ThePhD
COMS W3101 – Programming Language – Python
Professor Lawrence Stead
Python Exploration Project
November 2nd, 2015

Fireplace: A Realtime Particle Emitter Test

The idea came from seeing many traditional effects in computer graphics, which can mostly be traced back to a paper by William T. Reeves[1]. In it, Reeves laid out many of the fundamentals of particle systems (Page 93 and onward discuss the base equations used, including: distribution schemes, particle properties, and velocity/acceleration considerations).

The goal of this project was to see if something realtime could be made in python (for a very barebones definition of realtime and something). The end result is a little fireplace, with the heavy draw aspects such as Block Image Transfer (blitting), rectangle drawing, and graphics device setting taken care of by pygame. Perhaps it was the heavy use of `random()` and such, but the framerate visibly struggles when approaching 350+ particles. Not much more could be done with it (this excluded adding a wind effect and another natural phenomena) without drastically reducing the framerate and turning a realtime particle emitter into a not-so-realtime particle emitter. This was not a limitation of the actual physics simulation itself: commenting out the render code showed that the majority of the runtime was spent performing a render, and that drastically slowed things down. Pygame does not seem to be very performant for realtime graphics rendering, even when the shapes being drawn are absolutely simple. See the Table 1 below and the notes further down when switching out the rendering code for more details.

The core working of the program is to start up a continuous loop (see main.run) that never exits (unless the user tells it to by closing the program or closing the window). Once in that main loop, there are two primary components after setup finishes: the `main.update` method which runs the physical 'simulation' of the particles, and the `main.render` method which draws the current state. The core idea behind any realtime simulation on a machine is to simply run the simulation in various tiny steps as fast as possible, in order to fake the idea of movement or passage of time in an environment: thus, main loops continuously. In `update`, emitter's `step` method is called and physics functionality, making it add things like velocity to the current position and acceleration to the current velocity (with respect to the current time constant). In `render`, a representation of our simulation is chosen to show to the user: it could generate one text file with outputs and positions per frame, but it is much more intuitive and simple for the user to understand an actual moving image in front of them.

The general idea of the emitter is, if there are less than a certain amount of particles, to generate more particles with random velocities and offsets from a particular origin (up to a certain amount per each simulation step or frame). If enough particles with the right colors, velocities, and offsets, it begins to look like a single fuzzy object despite being composed of many individual pieces. Essentially any computable effect can be placed on the emitter. The fire emitter goes through quite a lot of mathematical calculations to get a nice-looking drop-off effect for ash (computed in the rendering step).

A possible enhancement would be to use textures instead of rectangles or circles. More could be added for additional effects (bubbles in the ocean, fog, and even smoke for the aftermath of the fire) and texturing would give the impression of more advanced effects, asides from just the pixilated fire imitation. There is only a basic fire and rain emitter, and both could be fairly convincing given the right environment (say, a basic pixilated game of some sort).

To get all of this to show up on the screen, as mentioned earlier, pygame[2] is used. It's really just a wrapper around the C library SDL[3], which is responsible for putting things on the screen without getting involved in low-level details. This allows us to clear the window to the white color and then draw the several particle bits that make the fire effect. The sluggishness around 300 particles seemed a bit strange, however, so to further test the limits of how well certain libraries performed with render, a quick test was done with another library called SFML[4]. There seemed to be a slight speed increase when using python SFML, but this could just be that the usage of the sprite drawing was better than normal. The below table shows some

Rendering with 350 particles, fire emitter	Physics Steps (10 ms per step)	Update Time (ms)	Render Time (ms)	Frames per Second
No Rendering	0-1	0.05	0.06	16,666
pygame Rendering	4-5	11.7-23.5	48-50	19.21-21.89
pySFML2 Rendering	1-2	11.6-22.2	12.1	45-89 FPS

In the end, 350 particles for the fire emitter ended up looking smoother in pySFML2. It could be that pySFML2, since it is working more closely with OpenGL than traditional SDL does (its legacy API tends to

model that of GDI+ and directly blitting into a target window), it make benefit from a tiny bit of acceleration that SDL may not.

[1] - Reeves, William T. & Lucasfilm, Ltd. *Particle Systems. A Technique for Modeling a Class of Fuzzy Objects* ACM Transactions on Graphics (TOG). Volume 2, Issue 2, April 1983. Pages 91-108. New York, NY, USA.

[2] - Pygame: http://pygame.org/hifi.html

[3] - Simple DirectMedia Layer (SDL): https://www.libsdl.org/

[4] - pySFML. http://python-sfml.org/index.html