Joseph Andaya
David Ngo
Daniel Urabe
Shandip Bhangal

TEAM PROJECT FIRE THINGY! or something like that...
COMP 137 Project Report

Description

Our project is a particle/fluid simulation of fire where the main thread generates an object that will act as a source of heat. As heat radiates from the source object, fire will emit from it when certain particles reach the temperature of 350. Each thread handles its own set of particles that contains several variables that describes the characteristics of said particle, namely size and color. The particles of the fire vary in color based on height while also moving around based on a simple sine wave algorithm. The size of each particle diminishes as it goes up so they get smaller and smaller until they get deleted at a very small size and reborn from the fire source in a continuous loop.

Design

- Framework
  - OpenGL/GLUT
  - SDL
- Simple Object (source)
  - Generates smaller fire particles at certain temperature.
  - Particles are distributed equally amongst each thread.
  - Each thread offsets his/her corresponding set of particles.
  - Pthreads is used to accommodate the fast refresh rate of the game loop (60fps).

Serial description

The serial implementation of the program used one process to generate and render each particle. The particles were generated randomly in different locations on the heat source while the colors of each particle changed depending on the height as the particles moved in a sine wave like motion upwards slowly shrinking in size.

Parallel description

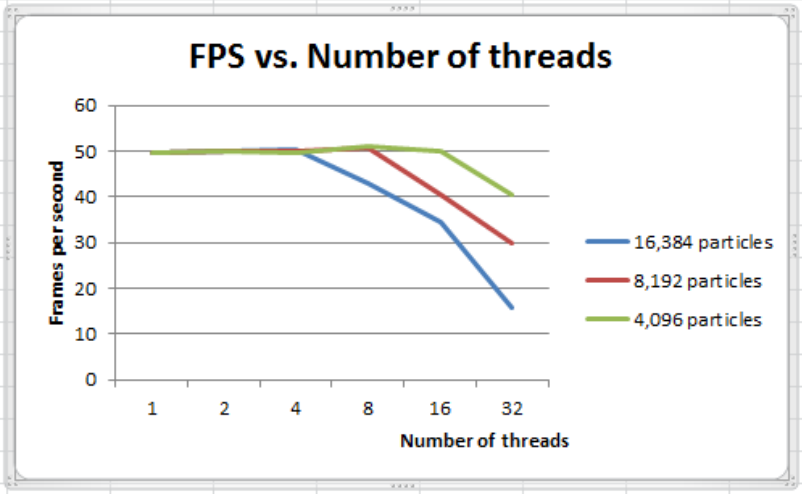
Our parallel implementation was done through the use of pthreads due to the fast updating mainloop (ideally 60x per second) and our desire to reduce overhead time from communication. The main thread renders the particles as every other thread gets its equal distribution of particles by #PARTICLES/#THREADS and updating each particle by the time the main thread is ready to render. Each individual thread handles an equal set of particles to update for the main thread to render ideally 60 times per second. The data that each thread is responsible for are the color of the particles, the coordinates of the particles relative to the heat source (the box), the changing size of the particles, and the offsets of each particles as they change after each loop. We used a barrier (see bool render[THREADS]) in each thread and also the render function to make sure that every particle is completely update before the main thread renders the next frame. This ensures a smooth animation.

Conclusion and Additional analysis

Timing Analysis

We timed our program according to two factors: the amount of threads and the amount of particles. To carry this out we tested our program against three different amounts of particles and each of those particle sets were tested against six different amounts of threads. The performance was measured by the frames per second (FPS) of each set, where the higher the frames per second, the better/smoother the program was running. The results are shown below:

16,384 particles		8,192 particles		4,096 particles	
num of threads	frames per second	num threads	frames per second	num of threads	frames per second
1	49.8	1	49.8	1	49.8
2	50	2	49.9	2	49.95
4	50.45	4	49.85	4	49.8
8	42.97	8	50.55	8	51.08
16	34.35	16	40.55	16	50.1
32	15.7	32	29.96	32	40.55



Effect of parallelization on efficiency

Parallelization of our program gives a FPS increase of about 1.2% with 4 threads at 16,000 particles. Unfortunately, the FPS decreased as we added more threads than 4. The reasons for this are explained in the “Challenges” portion below.

Ease of experimentation

Our project was a little bit difficult to parallelize at first, but once we got it to actually work, experimentation was quite easy to do. We could easily manipulate the amount of threads and the amount of particles, so testing was just a matter of changing values and executing the program multiple times. We were also able to make it so that by pressing "t" while executing the program, the particles being displayed at the current time would only belong to that of the selected thread, allowing for us to easily check if our code was parallelizing correctly or not.

Challenges

It would seem that increasing the amount of threads beyond 4 for 16000+ particles would cause a decrease in frame rate. This is partially because of our barrier that we put in our render function that waits for each thread to finish its update before rendering the next frame.

Initially we intended to implement the fire using real physics where temperature would affect the particle coloration and the particles would act more life like. However this drew away attention from the real point of this project, to understand and implement parallelization. So in favor of the true intention of the project, we cut this feature out of our project. The reason why temperature is monitored and why pressing the "w" key to reach a temperature of 350 degrees before the particles could start rendering is a relic of this forgotten feature. Other lost intended features in our original proposal were a halt or pause feature and a hover over mouse feature that would display the information on any particle. Again, these were dropped in favor of focusing on parallelization.

Issues with parallelization

Communication between the subthreads and the main thread for telling the main thread that they are ready for updating created overhead time that most likely is the cause for lower FPS with more than 4 threads with 16000+ particles. We also were not able to hit our expected goal of 60 FPS for any amount of threads/particles.