Lava Lamp simulation

Credits include: Nicholas Marks creator, code adopted from examples in class by Matt Wright and Karl Yerkes. I also began from a homework submitted by Elliot Wobbler as a template. The Lava Lamp simulation program is my final for UCSB course MAT 201B for the Fall 2013 quarter.

I created a Lava Lamp simulation because I had a lava lamp on the desk as I worked on homework and studied throughout the quarter. Our new apartment does not have very good lighting so anything helped brighten the room. As I looked at it the first few weeks of class I thought that it would be a cool idea to feel as if we were inside of a lava lamp. In the allosphere we get the feeling as if we are in the middle of a lava lamp, with a heating element underneath and a cooling element above. The additional benefit we get in the allosphere is the sound representation, in this case I used temperature since we cannot see it. I used color to help visually represent the position, this helps a viewer to identify clusters of agents in transition.

The data structure for the agents include position, radius, temperature, frequency, color which are explained in detail later. An agent also holds a Boolean value for each rise and fall to ensure a continuous flow between top to bottom and less lingering around the origin. Each agent has an ID number and a state counter called rest which helps toggle rising and falling after a designated number of cycles (initially 500). The position of each sphere is determined through various conditions based on current position, temperature, and randomness. There is a bounding box in place in order to keep the agents in view, and the movement is dampened in the X and Z direction so that the Y movement is more dramatic. The color of each sphere changes depending on the position in the Y axis, lowest position is red and highest is blue with a smooth transition over the Y positions. This equation translates position from -10, 10 to a float between 0.0 and 0.6667 to have a smooth transition between red to blue.

color = (position.y + 10 ) / 30 ;

The sound that each sphere makes is based on the temperature translating a temperature from 0-100 into frequencies in an audible range 100-500.

frequency = temperature \* 4 + 100 ;

The radius changes based on the spheres Y position. At the origin the sphere has a constant 0.5 radius, but as it rises or falls the radius changes based on different equations. When position.y <= 0 the radius follows this equation:

radius = StartRadius(0.5) + (position.y \* position.y \* 0.01) ;

Otherwise when position.y > 0 :

radius = StartRadius + (position.y \* position.y \* 0.1) ;

Which seemed too big and was capped at a MaxRadius of 7.

Therefore when position.y = -10 (maxY) , the max radius is 1.5; and when position.y = 10 the max radius is 7. This gives the perception that the agents are bonding together.

Lava Lamp Simulation is written using the C++ programming language including the use of Allosystem, Gamma, and various math/random libraries. The basic idea of the program is to simulate a lava lamp. My program is not modeled to follow newtonian physics but my own interpretation of how a lava lamp works. All spheres of lava behave in a similar pattern with a bit of random noise added. Spheres constantly lose a small amount of temperature, this allows the balls to float downwards. Once the sphere is below a Y threshold the sphere, is held in the lower region and begins to increase rapidly in temperature. Either when the temperature is high enough or by random chance the sphere would rise only making it to the top if it has a high enough temperature to counteract the constant temperature loss. Again once the sphere is above a Y threshold its temperature is increased by a constant to keep the sphere afloat, but once the sphere loses enough temperature or by a random chance the sphere will begin to float down again. Technical difficulties mostly came with my original proposed project which would have required me to learn more than I had time for in addition to the course requirements. Changing the radius was an artistic interpretation as one technical difficulty would have been to simulate the shapes of a mesh representing the blobs forming in a lava lamp instead of using spheres. The shapes formed in a lava lamp seem to follow complicated fluid dynamics and physics which I would have liked to implement.

I learned that not all programming languages behave the same way. I first tried to translate code I wrote for a visual programming language Scratch, into C++ Allosystem to create a keyboard instrument. It was not as easy as copying the same logic as Scratch handled my commands differently than C++. My own attempts at redoing my logic with different visuals were not impressive visually or mechanically. What I mean by mechanically is that tracking multiple buttons key presses became difficult to interpret correctly. I looked into learning PureData to create my instrument and controller but I could not find enough documentation explaining the complicated “instruments” I was looking at. I would like to continue learning other languages but for the scope of the class I decided the best thing would be to stay with C++ and Allosystem libraries. As I have been waiting to take this course for a number of years I didn't want to deliver an app that didn’t work in the Allosphere. I am glad that I took a step back and focused on a full surround app because I did not get the hw4 surround app working and I really wanted to have something displayed in the Allosphere.