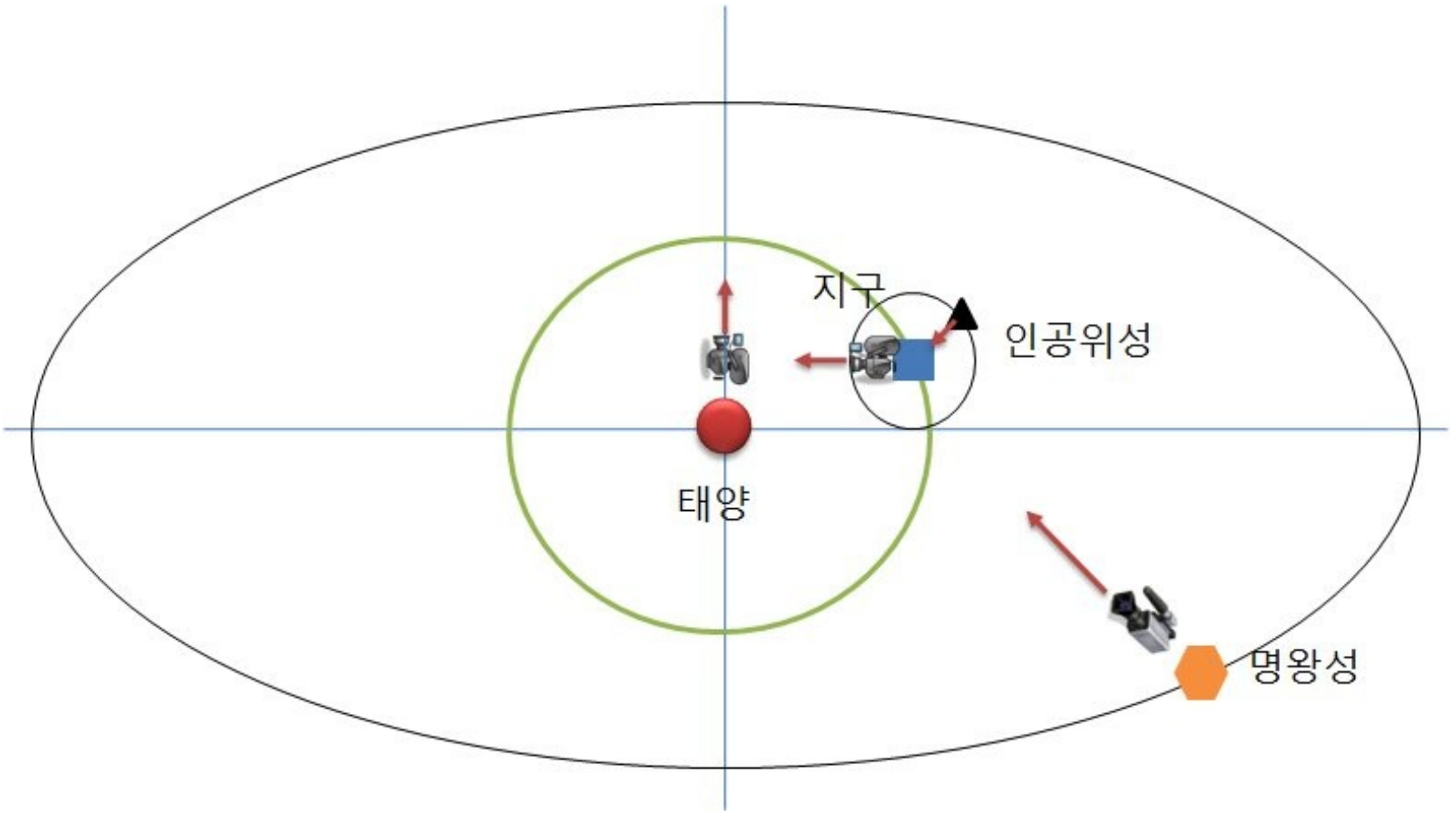
2016년 2학기
그래픽스 과제2 보고서

200721395 한경수

Table Of Contents

개요

간단한 태양계를 시뮬레이션 해보는 과제다.



그림과 같이 태양, 지구, 지구위성, 명왕성을 다양한 위치에서 볼 수 있도록 뷰포트를 4개를 만들어서 수행한다.

모양에 대해선

태양 : 구 지구 : 정사각형 위성 : 사면체 명왕성 : 팔면체

구현 요구사항은 7가지 정도로 압축 될 수 있다. 다음과 같다.

- 그 상대적인 위치는 다음과 같다. 초기위치는 태양, 지구, 인공위성, 명왕서이 태양의 오른쪽으로 정렬된 상태로 시작된다. 인공위성은 처음에는 수평면과 평행하게 회전한다 한국은 시작할 때 바로 태양을 보는 면에 있다고 가정한다.
- 태양에서는 12시 방향으로만 본다. 즉 카메라가 12시 방향으로 설정되어야 한다. 그리고 지구는 한국 땅에서 수직으로 밖으로 본다. 명왕성은 항상 지구 중심을 본다. 인공위성도 지구의 중심을 보면서 돈다.
- 하면 1/4(1사분면) 은 태양에서 보는 장면, 4/4(제4사분면) 는 지구에서 보는 장면, 3/4(3 사분면) 은 명왕성에서 보는 장면, 2/4(2사분면) 에는 전체를 조망하는 장면이 나타나야 한다. 보는 시점은 적절한 거리의 (w,w,w) 위치에서 (0,0,0) 을 보는 것이다.
- 지구의 하루는 지구가 한바퀴 자전하는 날이다. 그리고 지구의 일년은 4일이다. 즉 태양을 한번 도는데 지구는 4번의 자전을 한다. 명왕성의 1년은 지구의 4년과 동일하다. 명왕성의 일년은 명왕성의 8일마다 명왕성 1년이 돌아온다. (1번 공전할 때 마다 8번 자전한다.) 단 명왕성과 지구의 회전축은 같은 높이다.
- 인공위성은 매번 한 번씩 공전할 때마다 30도씩 기울기를 높이면서 지구를 회전한다. 6 바퀴를 돌면 다시 원래 기울기로 회복된다.(실제 인공위성과 동일한 방법으로 돈다.)
- 지구의 세로축은 수직에서 15도 기울어져 있다.
- 인공위성의 처음 위치는 적도와 평행한 평면상에 12시 방향에서 시작한다. 명왕성은 3 시 방향에서 시작한다.

구현

각 뷰포트에 대한 구현 결과다.

모델 변환에 대해서

먼저 움직임을 어떻게 정의할것인가 고민해야 했다.

먼저 각 행성마다 자전의 각속도가 있을것이고, 각 행성의 공전의 속도는 자전의 배수로 설정하기로 했다.

코드로 표현하면 다음과 같다.

```
sun.jaJeonRad += 1;
earth.jaJeonRad += 4; // earth.userRad * 4;

myungWang.userRad += 2/8;
myungWang.x = 11 * Math.cos(-myungWang.userRad * 3.14 / 180);
myungWang.z = 6 * Math.sin(-myungWang.userRad * 3.14 / 180);
myungWang.jaJeonRad = myungWang.userRad * 8;
satel.jaJeonRad += 2;
satel.scale = 0.3;

```

코드에서 Rad 라고 되어있는데, 의미는 degree 다.

지구와 위성은 자전각속도만 잡아주면 적절한 transform 을 이용해서 좌표가 세팅된다.

명왕성의 경우엔 타원의 형태로 돌고 타원의 방정식을 쓰라고 되어있어서 paramatic form 인

```
y = b * sin(t),
x = a * cos(t)
```

를 이용하여 좌표를 구했다.

그리고 각 분면마다 drawScene 이 호출되기 전에 projection 을 잡아준다.

```
// sun.js
function Sun(){ };
Sun.prototype = new Tetra();
Sun.prototype.draw = function(lookAt){
  modelViewMatrix = mat4();
  modelViewMatrix = mult(modelViewMatrix, translate(this.x, this.y, this.z));
  modelViewMatrix = mult(modelViewMatrix, rotate(this.jaJeonRad, [0, 1, 0]));
  modelViewMatrix = mult(modelViewMatrix, scale3(this.scale,this.scale,this.scale));
  normalMatrix = [
    vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0][2]),
    vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1][2]),
    vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2][2])
  ];

  uLookAtMatrix = lookAt;
  ...
  ...
}



function drawScene(lookAt){
  sun.draw(lookAt);
  earth.draw(lookAt);
  satel.draw(lookAt, earth);
  myungWang.draw(lookAt);
}
// 2분면
projectionMatrix = perspective(50, gl.viewportWidth / gl.viewportHeight, 0.5, 100.0);
projectionMatrix = mult(projectionMatrix, scale3(0.3, 0.3, 0.3));
gl.viewport(0, gl.viewportHeight / 2, gl.viewportWidth / 2, g
```

```
l.viewportHeight / 2);
eye = vec3(0, 6, eyeDistance);
drawScene(lookAt(eye, at, up));
```

2분면에 대한 코드다. sun, earth, satel, myungWang 의 draw 함수로 lookAt 변수를 넘겨줌으로써 각 뷰에 대한 camera 를 세팅할 수 있다.

각 행성은 buffer 만들어주는 클래스를 상속받아서 draw function 을 overriding 하는 방식으로 구현하였다.

가장 복잡한 부분인 지구와 위성의 transform 을 살펴보도록 하겠다.

```
function Earth(){ };
Earth.prototype = new Cube();
Earth.prototype.days = 1;
Earth.prototype.draw = function(lookAt){
  var vNormal = gl.getAttribLocation( program, "vNormal" );
  gl.vertexAttribPointer( vNormal, 4, gl.FLOAT, false, 0, 0 );
  gl.enableVertexAttribArray( vNormal);

  var vPosition = gl.getAttribLocation( program, "vPosition");
  gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
  gl.enableVertexAttribArray(vPosition);



  var T = mat4(); // [1]
  T = mult(T, rotate(this.jaJeonRad / 4, [0, 1, 0]));
  T = mult(T, translate(5, 0, 0));
  T = mult(T, scale3(this.scale,this.scale,this.scale));
  T = mult(T, rotate(this.jaJeonRad, [Math.cos(75 * 3.14 / 180),
Math.sin(75 * 3.14 / 180), 0]));

  var coord = mult(T, vec4(0, 0, 0, 1));
  this.x = coord[0];
  this.y = coord[1];
  this.z = coord[2];
  modelViewMatrix = T;

  uLookAtMatrix = lookAt;
  ...
}


function Satel(){ };
Satel.prototype = new Tetra();
Satel.prototype.slope = 90;
Satel.prototype.draw = function(lookAt, earth){
  // 180 -> 30 / 180 => 1/6 도씩
  modelViewMatrix = mat4();
  modelViewMatrix = mult(modelViewMatrix, translate(earth.x, eart
```

```
h.y, earth.z)); // [2]

  this.slope += 1/6;
  if(this.slope >= 360){
    this.slope -= 360;
  }
  modelViewMatrix = mult(modelViewMatrix, rotate(this.jaJeonRad,
[0, 1, 0]));
  modelViewMatrix = mult(modelViewMatrix, translate(2, 0, 0));
  modelViewMatrix = mult(modelViewMatrix, rotate(this.jaJeonRad,
[0, 1, 0]));
  modelViewMatrix = mult(modelViewMatrix, scale3(this.scale,this.
scale,this.scale));
  normalMatrix = [
    vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewM
atrix[0][2]),
    vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewM
atrix[1][2]),
    vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewM
atrix[2][2])
  ];

  uLookAtMatrix = lookAt;
  ...
  ...
}
```

[1] 부분은 model 변환에 대한 코드인데 하나식 살펴보면

처음에 T 에 단위 행렬을 넣고, 코드에서는 거꾸로 작성 되었지만 개념상으론 rotate - scale3 - translate - rotate 순으로 transform 되는 행렬이 생성된다.

먼저 제자리 회전시키고, 크기조절을 하고, 처음으로 자리잡히는 좌표인 5를 가져온다. 그리고 공전 transform 을 마지막으로 적용시킨다.

이 과제를 하면서 곤란했던 부분은 위성의 좌표를 잡는것이었는데, 이와 같이 lookAt matrix 를 shader 에서 따로 처리하게끔 하니까 이 문제를 해결할 수 있었다.

지구 좌표를 구하기 위해서 coord(0, 0, 0) 좌표에 대해 이 변환 행렬을 적용시키면 지구의 좌표가 나오게 된다. 이 좌표는 earth.xyz 로 들어가게 된다.

[2] 는 인공위성에 대한 코드인데, 일단 최대한 간단히 구현하기 위해서 인공 위성의 주인을 태양으로 잡았다. 그리고 잘 돌아가는지 확인한 후에

마지막에 modelViewMatrix = mult(modelViewMatrix, translate(earth.x, earth.y, earth.z)); 코드로 지구 좌표로 옮겨주었다.

그렇게 하니까 인공위성이 지구의 인공위성으로 바뀌는것을 볼 수 있었다.

뷰 변환에 대해서

과제에서 요구하는 것 중 하나가 4개의 뷰포트를 쓰면서 원하는 카메라위치에서 뷰를 할 수 있는지

물어보는 과제다.

```
// 2분면
projectionMatrix = perspective(50, gl.viewportWidth / gl.view
portHeight, 0.5, 100.0);
projectionMatrix = mult(projectionMatrix, scale3(0.3, 0.3, 0.
3));
gl.viewport(0, gl.viewportHeight / 2, gl.viewportWidth / 2, g
l.viewportHeight / 2);
eye = vec3(0, 6, eyeDistance);
drawScene(lookAt(eye, at, up));

projectionMatrix = perspective(50, gl.viewportWidth / gl.view
portHeight, 0.5, 100.0);
projectionMatrix = mult(projectionMatrix, scale3(0.3, 0.3, 0.
3));

// 1분면
gl.viewport(gl.viewportWidth / 2, gl.viewportHeight / 2, gl.v
iewportWidth / 2, gl.viewportHeight / 2);
drawScene(lookAt(vec3(0, 0, 0), vec3(0, 0, -1), vec3(0, 1, 0)
));

// 3분면 : 명왕성에서 지구 보기
gl.viewport(0, 0, gl.viewportWidth / 2, gl.viewportHeight / 2
);
drawScene(lookAt(vec3(myungWang.x, myungWang.y, myungWang.z),

      vec3(earth.x, earth.y, earth.z), vec3(0, 1, 0)));


// 4분면 : 지구에서 수직면으로 보기, h 로 인공위성에서 지구 보는것
토글.
gl.viewport(gl.viewportWidth / 2, 0, gl.viewportWidth / 2, gl
.viewportHeight / 2);
if(viewToggle){
  projectionMatrix = perspective(60, gl.viewportWidth / gl.vi
ewportHeight, 0.5, 100.0);
  projectionMatrix = mult(projectionMatrix, scale3(0.3, 0.3,
0.3));
  var v = mult(rotate(earth.jaJeonRad, [Math.cos(75 * 3.14 /
180), Math.sin(75 * 3.14 / 180), 0]), vec4(earth.x, earth.y,
earth.z, 1));
  //drawScene(lookAt(vec3(earth.x, earth.y, earth.z), vec3(su
btract(vec4(earth.x, earth.y, earth.z, 1), v)), vec3(0, 1, 0)
));
  drawScene(lookAt(vec3(satel.x, satel.y, satel.z), vec3(eart
h.x, earth.y, earth.z), vec3(0, 1, 0)));
}
else{
  var v = mult(rotate(earth.jaJeonRad, [Math.cos(75 * 3.14 /
180), Math.sin(75 * 3.14 / 180), 0]), vec4(earth.x, earth.y,
```

```
earth.z, 1));
  drawScene(lookAt(vec3(earth.x, earth.y, earth.z), vec3(subt
ract(vec4(earth.x, earth.y, earth.z, 1), v)), vec3(0, 1, 0)))
;
}
```

최초 구현시에 draw 함수가 lookAt 을 각자 처리할 수 있는 구조로 잡아놔서 이는 크게 어렵지 않았다.

요구사항에 따라 카메라 좌표를 설정하고, 보는 위치가 어딘지 알아내는것은 약간의 수학적 표현만 해주면 되는 문제였다.

1분면은 태양의 위치 (0, 0, 0) 에서 12 시 방향(0, 0, -1) 으로 뷰를 세팅했다.

2분면은 적당한 거리에서 적당한 perspective 값을 주어서 뷰를 만들었다.

3분면은 타원의 방정식으로 구한 명왕성의 좌표와 지구의 좌표만 있으면 해결되는 문제라서 그냥 해당하는 값을 넣었다.

4분면은 카메라의 위치는 쉽게 잡을 수 있었는데, at 을 구하는데 조금 생각을 했다. 자전축에 누워있는데다가 회전에 따라 바뀌는 문제가 있었다.

일단은 자전축이 누워있는 문제는 제끼고 수평일 때의 처리를 강구했다. 일단은 earth.jaJeonRad 만큼 [0, 1, 0] 축에 대해서 회전하는것처럼 at 이 이동하는것을 상상 할 수 있다.

그러면 mult(rotate(earth.jaJeonRad, [0, 1, 0])) earth 좌표에 대해서 적용시키면 보는 방향이 정해지고, 여기서 카메라 좌표를 빼면 볼 좌표를 구할 수 있다.

유저 키보드 단축키 처리

'+' 키를 어떻게 처리해야할지 몰라서 zoom in/out 키는 -. = 키로 대체하여 처리했다.

그리고 h 와 t 키도 처리를 했다. 브라우저로 열어보면 정상적으로 처리되는 모습을 볼 수 있다.

```
function on_key_down(){
  console.log(event.keyCode);
  if(event.keyCode == 189) {
    eyeDistance = Math.min(eyeDistance + 1, 30);
  }
  if(event.keyCode == 187){
    eyeDistance = Math.max(eyeDistance - 1, 3);
  }
  if(event.keyCode == 72){
    isRun = !isRun;
  }
  if(event.keyCode == 81){
    window.close();
  }
  if(event.keyCode == 84){
    viewToggle = !viewToggle;
  }
```

```
18 }
19
```

on*key*down 를 통해 유저의 키보드처리를 하게 했다. 이 변수는 main.js 에 정의되어있고, 이 변수에 따라 적절한 lookAt 을 호출하게 했다.

결과

이 프로젝트의 핵심은 원하는 변환을 할 수 있느냐, 그리고 원하는 뷰를 통해 최종적으로 frame buffer 에 그래픽을 찍어낼 수 있는지 배우는 것이었다.

그리고 기존의 gl 과는 다르게 shader 를 어떻게 uniform 변수를 쪼갤 것인지 공부하는데 아주 도움이 되었다. 예전부터 GLSL 는 그래픽스 테크닉의 어려운 부분이었는데, 초장부터 shader 없이는

그래픽 프로그래밍을 못하게 설계된 es 라서 많이 당황했다. 하지만 es 의 compact 함의 개념을 숙지하고 잘 쓰게 되면 소형화된 스펙이지만 안되는것은 없는것을 알 수 있었다.

요약하면 es 는 gl_Position = uPMatrix *uLookAtMatrix * uMVMatrix * vPosition 를 설정하고, 그 해당 vertex 에 대해 fragment 를 처리하는 것으로 줄일 수 있겠다.

또한, 이 과제에서 은근히 시간을 많이 까먹은 부분중 하나가 vertex buffer 설정하는것을 정리하는 것이었다. 실제로 다양한 모델링 파일들은 다른 3d 프로그램에서 export 된 결과를 쓰게 되는데

이러한 부분을 추가적으로 학습해서 가져놀면 재밌을것 같다. 거기다가 최근에 배운 texture 도 입혀보고 싶다.

전체 소스

전체 소스는 MV.js, cube.js, earth.js, iniShaders.js, myung.js, pyramid.js, satel.js, sun.js, tetra.js, util.js, webgl-utils.js 로 구성되어있으며

제출된 압축파일에 제공 되어있다.