A3: Animation and Physics

Course: CPSC 427 - Sep 2021
Due: see course schedule

1 Introduction

The goal of this assignment is to introduce you to basic 2D animation. You will extend the salmon game you made for Assignment 2 by including in it a basic particle system implementation.

2 Template

You should use your own Assignment 2 code as a starting point. You will find comments throughout the files to help you guide in the right direction. Entry points are marked with `TODO A3`. The following classes will be important.

Particle Animation You will make your salmon shoot pebbles by implementing a CPU particle system that instanciates spherical objects and simulates their path in water.

Bouncing Pebbles To make the pebbles collide with other pebbles and characters you will need to add functionality to `PhysicsSystem::step`.

3 Required Work (90%)

1. Getting Started

   (a) We recommend pushing your code to a **private** git repository. Commit all your edits from Assignment 2 to git, such that you can track and potentially revert changes.

   (b) Keep a separate copy of your Assignment 2 executable and dlls, to be able to showcase your A2 solutions to TAs on request.

   (c) Play the `a3_reference.mp4` video to get a sense of what a possible assignment solution should look like.

2. Particle Animation (45%, prereq Rigid Body Physics lecture)

   Implement a particle system which generates pebbles that shoot from the salmon's mouth every few seconds (adjust the timing for a compelling visual effect). The pebbles should have randomized initial directions (away from the salmon) and initial velocities. Their subsequent motion should be driven by a combination of these initial properties and gravity. Implement this animation in stages:

   (a) Generate periodic pebbles that shoot from the salmon's mouth and follow a fixed straight-line path at a fixed speed. The `RenderSystem::restart_game()` function provides example code for creating static pebbles on the floor. Remove the line that accidentally adds the player component to each pebble in the `createPebble()`.

   (b) Randomize initial pebble directions and velocities.

   (c) Introduce a new `Physics` component and add it at pebble creation time. It should indicate that an entity is affected by physics and may store related properties such as object mass.

   (d) Add gravity into the system to produce physically plausible (non-straight) pebble paths. Note that the game template computes in pixel units and miliseconds; not meters and seconds as common in physics;

3. Bouncing Pebbles (35%, prereq Rigid Body Physics lecture)

   (a) Add interaction in-between pebbles. Detect when two pebbles collide based on their radius. Make both bounce using physically plausible bounce direction and speed computations.

   (b) Handle the interaction between pebbles and other assets. Make pebbles bounce with the turtles but pass by fish and salmon (these are slim and avoid collisions). Make pebble and turtle bounce assuming both have spherical geometry with a suitable radius and mass. You will have to replace the 'SUPER APPROXIMATE' `collides()` function that is provided in the template.

   (c) One could implement the above with if conditions on HardShell and Physics components, but this is inflexible. Introduce a new component alongside `Physics` such that you can ensure that turtles bounce with pebbles but are unaffected by gravity. You should implement it such that there is one component to indicate that an object is affected by gravity and another that it bounces on collision. Ignore the comment in `components.hpp` that suggests Pebbles should have a HardShell, it would make them deadly to the salmon.

4 Creative Part(20%)

The required code changes described so far will let you earn up to 80% of the grade. To earn the remaining 20% to make the game more appealing by implementing one advanced

feature. You can also gain bonus points when exceeding our expectations. **Marks for the advanced features will be granted only if both they and all basic features are fully implemented and functional.** Advanced feature suggestions:

1. Machine-oblivious time-stepping that produces consistent animation and AI across platforms and computational loads. Create and document a test case that showcases the difference.

2. Include the force of water and the water flowing leftwards in the pebble motion computation. Moreover, add collisions with the bottom side of the window and add a rest state that disables gravity when pebbles come at rest on the ground to improved the computational efficiency.

3. Use a single draw call to render all the pebbles at once (requires good OpenGL knowledge), e.g., using `glDrawArraysInstanced()`, `glDrawElementsInstanced`, or by using geometry shaders. This should be more efficient since pebbles only differ in position and scale, they have the same appearance (shader).

Use your imagination to make other additions than the ones listed above, however, please make sure you focus on tasks involving advanced OpenGL rendering, advanced physics, and advanced animation knowledge.
To support both basic and advanced visualization and control features, you need to add a toggle option where the user switches between the two modes by pushing the 'a' and 'b' keys ('a' for advanced mode and 'b' for basic mode; either at startup or during the game).
Document all the features you add in the README.md file you submit with the assignment. Advice: implement and test all the required tasks first before starting the free-form part.
To get full credit you should add at least one of the advanced features above and make it fully functional **and** free from bugs. The grading of additional bonuses, features, and the size of bonuses will be at the marker's discretion. A bonus is given for solutions that go beyond the examples listed above. **Multiple partially implemented features will not receive full credit.**

5 Hand-in Instructions

1. Create a folder called "a3". As before, copy all your source files and the CMakeLists.txt as present in the template to this folder (same folder structure; the TA should be able to run CMake and compile). Double check that you include the `shader` folder. Excluded all generated files, such as /build, .vs, /out and the example videos! These would consume a lot of space on our server.

2. In addition, create a README.md file (Markdown language as used on github) that includes your name, student number, and any information you would like to pass on to the marker.

3. The assignment should be handed in with the exact command `handin cs-427 a3`

   This will handin your entire a3 directory tree by making a copy and deleting all subdirectories. If you want to know more about this handin command, use: man handin. You can also use the web interface on your myCS page to upload the assignment.

Recall, do not publish your solution on github or any other place. Neither during the course nor after; both is considered cheating.