INFO6022 - Physics 2

Project #2: Spheres!

Due Date: Sunday, February 2nd 11:59PM

For this project, you will create a program that demonstrates collision detection and reactions between sphere-sphere, and sphere-plane pairs. This will be implemented with your own physics engine that is structured similarly to many common 3rd party physics engines.

Your program will load a scene defined by an external file. This scene will be an enclosed world (enclosed by planes!) containing 10 – 20 spheres of varying sizes. The user can take control of any one of these spheres and “drive” it around the scene to witness all the marvelous physics in action.

|  |  |  |
| --- | --- | --- |
|  | Item | Marks |
|  | **MUST HAVES: ( Just do it, or you get zero ☹ )**  **ReadMe:** Help me help you. For each point below, an attached file must describe and locate where and how the feature is accomplished.  **Compile and be Self-Contained:** It must compile in Visual Studio 2019 Community without installing anything. You can use 3rd party whatevers (except physics), but all externals must be included in your submission.  **Decoupling:** Except for where you instantiate a concrete physics factory, only your interface library may be referenced in your code.  (**Almost all your marks from upcoming Project 3 will depend on this!)**  **External File:** The scene is defined via an external file. Your entities’ initial physical values must be able to be manipulated here.  **Reasonable Rendering:** This is VISIBLY 3D. You have to be able to see that there are in fact 3D spheres bouncing around in a 3D box. No solid circles on solid planes – how do you expect to see all the cool physics happening? Mmmkay?  **A Good Camera:** The camera has to *at least* :   * Follow around the currently targeted sphere * Be able to rotate around the targeted sphere. * Be able to zoom IN and OUT.   Note: set some reasonable max and min zoom levels for your camera so that I can see most of the scene by zooming all the way out. |  |
|  | **A Better Camera:**   * No Bouncing [1 mark]   + Follow the targeted sphere around in two dimensions, but have the target height remain constant (so the camera doesn’t bounce up and down with the sphere you’re controlling) * Smooth transitions [1 mark]   + When you switch to a new sphere, the camera doesn’t ‘snap’, it smoothly transitions to the new target. | 2 |
|  | **Good Integration:**  Integration scheme is RK4, or some other 4th order method.  <https://gafferongames.com/post/integration_basics/> | 3 |
|  | **Planes:**  There is 1 plane that acts as a floor, and 4 planes acting as walls enclosing the scene; all with appropriate collision detection and reaction. (note the planes are static so they don’t move or interact with each other).  \*The planes’ positions must be configurable.  **Spheres:**  There are 10-20 individual spheres, **each with a different radius** (and corresponding mass).  \*The spheres’ position, mass, and radius must be configurable.  (Marks are focused on good clean implementation of your physics engine classes, your interfaces, and your loading process) | 15 |
|  | **Interactions:**  Reactions are between sphere-sphere pairs, and sphere-plane pairs.  Each pair is checked a single time by the physics engine.  These reactions are *elastic* – that means some energy is lost when they interact.  **It "looks right"** - meaning I should see what I would expect to happen in the real world: small spheres being pushed around and bouncing off of larger ones, while larger spheres more easily push past smaller ones, spheres sliding/rolling around!  (Marks are both for implementation of your collision detections/reactions and for the overall ‘feel’ of the results) | 10 |
|  | **Control:**  A single sphere is under the control of the user.  The controlled sphere has a distinct color (the colors change as you pick spheres)  The control is implemented by applying a force or an impulse to the sphere.  Space (on-pressed) is the hot key that can decide which one is currently under control.  The controls are the ASDW keys (while-pressed), s.t.:  A - applies a force left  S - applies a force backward  D - applies a force to the right  W - applies a force forward  **All applied forces are relative to the camera in the x-z plane and parallel to the "floor" plane**! | 5 |
|  | **BONUS :** You’ve implemented a debug renderer. There’s a hot-key that can switch the rendering to some physics “debug” mode, ie. It is driven entirely by the positions and shapes in your physics engine.  Ideas: Include lines for radii, translucent or wireframe spheres, change colors during collisions. | 2 |
|  | TOTAL: | 35 + 2 |