Project #7

OpenCL / OpenGL Particle System

120 Points

Due: June 13

Introduction

Particle systems are used in games, movies, etc. to depict phenomena such as clouds, dust, fireworks, fire, explosions, water flow, sand, insects, wildebeests, etc. Once you know what they are, you can't stop seeing them.
We think of particle systems as being created digitally, like you will be doing. But recently, one was create analogy, like with LEDs attached to pigeons. (No, really.)
A bit of particle history, particle systems were first seen in the *Star Trek II: The Wrath of Khan* Genesis Demo.

To make a particle system work, you manipulate a collection of many 3D particles to exhibit some behavior. (Look here for more information.)

In this project, you will use OpenCL and OpenGL together to make a cool particle system. (The degree of cool-ness is up to you.)

Requirements:

1. Design your 3D environment. The particles need to start from somewhere. Where should that be? The particles need to start with initial velocities. What should those be?

   Your 3D environment needs to have at least two "bumpers" in it for the particles to bounce off of. Each bumper needs to be geometrically designed such that, given a particle's XYZ, you can quickly tell if that particle is inside or outside the bumper. To get the bounce right, each bumper must be able to know its outward-facing surface normal everywhere.

   Let's face it. Spheres are computationally "nice". In computer graphics, we *love* spheres. It is fast and easy to tell if something is inside or outside a sphere. Determining a normal vector for a point on the surface of a sphere is even easier.

It is OK to assume that the two bumpers are separate from each other, that is, a particle cannot be colliding with both at the same time.

2. Create an OpenGL buffer object to hold the particles' XYZW positions as an array-of-structures.
   Create an OpenGL buffer object to hold the particles' RGBA colors as an array-of-structures.
   Create a C++ array-of-structures to hold the particles' XYZ velocities.
   (OpenGL buffer objects are used for position and color, but not velocity. This is because OpenGL will need to use positions and colors in the drawing, but not the velocities.)
3. Determine good starting values for all 3 data structures.
   For the position and color buffers, use glMapBuffer( ) to send the values directly to the GPU.
   The velocity array is a C++ array-of-structures, so just fill it using C++ code.
4. Create two OpenCL buffers from the position and color OpenGL buffers using calls to clCreateFromGLBuffer( ). You don't need to transmit the data to these OpenCL buffers -- it is already there in the OpenGL buffers that they are now linked to.
5. For the velocity array, create an OpenCL buffer using *clCreateBuffer* and transmit the C++ array-of-structures to it like you've done before using clEnqueueWriteBuffer( ).
6. Decide on a time step, **DT**. Your .cl program will need to know about it. You can pass it in, or hard code it.
7. Create a .cl OpenCL kernel that will advance all of the particles by the timestep **DT**. The sample code shows giving the .cl program access to the particles' positions and velocities. You will need to use these, and will need to update them.
8. Your OpenCL .cl program must also handle the bounces off of your bumpers. Be sure to draw these bumpers in your .cpp program so that you can see where they are. Again, spheres are "nice".
9. The sample code shows giving the .cl program access to the particles' colors. But, Joe Parallel didn't do anything with these. But, *your* .cl kernel needs to dynamically change the color of the particles. You could base this on position, velocity, time, bounce knowledge, etc. ***But, the color of each particle needs to change in some way during the simulation.***
10. Leave the local work-group size at some fixed value. You can pull this out of your experience with Project #6, or you can experiment with the timing. Vary

the total number of particles and measure the performance using units that make sense.

11. Make a table and a graph of Performance versus Total Number of Particles
12. Turn into *teach*, individually, not in a .zip file:
    1. Your source code (.cpp and .cl). Give your .cl file a name that is some variation on your name or login.
    2. **Your executable**
    3. Your commentary in a PDF file.
13. Your commentary PDF should include:
    1. What machine you ran this on
    2. What dynamic thing did you do with the particle colors
    3. Include at least one screen capture image of your project in action
    4. Show the table and graph
    5. What patterns are you seeing in the performance curve?
    6. Why do you think the patterns look this way?
    7. What does that mean for the proper use of GPU parallel computing?