Computation of Global Minimum Separation Distance on Bezier Curves and Meshes Using CUDA

Lauren Bisset, Nicholas Woodfield

University of Connecticut, Storrs, CT 06269
{lauren.bissett, nicholas.woodfield}@uconn.edu

A common problem in computer graphics is the detection of temporal artifacts, which are small errors in animation that are detectable by the viewer, and so interrupt the fluidity of motion. A subset of this problem is the detection of self-intersections or imminent self-intersections, which is helpful not only in preventing this temporal aliasing in animation, but also in scientific visualization, where it is useful in detecting interactions in complex molecules without relying on the error-prone human eye.

The method by which to determine these self-intersections involves calculation of the global minimum separation distance; in plain, it is to find the smallest distance between any two points. The algorithm to compute this has two general components; first to find candidate points for the minimum distance, and then refine these using Newton's method. This current method is slow and impractical; therefore as part of the project the algorithm will be highly parallelized. In doing this, the parallel programming architecture chosen was Nvidia's CUDA programming language, which allows the programming of thousands of simultaneous threads on the GPU (in this case, a Quadro FX 5800 GPU). The use of parallel architecture is anticipated to significantly speed up in particular the Newton's method portion of the algorithm.

The project itself is split into three major milestones. The first milestone involves the computation of the minimum separation distance over Bezier curves in the C programming language; this milestone has already been completed successfully. The second milestone involves porting the existing C code to CUDA. As for 25 June 2009, this step is currently in progress and nearing completion. The final milestone involves expanding the existing algorithm to Bezier meshes and rewriting the existing CUDA code to support it; this is anticipated to be the most complication portion of the project.

Finally, an analysis of the software's performance will be used to determine if the speed up from using CUDA will be significant in the future of using this method to detect self-intersections.