Project report for CG 100433 course

Project Title

Simple Three-Body Motion Simulator.

Team member

Guo Zhanyu(1851170).

Abstract

A Simple Three-Body Motion Simulator was developed out of my interest and my aim at enhancing my knowledge on CG(Computer Graphics). The goals of this project can be summarized as follows: to build the visual model of the bodies and the physical model of the whole system, i.e., the calculation of the multiple bodies' motion. In the project, users can view the motion of multiple bodies, and some typical body motions are preset, also, which can be generated randomly. The numerical solutions of the system state are obtained by iterative method and various CG techniques are involved, such as color, texture, light, light map, sky box, etc.

Motivation

After reading novels called Three-Body by Liu Cixin, I gained great interest in the Three-Body Problem. In these novels, it is said that the analytical solutions of Three-Body Problem do not exist, because only an extremely small error of the initial condition will bring chaos effect to subsequent solutions. However, through the iterative steps of calculation, observation, and re-calculation, accurate solutions can be obtained, which are called numerical solutions. I have been hoping to calculate and visualize the three-body motion via programs since then. The CG course project is a rare opportunity I have been looking forward to.

After making an initial plan, I found that I can develop the project iteratively from the shallower to the deeper about the Three-Body Problem, i.e., develop from a single body, double bodies to multiple bodies. In addition, the physics of the simple Three-Body Problem, which only considers the attraction between bodies, are within my reach.

Moreover, I can develop it from the shallower to the deeper in CG techniques aspect: from shaders, light and light maps to textures, shadows and ray tracing, etc. Many CG techniques can be applied to the project, which enhances my understanding of the theoretical knowledge of CG and makes me proficiently use OpenGL.

The Goal of the project

To develop a system visualizing the three-body motion, the project's goals are listed as follows.

Firstly, the physical model of the whole system is need to build, i.e., the numerical solutions of multiple bodies' motion. It will be a iterative process: for one body, that is, the attraction from others will be calculated, and the subsequent motion will be obtained; each body is updated and get a new state of the whole system; repeat steps above until any body hit another.

Afterwards, the visual model can be attached onto the previous work, i.e., use the solutions obtained above to role as the motion of the visual model. Various techniques will be applied, which will be claimed in sequent sections.

Last but not least, the user interface is also significant to the project. Users can view the motion of multiple bodies through moving the camera freely. In my plan, I would develop an FPS camera first, and then is the mouse camera which is common in CAD software. Users can also choose the display mode, including some typical body motions or generating specified number of bodies randomly.

The Scope of the project

On the physical level, the numerical solutions of the system state are obtained by iterative method, contrast to the analytical solutions which will get chaos effects by only an extremely small error in the initial state. The final scope is determined according to the development schedule. The order is: single body, double bodies, three bodies and multiple bodies. In the end, the schedule was done, and the self-rotation of bodies was also added. The relativity is not involved in the calculation.

On the visual level, the final scope is determined according to the development schedule. The order is: color, light, light map, texture, shadow, ray tracing, etc. In the end, because of the tight schedule, shadow and ray tracing are not applied to the system. To avoid the drabness of the universe I created, a skybox was added, which contains lots of stars.

On the interaction level, I planned to develop a mouse view adjustment strategy similar to those in 3D modeling software, rather than FPS camera learned in learnopengl. But this is a hard problem and it is not realized in the end. The project use an FPS camera finally. The rest of the human-computer interaction uses the command line. I did not develop a beautiful GUI.

Related CG techniques

OpenGL expects all the vertices, that we want to become visible, to be in NDC(Normalized Device Coordinates) after each vertex shader run. Transforming coordinates to NDC is usually accomplished in a step-by-step fashion where we transform an object's vertices to several **coordinate systems**: local space, world space, view space, clip space and screen space.

Transformations are important in object's motion, including six DoF(Degree of Freedom), i.e., rotation and translation in three dimension space. We use multiple matrices to transform an object. Matrices can not only represent the motion of an object, but also represent the coordinate transformations mentioned above.

Light is a part of the real world, which makes objects more realistic. I use Blin-Phong model in the project, which includes ambient lighting, diffuse lighting and specular lighting. Each of them has a simple computational formula but a good effect.

Without **texture**, it will be hard to specified the color each pixel. Texture coordinates correspond to vertex coordinates one by one. With this technique, the bodies in the system are more realistic.

Shadows can provide information between objects. The most commonly used shadow technique is shadow mapping. The depth is first calculated from the light source, and then from the camera. If the depth of one point in the depth map from the light source is smaller than the one from the camera, it means that the point is blocked to the light source. Then the point only has the ambient lighting.

Ray tracing is different from traditional lighting model. It shots a light from each pixel, and get the object that the light hits. At this intersection point, shadow light is also need to calculate. Repeat the steps above and we will get a more realistic world.

Project contents

Sphere vertices are generated in polar coordinates and drawn with preset precision without relying on any model, shown in Fig.1. One sphere can represent one body.

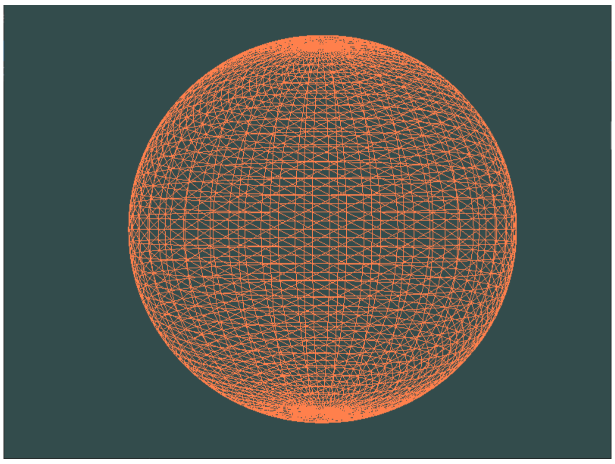


Fig.1: A sphere consisting of triangles.

Shade the sphere using CG techniques, such as light, texture, shadow, ray tracing, etc.

For example, use Blin-Phong lighting model to make lighting on the sphere, which consists of 3 components: ambient, diffuse and specular lighting, shown in Fig.2.

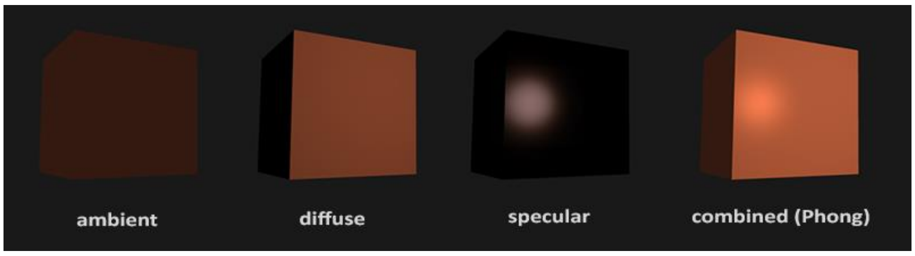
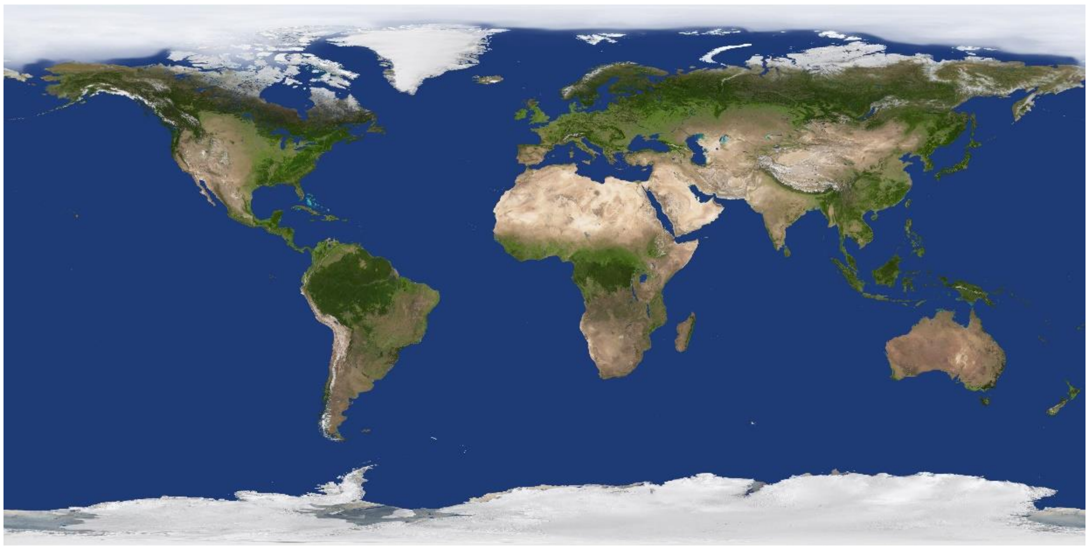
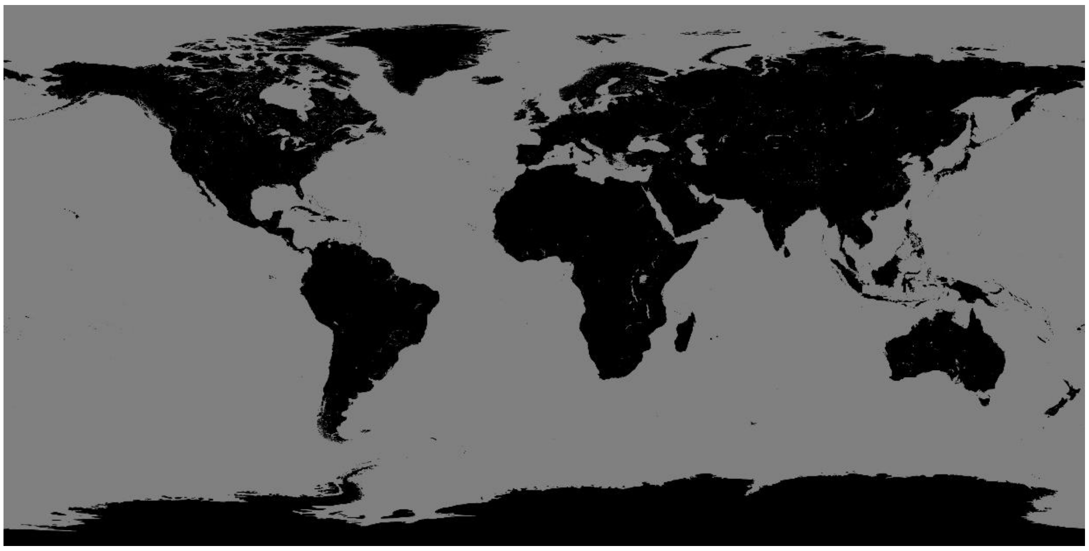
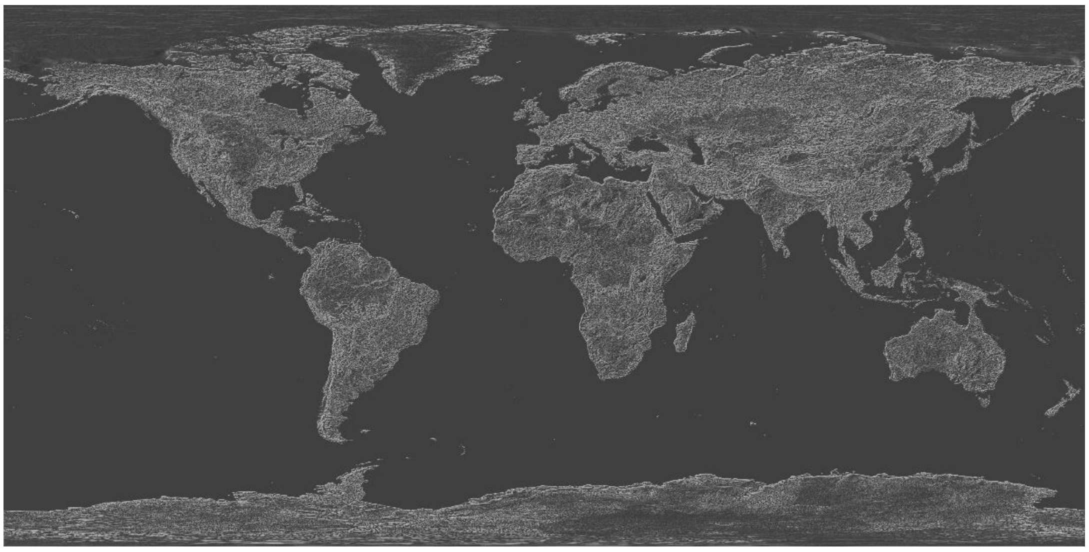


Fig.2: Blin-Phong lighting model.

Textures are used in the project. Some materials are shown as follows.


(a) Diffuse map


(b) Specular map


(c) Height map

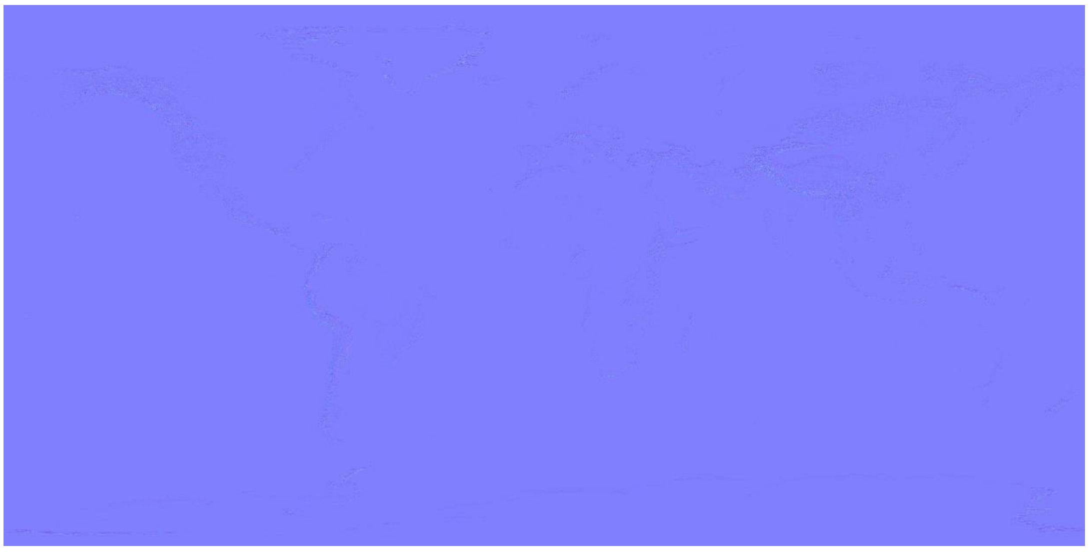

(d) Normal map

Fig.3: Some materials used in the project.

The project models the whole system on physical level using the law of gravitation. Solutions of the multiple bodies' motion may be obtained by these two following methods.

One method is the law of conservation of energy. Compute the kinetic energy and potential energy of each body and get the sum of them. The energy of the whole system will not change and bodies will change because of conversion of two kinds of energies. Get the solution of the system iteratively.

Another method is numerical solutions of second order differential equations. Compute the attraction between bodies and calculate the acceleration (2nd differential of pose) of each body. And get the small changes on velocity and position. Get the solution of the system iteratively, too.

Finally, I applied the second method to calculate the solutions. The method can be described in the following pseudocode.

pseudocode:
Input: Bodies' initial position, velocity and acceleration.
Output: Bodies' trajectories.

```
init a step dt, gravity constant G
for body in bodies
    body.position += body.velocity * dt + 0.5 * body.acceleration * dt * dt
    body.velocity += body.acceleration * dt
    body.acceleration = vec3(0)
    for another in bodies
        if another == body
            continue
        distance = norm(another.position - body.position)

        body.acceleration += another.M * (another.position - body.position) / distance ^ 3
        // Why the index is 3?
        // because acceleration is a vector, with normalize direction vector provided by
        // (another.position - body.position) / norm(another.position - body.position)
    end for
    body.acceleration *= G
end for
repeat until one body hit another
```

The system is established oriented to object and the solution steps are described oriented to process.

Implementation

~Week 09: Search for relevant information and finish the proposal.

Week 09 ~ 10: Finish the single body modeling, and apply lighting on the model.

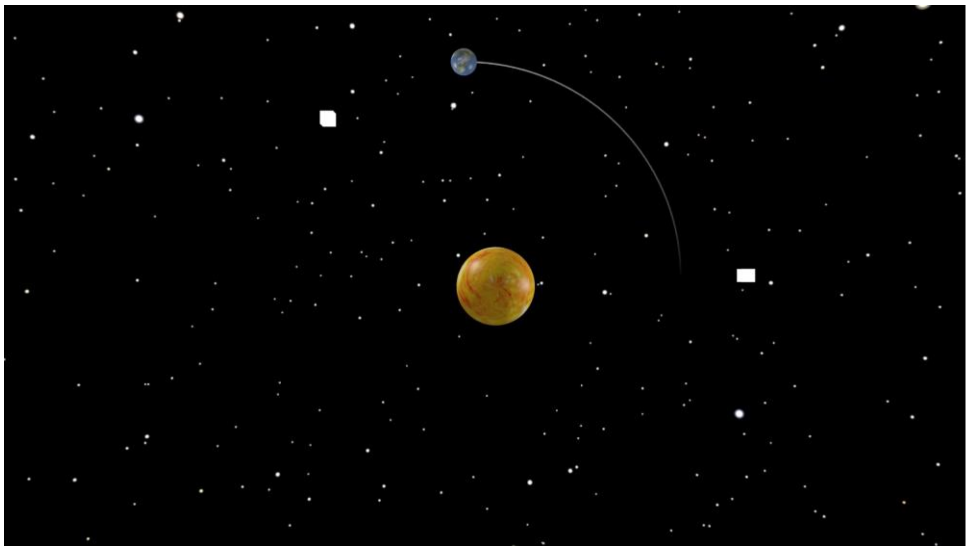
Week 10 ~ 12: Finish the three bodies system modeling and finish the midterm report.

Week 13 ~ 15: Finish the interface and apply deeper techniques.

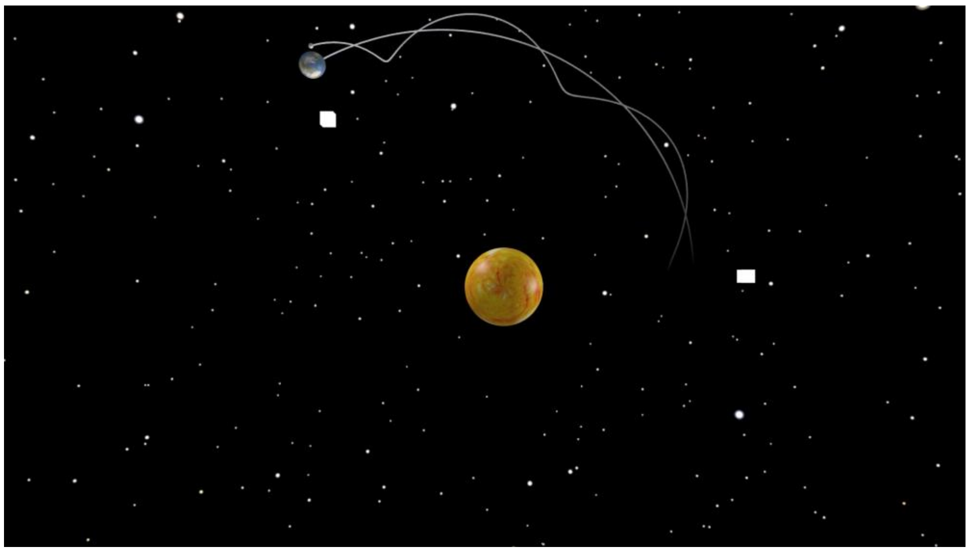
Week 16: Finish the final report.

Results

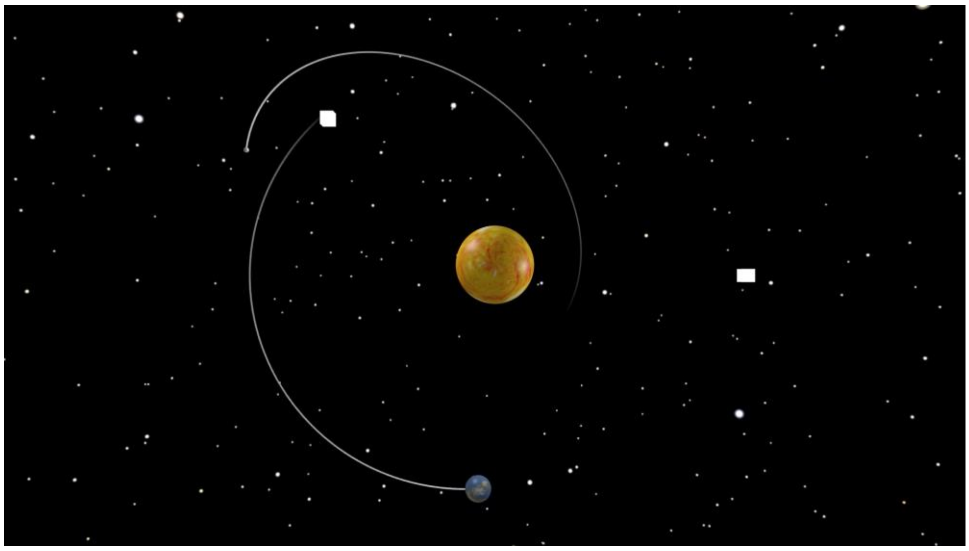
Some typical body motions are preset and realized as Fig.4

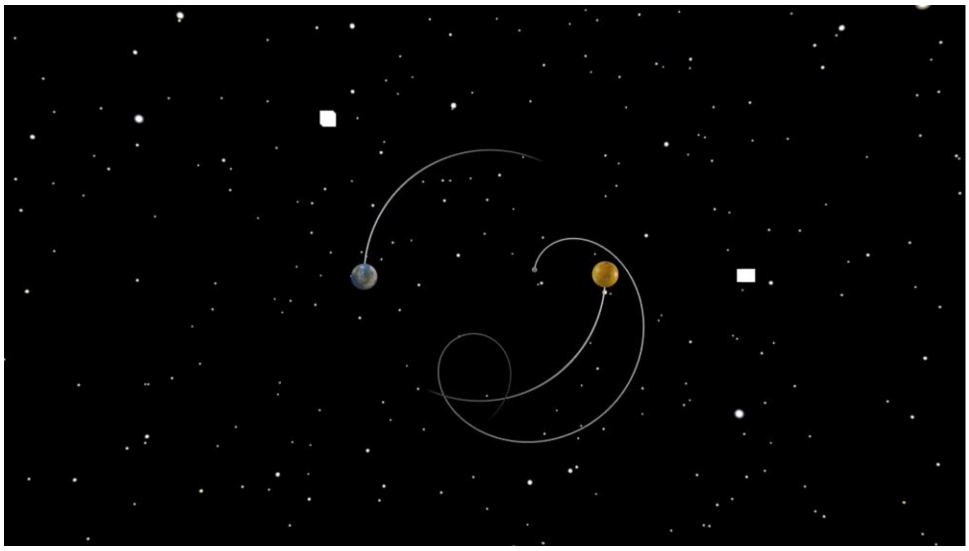


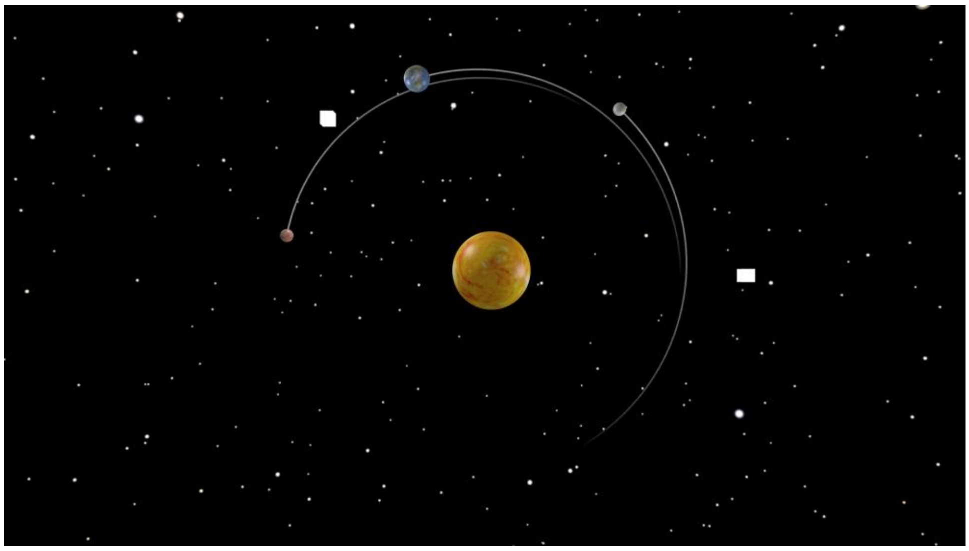
(a) Sun and planet

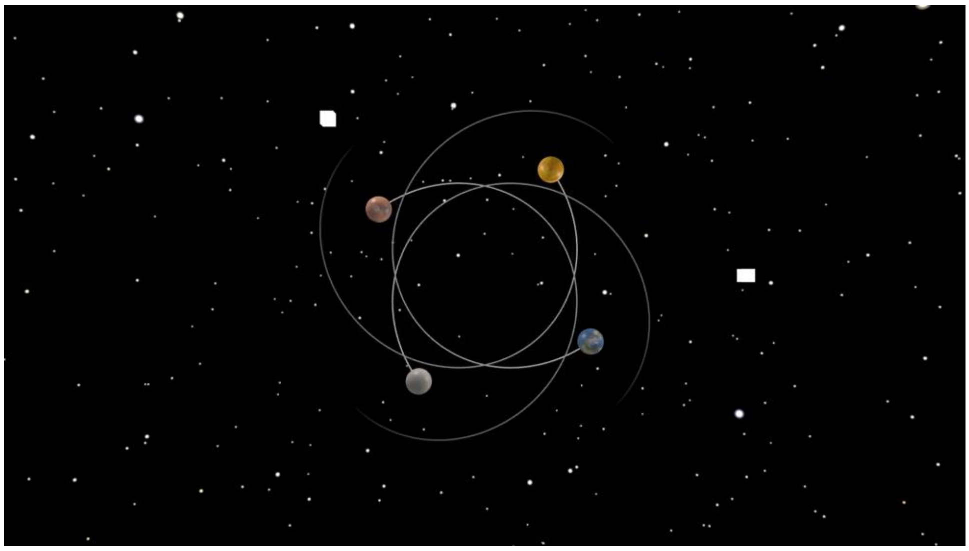


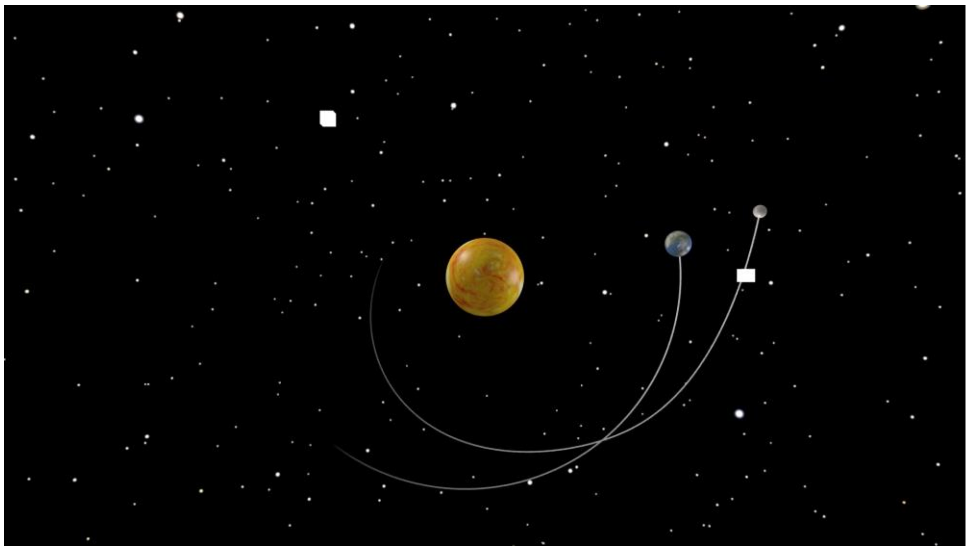
(b) Sun, planet and moon

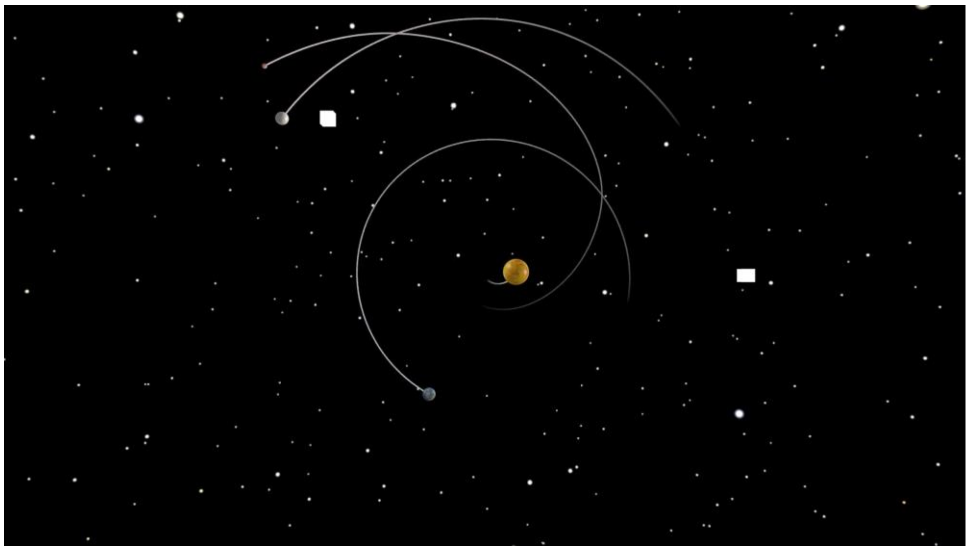

(c) Sun, planet and comet

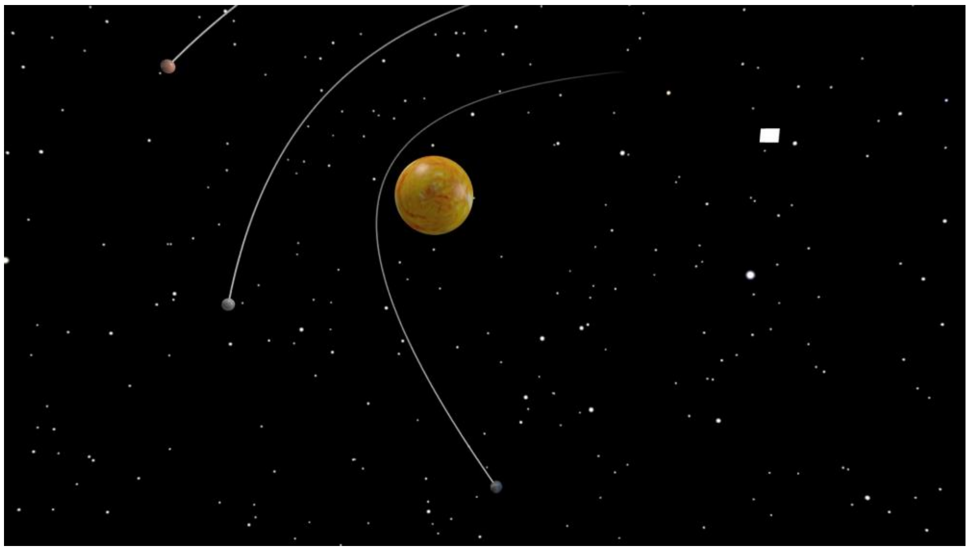

(d) Binary star, planet

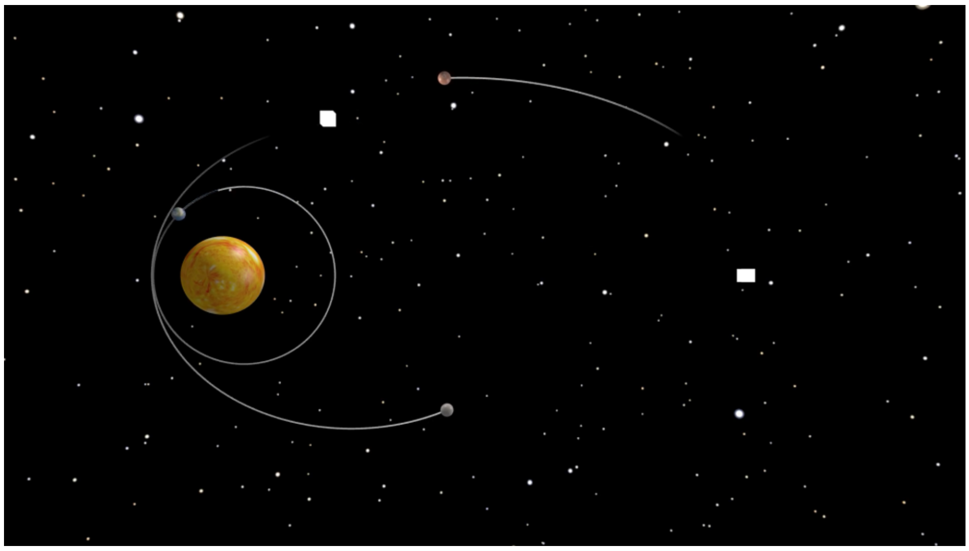

(e) Trojan asteroids


(f) Four star ballet


(g) Slingshot


(h) Double slingshot


(i) Hyperbolics


(j) Ellipses

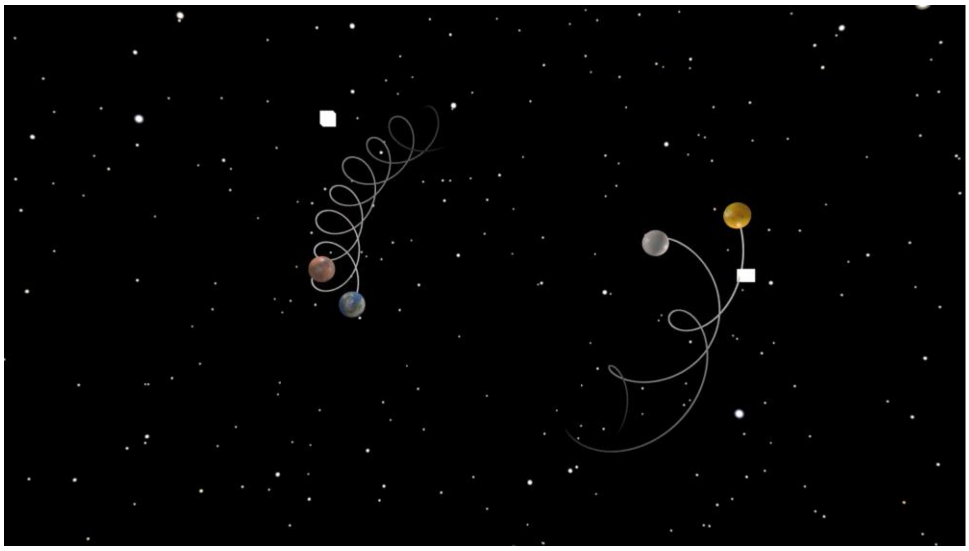
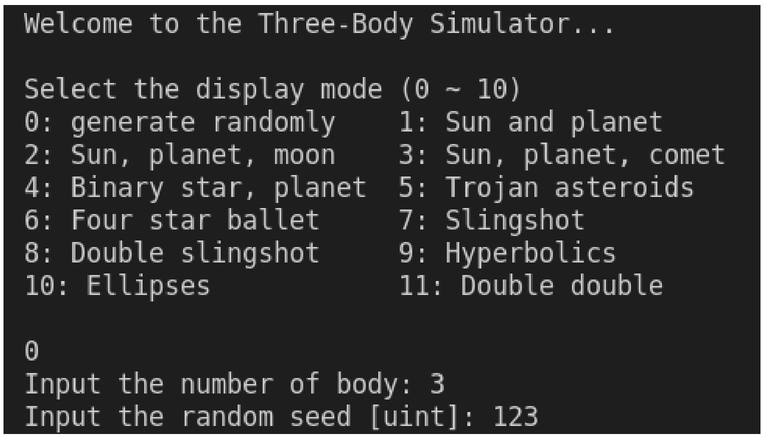

(k) Double double

Fig.4: Some typical body motions.

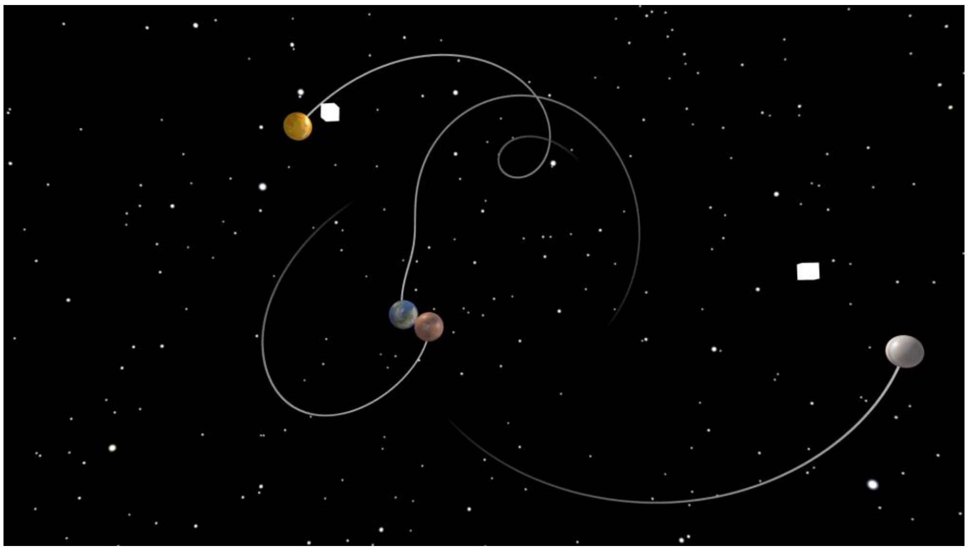
Bodies can also be generated randomly by setting the number of bodies and the random seed to get and record different attempts. User can input the parameters through command line, as shown in Fig.5.



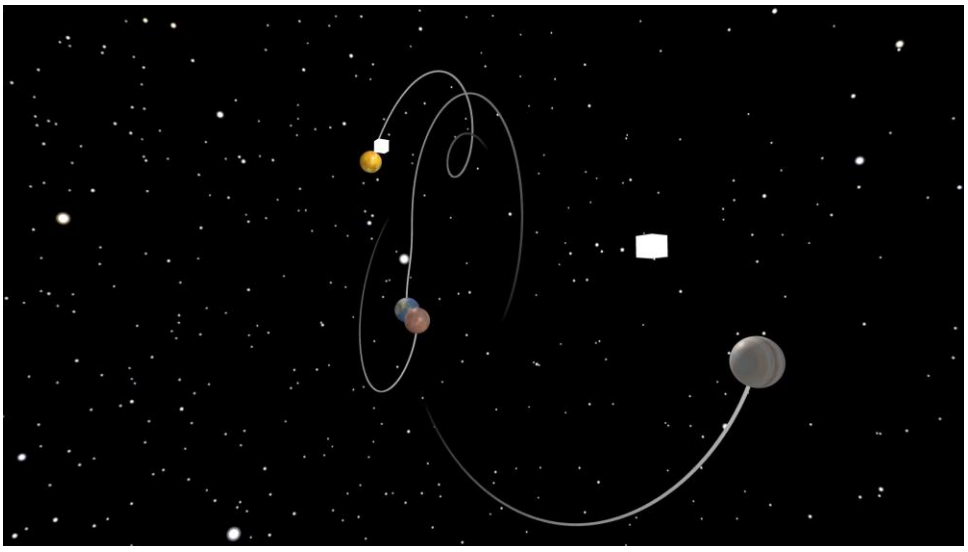
```
Welcome to the Three-Body Simulator...

Select the display mode (0 ~ 10)
0: generate randomly    1: Sun and planet
2: Sun, planet, moon    3: Sun, planet, comet
4: Binary star, planet  5: Trojan asteroids
6: Four star ballet     7: Slingshot
8: Double slingshot     9: Hyperbolics
10: Ellipses            11: Double double

0
Input the number of body: 3
Input the random seed [uint]: 123
```

Fig.5: Command line.

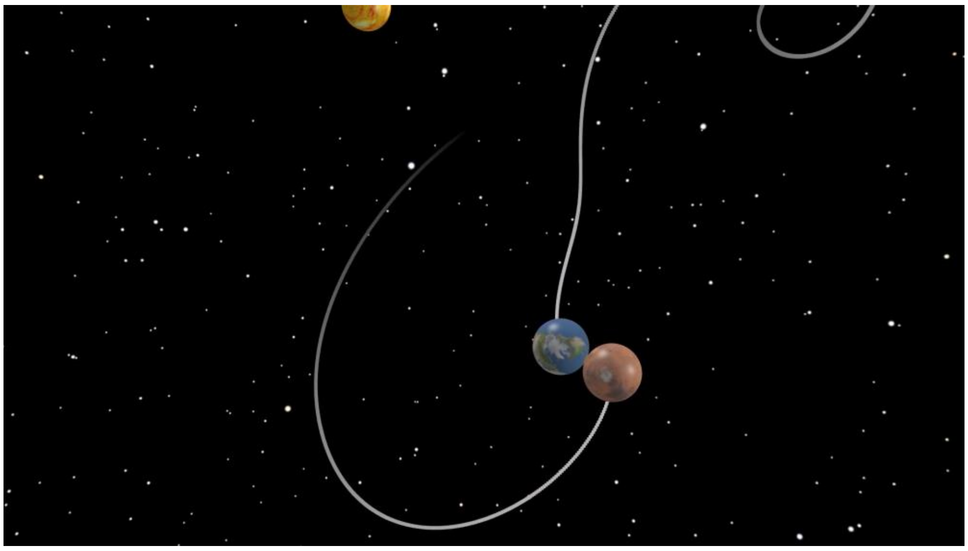
An FPS camera is developed to let users move freely in the space, shown as Fig.6.



(a)



(b)



(c)

Fig.6: Camera motion.

Roles in group

Guo Zhanyu: Responsible for all matters.

References

1. Learn OpenGL, extensive tutorial resource for learning Modern OpenGL