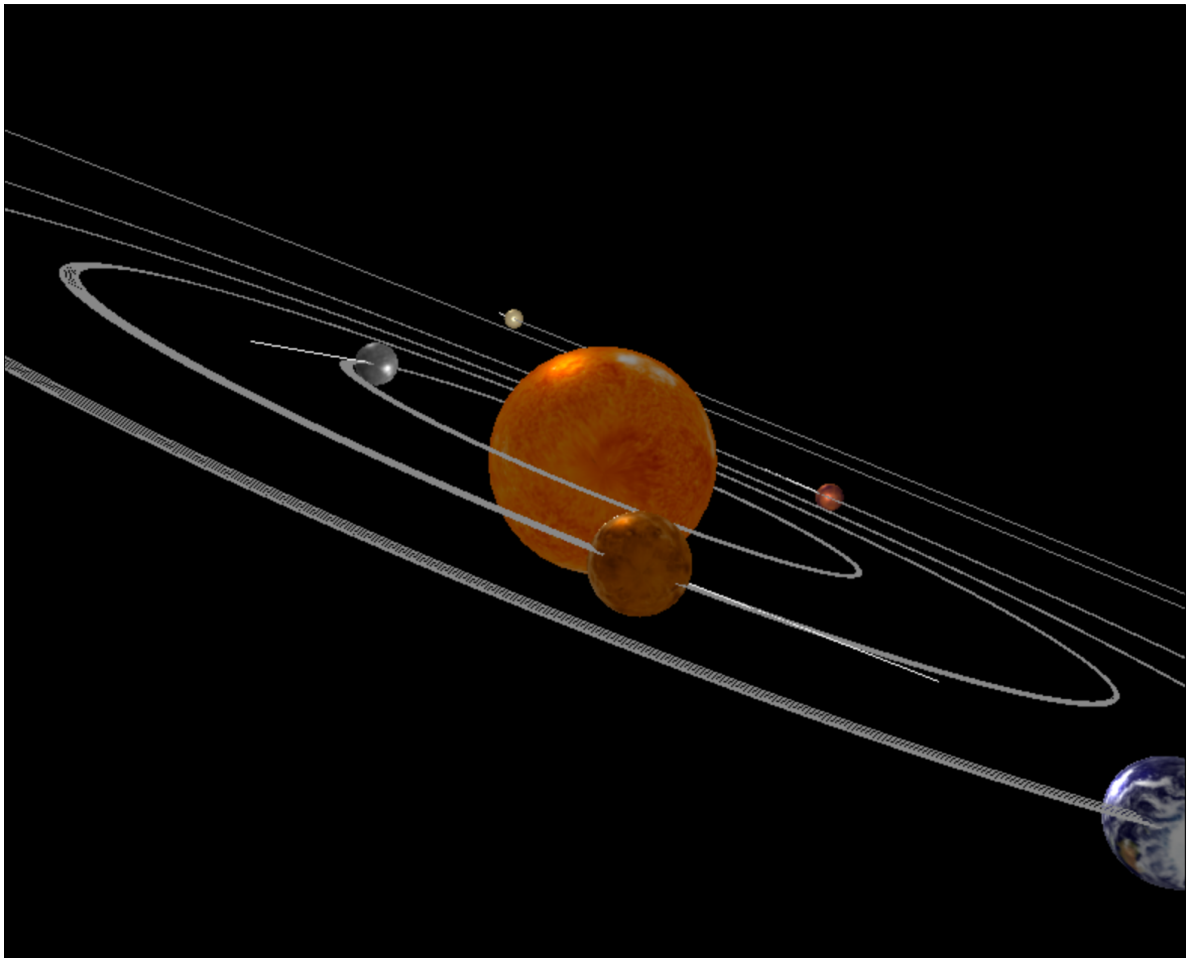


2D/3D Gravity Simulator

KEY FEATURES

- Simulation of Newton's law of universal gravitation among objects
- Simulation of elastic collision between two objects
- Projection of spheres/circles onto a 2D/3D plane

DEVELOPMENT ENVIRONMENT

Language: ECMAScript 6

External Library: jQuery, three.js

Tested on: Safari, Chrome, Firefox

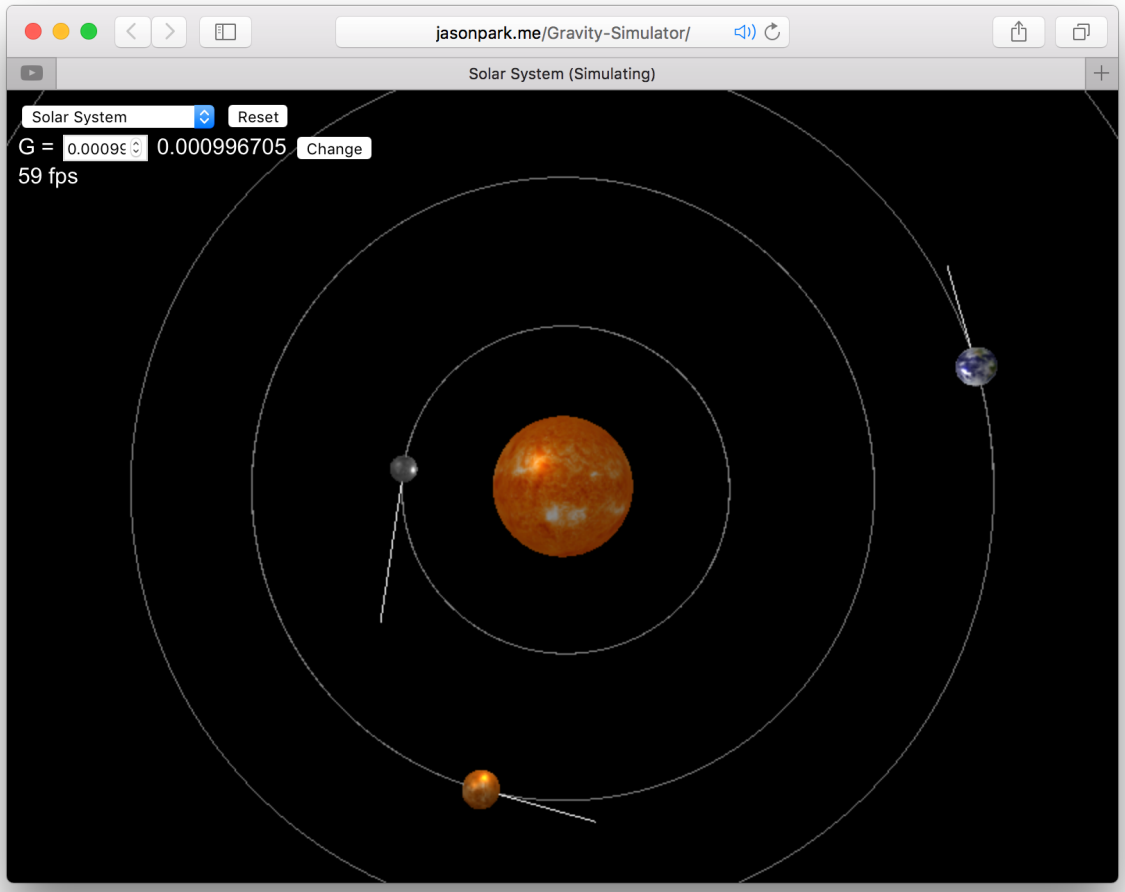
LINKS

Simulator: http://jasonpark.me/Gravity-Simulator/

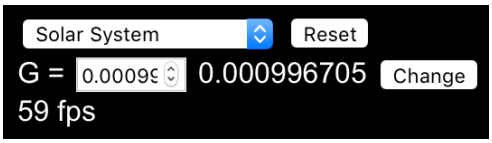
Source code: https://github.com/parkjs814/Gravity-Simulator

INSTRUCTION

Screenshot



Presets



There are several presets including the solar system.

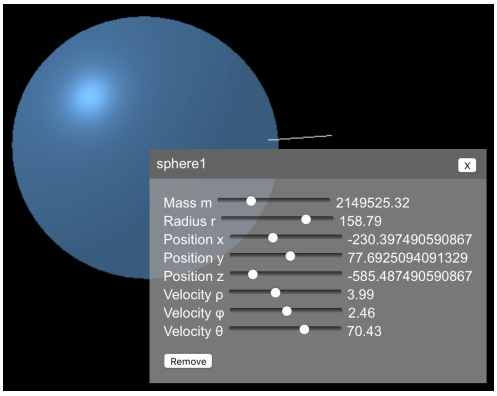
- **2D Gravity Simulator** - 2-dimensional empty scratchpad
- **3D Gravity Simulator** - 3-dimensional empty scratchpad
- **2D Manual** - 2D Gravity Simulator with detailed controllers
- **3D Manual** - 3D Gravity Simulator with detailed controllers
- … and more

User can change the gravitational constant **G** of each preset.

Components

The simulator shows objects (**circles in 2D** and **spheres in 3D**), **traces** of them, and **directions** of them. Traces are represented by gray lines after moving objects, and only the certain length of last traces is shown due to the performance issue. Directions are represented by white lines, and the length is proportional to the magnitude of the velocity vector of the object.

Object Control



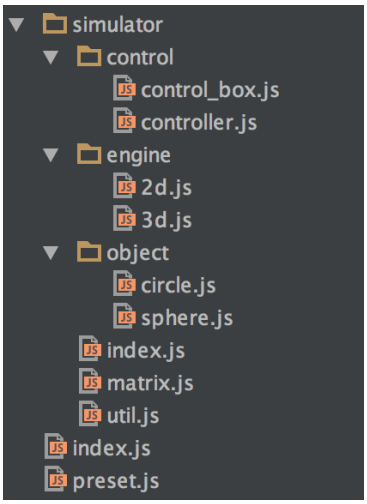
Control of object is only available when the simulator is paused. User can **add an object by double clicking** an empty space on the screen and can **control the objects by double clicking** each object.
Positions of objects are expressed in the Cartesian coordinate system, and velocities of them are expressed in the polar (2D) or spherical (3D) coordinate system.
The objects can be removed by clicking 'Remove' button.
A simulation can be **run or paused by pressing the spacebar**.

Camera Control

In both 2D and 3D, user can **drag the screen with right mouse button clicked** in order to **move** the camera on XY-plane. In 3D only, User can **rotate** the camera by **dragging the screen with left mouse button clicked**. Dragging left and right rotates the camera about Z-axis, and dragging up and down rotates it about X-axis. User can **move along Z-axis in 3D by scrolling**.

DEVELOPMENT

Project Structure



2D/3D Gravity Simulator is written in ECMAScript (or JavaScript) and fragmented into several files according to their roles.

- **simulator/control/** - object control
- **simulator/engine/** - simulation / camera control
- **simulator/object/** - projection / calculation
- **simulator/index.js** - initiation of simulator
- **simulator/matrix.js** - calculation of matrix and vector
- **simulator/util.js** - commonly used functions
- **index.js** - management of simulator and presets
- **preset.js** - definition of presets

Calculation of Gravity

The gravitational forces among objects are calculated based on Newton's law of universal gravitation:

$$\mathbf{F}_{12} = -G\frac{m_1 m_2}{|\mathbf{r}_{12}|^2}\hat{\mathbf{r}}_{12}$$

where **F** is the gravitational force between two objects, **G** is the gravitational constant, **m**'s are the masses of objects, and **r** is the vector from object 1 to object 2.

In order to calculate the gravitational force on an object, it is required to calculate the summation of all the **F**'s between the object and every other object because each object gravitationally interacts with all the other objects. Since **F** and **r** are in the form of vector, the exact same function can be used to calculate the gravitational force regardless of the dimension. After calculating **F**, the acceleration is being calculated using Newton's laws of motion, **F = ma**, and then added to the velocity of the object.
Following **calculateVelocity()** function in **simulator/object/circle.js** implements the above calculation.

```
calculateVelocity() {
    let F = zeros(this.config.DIMENSION);
    for (const obj of this.engine.objs) {
        if (obj == this) continue;
        const vector = sub(this.pos, obj.pos);
        const magnitude = mag(vector);
        const unitVector = div(vector, magnitude);
        F = add(F, mul(unitVector, obj.m / square(magnitude)));
    }
    F = mul(F, -this.config.G * this.m);
    const a = div(F, this.m);
    this.v = add(this.v, a);
}
```

Calculation of Elastic Collision

An equation for calculating new velocities of two objects after colliding elastically can be derived according to the law of conservation of momentum, implied by Newton's laws of motion:

$$m_1\vec{v}_{1i} + m_2\vec{v}_{2i} = m_1\vec{v}_{1f} + m_2\vec{v}_{2f}$$

where **m**'s are the masses of objects, **v**'s are the velocities of objects, ***i*** represents 'before the collision', and ***f*** represents 'after the collision'.

Respect to the final velocities of object 1 and 2, the equation can be transformed to the following equations:

$$v_{1f} = \left(\frac{m_1 - m_2}{m_1 + m_2}\right)v_{1i} + \left(\frac{2m_2}{m_1 + m_2}\right)v_{2i} \quad v_{2f} = \left(\frac{2m_1}{m_1 + m_2}\right)v_{1i} + \left(\frac{m_2 - m_1}{m_1 + m_2}\right)v_{2i}$$

In order to correctly and conveniently regard the directions of the velocities, the vectors of the initial velocities are temporarily rotated until the two objects are aligned on the certain axis (X-axis for 2D and Z-axis for 3D). The vectors of the final velocities are rotated back after calculating them based on the temporarily rotated vectors of the initial velocities.
Following **calculateCollision()** function in **simulator/object/circle.js** implements the above calculation.

```
calculateCollision(o){
    const dimension = this.config.DIMENSION;
    const collision = sub(o.pos, this.pos);
    const angles = cartesian2auto(collision);
    const d = angles.shift();

    if (d < this.r + o.r) {
        const R = this.getRotationMatrix(angles);
        const R_ = this.getRotationMatrix(angles, -1);
        const i = this.getPivotAxis();

        const vTemp = [rotate(this.v, R), rotate(o.v, R)];
        const vFinal = [vTemp[0].slice(), vTemp[1].slice()];
        vFinal[0][i] = ((this.m - o.m) * vTemp[0][i] + 2 * o.m * vTemp[1][i]) / (this.m + o.m);
        vFinal[1][i] = ((o.m - this.m) * vTemp[1][i] + 2 * this.m * vTemp[0][i]) / (this.m + o.m);
        this.v = rotate(vFinal[0], R_);
        o.v = rotate(vFinal[1], R_);

        const posTemp = [zeros(dimension), rotate(collision, R)];
        posTemp[0][i] += vFinal[0][i];
        posTemp[1][i] += vFinal[1][i];
        this.pos = add(this.pos, rotate(posTemp[0], R_));
        o.pos = add(this.pos, rotate(posTemp[1], R_));
    }
}
```

Projection of Objects

2D/3D Gravity Simulator uses a JavaScript 3D library, *three.js*, to project objects in the 3-dimensional space onto the 2-dimensional plane and to render them to the screen. *PerspectiveCamera* class in *three.js* is used to draw objects in perspective, and *DirectionalLight* object is added for users to easily have a sense of direction.

CONCLUSION

Challenges I Ran Into

- I began writing the program in *Python* and used *Canvas* in *Tkinter* built-in package. After I finish writing about 80% of the program, I realized that it is not fast enough to render more than 30 frames per second. I decided to rewrite the entire program in JavaScript, and it now renders 50~60 fps. Old code written in *Python* still remains with several bugs (which is fixed after rewriting) in **old_python** directory.
- I tried to implement 3D projection and rendering without using any library and had a hard time to do all the maths. However, I found several significant bugs on implementing perspective. I had not enough time to fix those, so that I removed all the related parts and replaced them with *three.js* few days before the submission.

Problems to be Solved

- The planets in the solar system do not orbit elliptically.
- Rotating the camera with the mouse is quite complicated and confusing.
- When more than two objects collide simultaneously, they are overlapped and repel each other extremely quickly.
- Too few presets.

What I Learned

It has been a great time to be familiar with calculation of matrix and vector and to fully understand the concepts of gravity and collision. I also learned the importance of a preliminary investigation on technologies I would use, so that I do not make such a time-wasting rewriting again.

REFERENCES

https://en.wikipedia.org/wiki/Polar_coordinate_system

https://en.wikipedia.org/wiki/Spherical_coordinate_system

https://en.wikipedia.org/wiki/Newton's_law_of_universal_gravitation

http://www.real-world-physics-problems.com/elastic-collision.html

https://en.wikipedia.org/wiki/List_of_Solar_System_objects_by_size

http://nssdc.gsfc.nasa.gov/planetary/factsheet/

Special thanks to *Henry Woo* in *KAMS* for helping me do the maths.