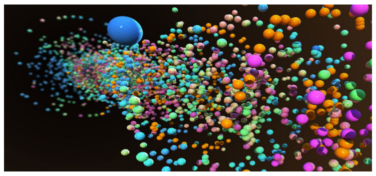
(Optional) MP4: Particle System

Due: December 7 at 1 1:55pm



Write a simple particle system using WebGL to handle the display Particle systems are typically used to model fine-grained physical effects like fire, smoke, and water We will do something simpler and just render a system of bouncing spheres in 3D. The specifics of how you implement the following features are up to you...write an app that you think is fun...

1. Your program will render a set of spheres bouncing around an invisible (or visiible if you wish...) 3D box. You could use a box with corners (-1,-1,-1) to (1,1,1) for example.
   When a sphere hits one of the walls in the box, it should reflect in physically realistic manner
2. You should keep an array or list of particles. Each particle will have a postion P and velocity V. Both of these quantities will be three-dimensional quantities.
   1. It is accapteble to render all the spheres with a uniform radius and color, however..
   2. ...you can include other attributes if you like. For example, you could have each sphere have an individual color or radius or alpha value....
3. You only need to generate one sphere mesh...you simply draw that mesh in multiple different spots each frame.
   1. You can use some of the code from the HelloSolarSystem3.html demo that is one the course website....grab the code to generate and draw a sphere.
4. Your user interface should allow you to create spheres using a mouse click or key press. Each creation event should create X spheres, where X is some number of your choosing.
   1. The spheres should be genereated with a random position and velocity. You will need to bound those values to be reasonable (e.g. position inside the box).
   2. You will also need a reset button that will remove all existing spheres from the scene.
5. After rendering a frame showing the current position of the spheres, you will need to update the position and velocity of each sphere.
   1. Update the position using the current velocity and Euler Integration
   2. Update the velocity using the set of forces you are implementing (e.g. gravity and friction)

Collision Detection

You will need to implement two types of collision detection:

- Sphere-Wall
- Sphere-Sphere

Both techniques are described in the lecture notes.

You should have a user interface that allows you to turn Sphere-Sphere collision detection on and off

When you detect a collision, you will need to adjust the particle position and velocity accordingly

You will upload your files to compass in a zipped folder with name **${NetID}_MP4.zip**. Include all of the files necessary for your application to run locally. Name your webpage *Spheres.html*

Rubric:

Feature	Points	Description
Interactive UI	1pt	Using keyboard or mouse to create particles. Have a reset button to remove all existing particles. Be able to turn Sphe Sphere collision detection on or off. Optional: Add fields on the webpage to set parameters used in the physics.
Phong or Blinn-Phong Shading	1pt	Make sure the scene is well-lit. You can use the same Phong or Blinn-Phong shaders you used in previous MPs
Gravity	1pt	Implement a gravitational force on the particles
Friction or other force	1pt	Implement a non-gravitational force on the particles. Some options would be friction (due to air resistance), attraction, repulsion.
Euler Integration	2pts	Use Euler Integration to compute new particle positions
Sphere-Wall Collision Detection	2pts	Correctly test for sphere-wall collisons and adjust particle position and velocity accordingly
Sphere-Sphere Collision Detection	1pt	Correctly test for sphere-sphere collisons and adjust particle position and velocity accordingly
Documentation	1pt	Have your webpage explain the program interface.