컴퓨터학부 20162460 김종렬

1. Solar System 개요

Solar System은 Opengl을 이용하여 만든 가상의 태양계 시스템이다.

마우스를 활용한 trackball을 이용하여 자유로운 각도에서 행성의 자전, 공전을 관찰할 수 있다.

2. 구현 목적

Solar System은 2022년 1학기 컴퓨터 그래픽스 과목의 5번째 과제를 목적으로 제작되었다.

3. 환경

Solar System은 다음과 같은 환경에서 개발되었다.

- Windows10
- Visual studio 2019
- freeglut 3.0.0 MSVC Package
- GLEW 2.1.0, GLM 0.9.9.8
-

4. 구현 사항

Solar System에 구현된 사항은 다음과 같다.

1. Model transformation

   ➔ 행성의 자전, 공전 구현

2. View transformation

   ➔ Scene의 원점을 기준으로 Trackball 회전 구현

3. Shading

   ➔ fragment shader에서 Phong shading 구현

4. Texture Mapping

   ➔ Vertex의 texture coordinate를 이용하여 구현

5. 구현 상세

1. Model transformation

행성 및 항성의 자전, 공전을 표현하기 위해 translate, rotate, scale 변환 Matrix를 활용하였다.

씬 내의 object 모양이 구 형태로 동일하므로, Vertices + texture Coordinates buffer를 한 번만 만들어 놓고 매 프레임마다 하나의 object를 이동, 변형, 회전시켜서 Scene을 구성하였다.

예시로 달의 경우 지구의 공전과 달의 공전, 달의 자전을 표현할 때 다음과 같은 과정을 거친다.

(시작은 항상 원점 좌표에 있는 반지름 1인 구에서 시작한다.) 달의 크기 조절 → 달의 자전 적용 → 달의 공전, 지구의 공전에 의한 자전 제거 → 지구 ~ 달만큼의 거리 이동 → 달의 공전 적용 → 태양 ~ 지구만큼의 거리 이동 → 지구의 공전 적용

유념할 점은 자전의 경우 원점에서 회전시켜야 하고, 공전의 경우 중심이 되는 물체와의 거리만큼 이동시킨 상태에서 회전시켜야 한다.

또한, 공전에 의해 한바퀴 돌 때마다 자전을 한 번 더하게 되는 현상이 나타나는데, 이를 방지하기 위해 공전 각도의 반대방향으로 자전을 적용시켜 주었다.

2. View transformation

Scene을 돌리기 위해 다음과 같이 두가지 방법을 생각해 보았다.

- 카메라를 회전시키기

- Scene 내의 모든 Object를 회전시키기

gluLookAt을 활용하여 첫번째 방법을 시도하여 보았지만, 매번 바뀌게 되는 up 벡터 등의 문제로 두번째 방법으로 구현하게 되었다.

씬 내의 모든 Object는 하나의 Object를 변형, 이동, 회전하여 사용하므로 전부 동일한 Shader에 의해 렌더링 된다. rquat 행렬은 씬 내의 모든 vertex에 동일하게 사용되므로 Program 측에서 연산을 수행하여 렌더링 전에 Shader에 rquat를 전송한다. 실질적인 View transformation은 Vertex shader 내부에서 vertex에 Model transformation을 적용하여 vertex가 제 자리에 위치한 뒤에 적용된다. 구현 방법은 교재에 나와 있는 trackball 예제를 참고하였다.

화면에 반구 형태의 면이 있다고 가정하여 마우스로 드래그 하였을 때 반구의 어느 위치에서 어느 위치로 드래그 되었는지, 해당 드래그 방향의 회전축은 어떻게 되는지 등을 연산하여 Quaternion 값을 찾아내게 된다.

3. Shading

씬 내에 보여지는 모든 오브젝트는 동일한 Shading을 적용하였다.

fragment shader 내에서 Phong shading을 구현하였다.

vertex shader에서는 아래의 값들을 연산 한다.

- 해당 vertex로부터 광원(태양)까지의 거리 (fL)

  ➔ vertex에 model transformation, trackball 적용한 값 자체. 이유는 태양이 원점에 있음.

- vertex에서 카메라까지의 거리 (fE)

  ➔ vertex에 model transformation, trackball 적용한 값에 카메라 거리까지 적용 후 방향 반전

- vertex에서의 normal 값 (fN)

  ➔ 구는 해당 vertex에서 중심 좌표를 빼면 normal을 구할 수 있으므로 vertex에 model transformation, trackball 적용한 값 – 원점 좌표(0,0,0)에 model transformation, trackball을 적용한 값

fragment shader 내에서는 vertex shader에서 구한 fL, fE, fN 값을 이용하여 퐁 셰이딩 연산을 진행한다. fL과 fE를 이용한 half vector로 구현하였다. diffuse, specular, ambient 고유 세팅 값은 fragment shader 내에서 임의 상수 값으로 지정하였다.

4. Texture Mapping

Solar System은 Mesh를 불러오기 위한 별도의 오브젝트 파일을 사용하지 않는다.

런타임 시에 구 형태의 mesh와 Texture coordinate를 동적으로 생성한다. [1]

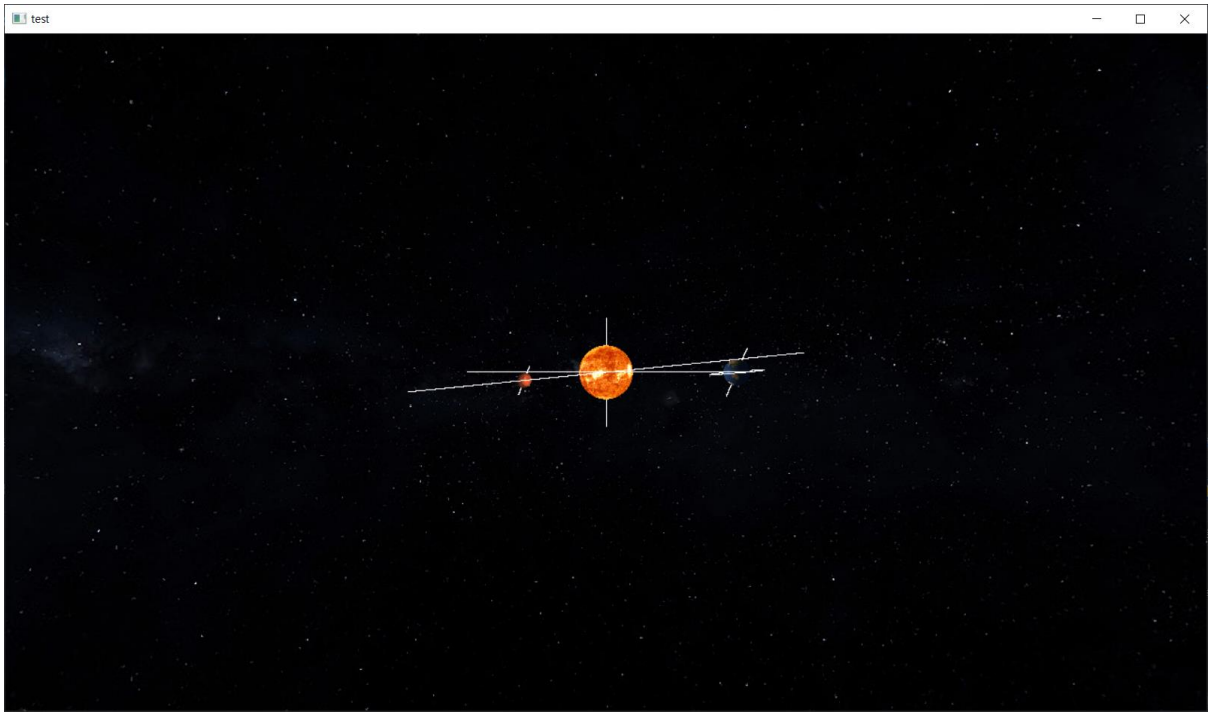
이후에 생성된 vertex와 Texture coordinate를 GPU로 복사한다.

Texture 파일의 경우 불러올 때 stb_image.h 헤더파일을 Github에서 별도로 받아서 사용하였다. [3]

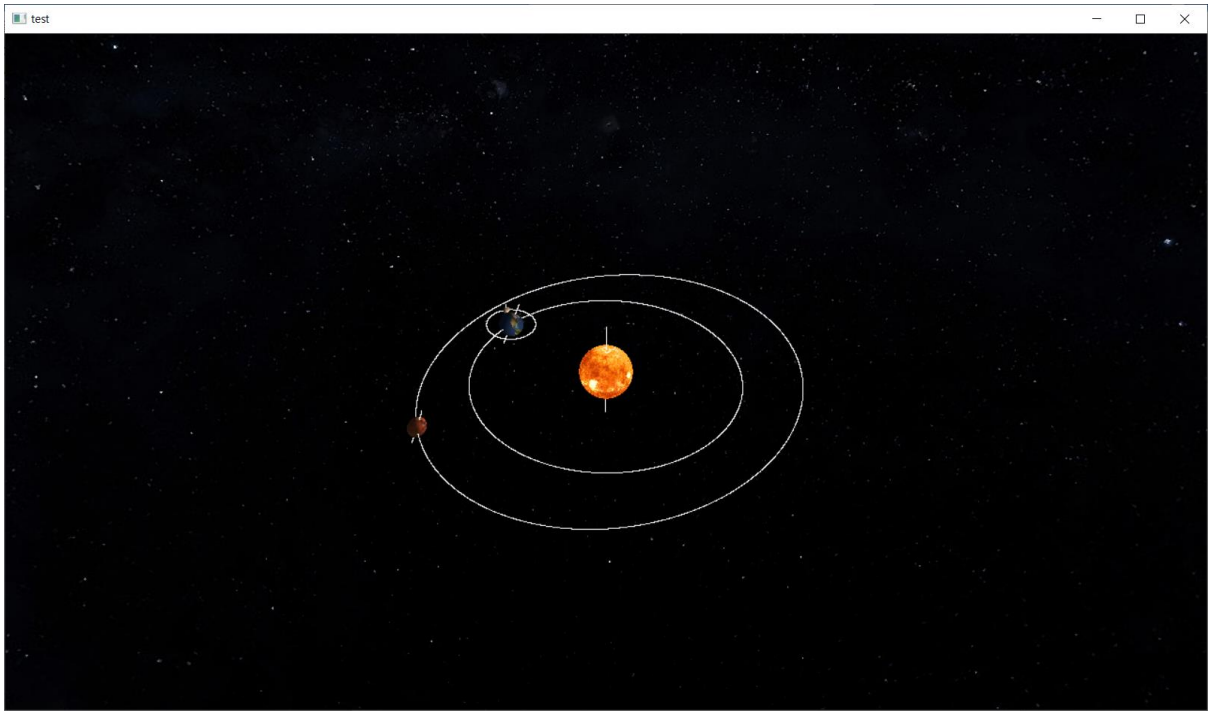
Vertex와는 다르게 Texture는 오브젝트 렌더링 시에 해당하는 Texture를 불러와야 하므로 GPU에서 제공하는 GL_TEXTURE 저장공간에 미리 저장해 두고, 오브젝트 각각 렌더링 시점에서

glActiveTexture, glBindTexture, glUniform1i를 순서대로 호출하여 미리 저장해둔 자신의 텍스처를 fragment shader의 texture 변수에 연결시켜주었다.

5. 실행 결과 및 사용 방법 설명

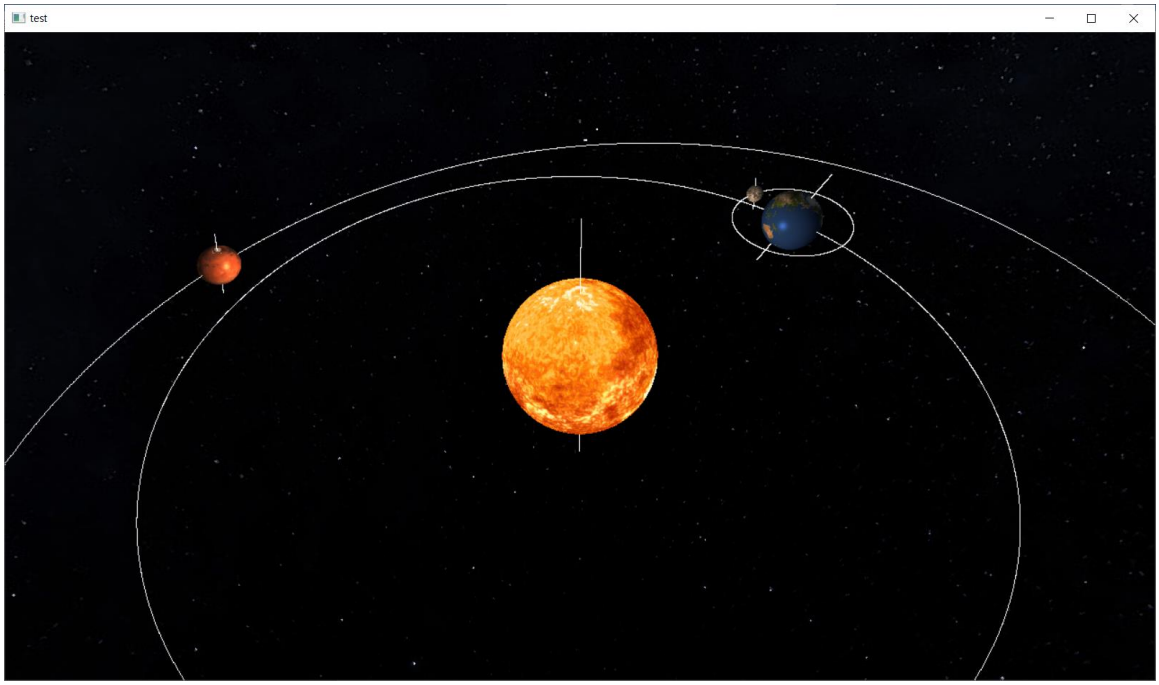


프로그램을 처음 실행시키면 다음과 같은 화면이 나오게 된다.



화면의 임의의 위치에서 마우스 왼쪽 클릭을 한 후에 원하는 방향으로 드래그 하면

Track ball이 작동하여 씬 전체가 태양을 중심으로 회전이 된다.

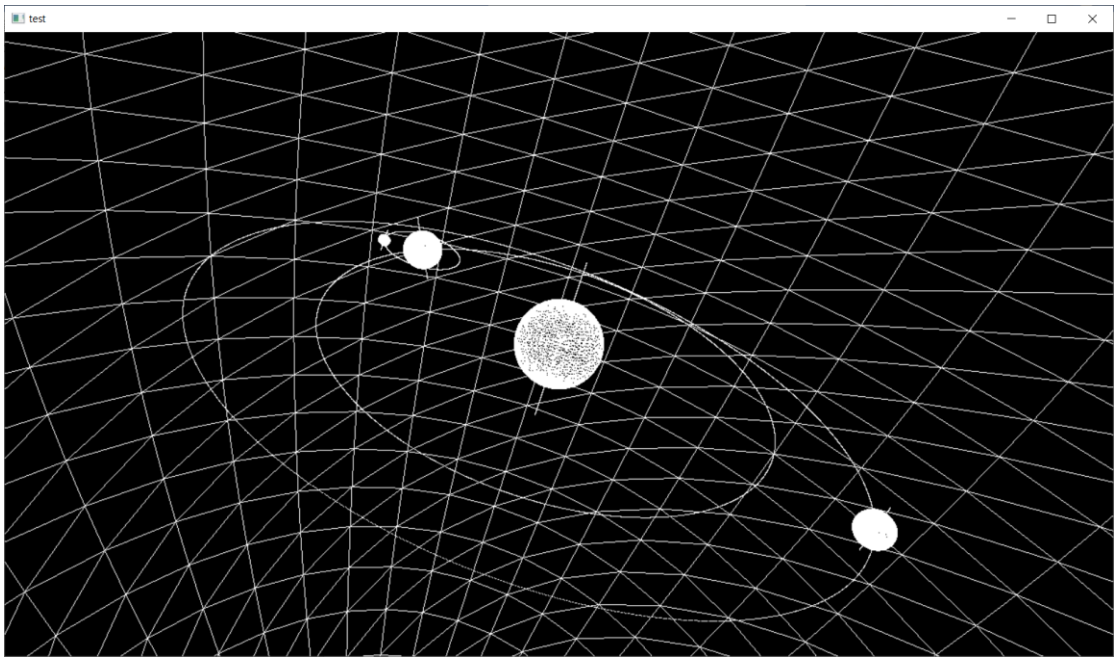


마우스의 가운데 휠을 위로 돌리면 확대가 된다. 오브젝트 들을 더 자세히 볼 수 있다.

Track ball이 작동하여 씬 전체가 태양을 중심으로 회전이 된다.

씬은 태양, 지구, 달, 화성, 그리고 배경 담당 구체 총 5가지의 오브젝트로 구성되어 있다.

Model Transformation, Viewer Transformation, Texture Mapping, Phong Shading이 적절하게 적용된 모습을 볼 수 있다.



추가적으로 마우스 우클릭을 누르고 있으면 Texture와 Shading이 제거된 wire 모습을 볼 수 있다.

구 형태의 Mesh가 적절히 생성된 모습을 볼 수 있다.

참고

[1] Sphere Mesh 생성 알고리즘

http://www.songho.ca/opengl/gl_sphere.html
https://gist.github.com/zwzmzd/0195733fa1210346b00d

[2] Texture 출처

https://www.solarsystemscope.com/textures/

[3] 이미지 쉽게 불러오기 위한 유틸리티 헤더파일

https://github.com/nothings/stb/blob/master/stb_image.h