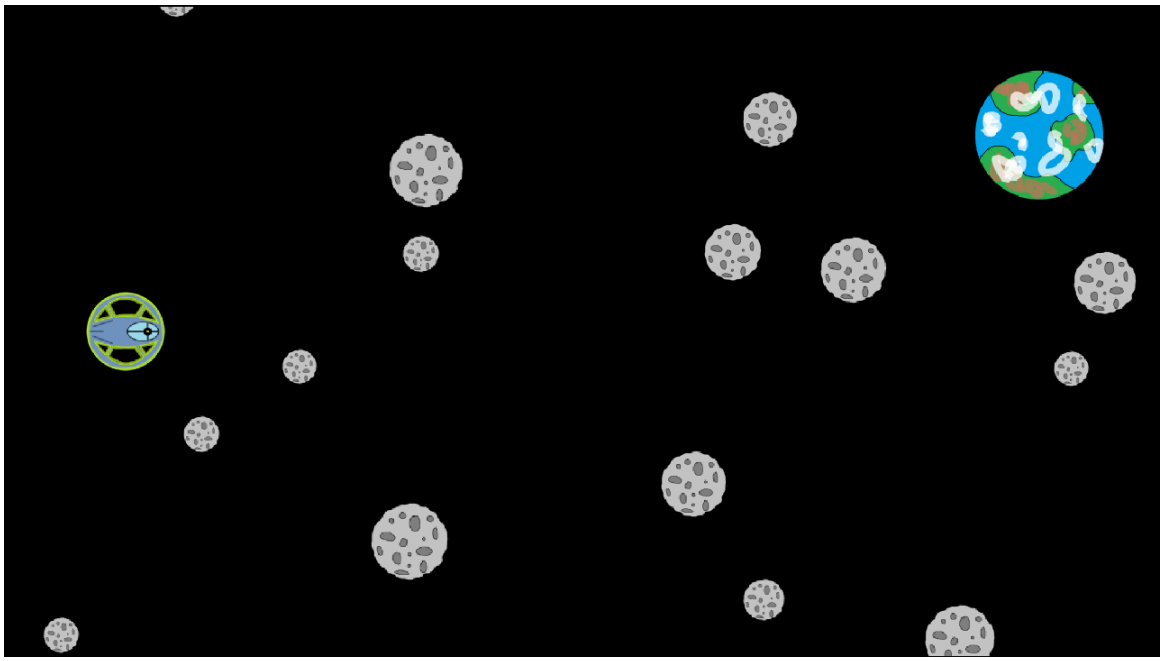
ETH Zürich
Computer Science Department

Visual Computing

Exercise 11- Physics Based Simulation



This exercise includes 5 different files (besides the texture images):

- *index.html*: Used to start the scripts. Open this file in your browser.
- *webgl.css*: Style sheet for index.html.
- *gl-matrix.js*: Contains auxiliary functions.
- *mainScript.js*: Script in charge of running and rendering.
- *Exercise.js*: Script containing the information about the animation.

You can have a peek at all the files (and change them if you want), but you can fully solve the exercise just by editing *Exercise.js*.

To properly load the textures, you need to run a local http server and access the html page at the **localhost:8000**. One way to start a server session is to run the command **-m SimpleHTTPServer** (or alternatively **python3 -m http.server** / **python -m http.server** depending on your python version) from the terminal. Make sure to run the above line from the correct folder.

Introduction

In the previous exercise you implemented different interpolation schemes to create physically plausible animations. Although powerful, sometimes can be tedious to find the right parameters to achieve physically looking results. An alternative to keyframing is to actually perform a physics simulation over time and display the results. In this exercise you will learn how to create a 2D rigid body simulation between spheres. You will be able to create a nice animation of multiple asteroids floating in space and bouncing against each others.
Physics simulations can also be used to build games. As a matter of facts, the same codebase implements also a small game where the goal is to bring the spaceship next to the earth (with low linear and angular velocity). Use the keys **w**, **a**, **s**, and **d** to apply forces and torques to the spaceship.

11.1 Time Integration

A sphere objects includes many rigid body parameters. Between those, you can find: *position*, *velocity*, *force*, *alpha*, *angularVelocity*, and *torque*. The former three are used to advance the simulation in the linear sense, whereas the latter three are used to update the angular properties of the rigid body. Implement the Symplectic Euler scheme for time integration

$$\begin{aligned} v_{t+\Delta t} &= v_t + \dot{v}_t \Delta t \\ x_{t+\Delta t} &= x_t + v_{t+\Delta t} \Delta t \end{aligned} \tag{1}$$

where you first compute the update for the (linear/angular) velocities, and then you use them to update the (linear/angular) positions of the rigid body. Implement the Symplectic Euler integrator in the following function:

```
1
2  function integrate(sphere, dt)
3  {
4    // Task 1
5
6    // Linear
7    // ######
8
9    // Angular
10   // #######
11 }
```

11.2 Collision Handling

At this point, you should see the asteroids floating around, however they overlay each other without colliding. In order to properly implement a full collision resolution between

2D spheres, multiple steps need to be implemented. All of them will be written in the following function:

```
1
2  function sphereSphereCollisionHandling(sphereA, sphereB)
3  {
4    // Check if collision
5    // ##################
6
7    // Task 2a ...
8
9    // Project to avoid overlap
10   // ############################
11
12   // Task 2b ...
13
14   // Compute Impulse
15   // ###############
16
17   // Task 2c ...
18
19   // Update Linear
20   // #############
21
22   // Task 2d ...
23 }
```

a) Two sphere objects are passed as parameters. Your first task is to figure out if they collide. Use the positions and the sizes of the spheres to check if a collision is happening. If that is not the case, no actions are needed and you can return from the function.

b) When two spheres are colliding they overlap. We solve this issue by displacing the two spheres along the collision normal, such that they intersect at a single point. *Note: at this point no velocity is affected!*

c) This is the juicy part of the exercise. By using the law of Conservation of Momentum we compute the linear impulse generated by two spheres colliding. Our technique to solve the collision is to *inject* some momentum into the system. Therefore we need to make sure that overall the additional momentum adds up to zero:

$$\begin{aligned} m^a v^a_{t+\Delta t} &= m^a v^a_t + Jn \\ m^b v^b_{t+\Delta t} &= m^b v^b_t - Jn \end{aligned} \tag{2}$$

where m^a and m^b are the masses of sphere a and b respectively, and J is the impulse applied along the normal n of the contact plane.

So far we defined only four equations, and we have five unknowns ($v^a_{t+\Delta t}$, $v^b_{t+\Delta t}$, and J). The last equation we need relates the relative projected velocities before and after the collision has happened:

$$(v^a_{t+\Delta t} - v^b_{t+\Delta t}) \cdot n = -\varepsilon(v^a_t - v^b_t) \cdot n \tag{3}$$

where ε is the elasticity coefficient which tells the system about the *bounciness* of the two objects. For our case we can use $\varepsilon = 0.9$. After rearranging the five equations above we obtain:

$$J = \frac{-(1+\varepsilon)(v^a_t - v^b_t) \cdot n}{n \cdot n(\frac{1}{m^a} + \frac{1}{m^b})}. \tag{4}$$

d) Once you obtained the impulse J, use Equations 2 to compute the new linear velocities.