CUDA-Driven Particle System

Michael Rubino

Overview

The program will utilize a parametrically defined helix to generate forces to apply to a large quantity of particles - hopefully several million - through the application of CUDA kernels.

Cyclone

Generated by parametrically defining a helix and giving it a variable width on the y-axis (I'm currently accomplishing this through a simple linear interpolation between two widths as height increases, resulting in an inverted cone-like shape).

Force vectors are defined by forming a vector between a point on the helix and its previous point. This results in one less force vector being formed than there are total points on the helix. Their magnitudes are specified by user-defined values when forming the helix, interpolating from a minimum to a maximum magnitude as the points rise on the y-axis.

These vectors are applied to each particle based on the particle's distance from the 'head' point of the vector, with the force on a given particle being dampened by its distance from the force vector.

The cyclone is transformable and continuously rotates at a user-defined rate to simulate particle movement appropriately.

CUDA

CUDA is Compute Unified Device Architecture, an NVIDIA-designed parallel computing platform that is implemented on their newer GPUs. It allows for access to the GPU's multiprocessors, enabling the programmer to process a high volume of identical calculations with different pieces of data very quickly.

The program will make use of this GPU interface by calling several CUDA kernel functions to process all the necessary components of force vector and gravity calculations on each particle in the system and perform the actual update to each particle's location after determining how it is affected in total by the cyclone.

Particle System

The particle system is defined, in its simplest form, by a series of points and velocities. These points will be generated evenly in a bounding box for the particle system at program start, and will be confined to this box over the duration of the simulation. This out-of-bounds check will be performed in a kernel function, most likely in the particle's position-updating kernel function.

The particle system may be altered to spawn particles at the base of the cyclone; in this situation, particles will be given a lifetime and emission rate such that they die near the top of the cyclone and are spawned at a rate fast enough to keep the appearance of the cyclone constant.

Each particle will use a point sprite, variable lifetime coloration, and alpha blending to give the cyclone the appearance of either a dust devil or tornado.

Terrain Collision

The particles are to react to collision with a 'ground' surface similar to that of the fractal terrain generated for our first program. I have not yet determined the best method for this, but am considering using a map containing each point on the terrain sorted by its X and Z components in order to allow a fast lookup of the point on the terrain that each particle is closest to.

This data will either be utilized by limiting a particle's minimum y-value by the nearest terrain point's y-value, or by locating the two nearest points on the terrain and interpolating between their y-values to find a more precise limit for the particle.

Progress

So far, the helix is fully implemented and can be generated based on a number of user-specified parameters for a wide variety of testing conditions.

The CUDA backend for the particle system is nearly complete, and all of the appropriate function calls to make OpenGL interface with CUDA in the necessary manner are in place.

Based on the work I have completed so far, the project could, potentially, be completed at the end of next week. If that is the case, I would like to look further into a volumetric lighting example I have found and possibly attempt to implement volumetric lighting on the cyclone's particle system with a greatly reduced particle count (between 30k-60k particles).