

# GPU Accelerated Particle System for Triangulated Surface Meshes

Brad Peterson\*  
School Of Computing  
University of Utah

Manasi Datar†  
SCI Institute  
University of Utah

Mary Hall  
School Of Computing  
University of Utah

Ross Whitaker  
SCI Institute  
University of Utah

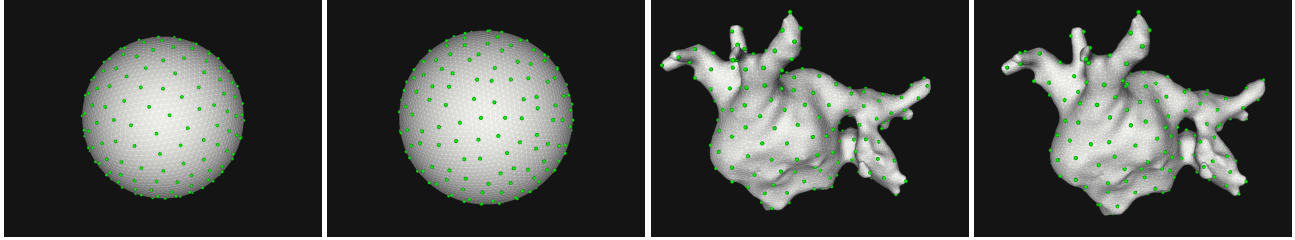


Figure 1: Particle distribution for test shapes using CPU (images 1,3) and GPU (images 2,4) schemes

## Abstract

Shape analysis based on images and implicit surfaces has been an active area of research for the past several years. Particle systems have emerged as a viable solution to represent shapes for statistical analysis. One of the most widely used representations of shapes in computer graphics and visualization is the triangular mesh. It is desirable to provide a particle system representation for statistical analysis of meshes. This paper presents a framework to distribute particles over a surface defined by a triangle mesh and provides an efficient and controllable mechanism to sample shapes defined by triangular meshes within a locally adaptive scheme. We propose GPU implementations to leverage inherent data parallelism. Results on triangular meshes representing synthetic and real shapes demonstrate speedups on the GPU up to 341X as compared to the CPU implementation.

## 1 Introduction

Shape analysis literature has, broadly, two strategies. One strategy is to compare images and quantify the differences in the underlying coordinate transformations that provide good alignments between intensities and shape. The other approach is to represent the shape (curves or surfaces) using salient points or landmarks and to present statistical analysis based on the configurations of these landmarks. Point-based surface representations have emerged as a viable alternative to parametric representations, especially in applications which present topological constraints that make such parametrization inefficient. Such systems can also serve as an efficient surface representation for many applications such as statistical analysis of shape ensembles [Cates et al. 2007; Datar et al. 2009]. The advantage of so-called point-set surfaces is that they do not require a specific parameterization and do not impose topological limitations; surfaces can be locally reconstructed or subdivided as needed.

A related technology in the graphics literature is the work on particle systems, which can be used to manipulate or sample implicit surfaces. A particle system manipulates large sets of particles constrained to a surface using a gradient descent on radial energies that typically fall off with distance [Meyer et al. 2005]. Although triangular meshes have commonly been used in rendering and simulation applications, an increasing number of such meshes are generated by statistical applications and it will be useful to extend the particle system representation described for implicit sur-

faces to these meshes. However, the concept of energy minimization required for computing an optimal distribution of particles on the surface is poorly defined for meshes and proves to be a major hurdle in extending particle systems to meshes. One of the first applications of particle systems to 3D meshes is remeshing to enhance a given 3D structure. [Peyre and Cohen 2006] provides an intuitive algorithm based on the Geodesic distance to construct a particle system for an underlying triangular mesh.

This paper explores a combination of the above ideas to present a particle system representation for triangular meshes based on a minimization scheme that uses Geodesic distances. At any stage, the particles are updated independent of each other and this inherent parallelism suggests a natural extension to the GPU. We explore two such parallel implementations using GPUs to obtain speedups on the order of 300X over the sequential algorithm.

## 2 Methods

Consider a triangular mesh  $M$  with  $V$  vertices. The problem of distributing  $n$  particles on this mesh can be formulated as an energy minimization problem with the following local update as the solution for each particle  $p_i$ , with the assumption that all  $n$  particles are initially placed at  $v_0 \in V$ :

$$p_i^{new} = \{v | v \in neighbors(p_i), energy(v) = \min\{energy(neighbors(p_i))\}$$

where,  $p_i^{new}$  is the new position of particle  $p_i$  and  $v$  is a vertex in neighborhood of  $p_i$ . The current implementation constrains the neighborhood of a particle to vertices directly connected to the particle by an edge (one-ring). At each iteration, energy at a candidate vertex can be defined as the accumulation of Gaussian forces exerted on the vertex by particles around the vertex. For example, the Gaussian energy can be defined as the following accumulation:

$$energy(v) = \sum_{i=1, i \neq c}^n e^{-\frac{d(v, p_i)}{2\sigma^2}}$$

where  $d(v, p_i)$  is the Geodesic distance between the vertex  $v$  and each particle  $p_i$  except the particle  $p_c$ , whose position is being updated. This process is repeated for a user-specified number of iterations.

### 2.1 CPU Implementation

The CPU implementation of the above algorithm is a sequential code where the Geodesic distances are precomputed using an extension of the method [Jeong and Whitaker 2008], implemented

\*e-mail: bradpeterson@gmail.com

†e-mail: datar@sci.utah.edu

in [Zhisong and Whitaker 2009]. These pairwise distances for all vertex pairs in the mesh are stored in a 2D distance map (`geo[][]`). It is important to note that this map stays constant throughout the particle position update process since we restrict particles to lie on vertices, and do not introduce any new vertices at any point. The pseudo code for the CPU algorithm is given below:

```
do {
  for each particle from 1 to n {
    create list of candidate vertices
    for each vertex in list {
      for each particle except current {
        accumulate energy
      }
    }
    move current particle to
    vertex with minimum energy
  }
} until convergence
```

There are no dependences crossing loop iteration boundaries and each particle interacts with its own neighborhood. On the other hand, energy at each vertex is computed using the current stable positions of particles (note that the particle being updated is excluded from the energy accumulation). As such, parallelism can be obtained at each loop-level in a hierarchical manner. It is clear that the CPU time is almost quadratic in the number of particles and linear in the size of the neighborhood defined. Considering that meshes from real applications have tens-of-thousands of vertices and conventional statistical shape analysis models use thousands of particles, the scalability of the CPU implementation falls apart. Moreover, extending this method to compute a complete statistical shape model will involve particle system optimization on each shape in a given ensemble, while optimizing correspondences within particles across shapes. In such an extended system, GPU acceleration will prove extremely beneficial. Seeking numbers from an equivalent system working with implicit surfaces, as reported in [Datar et al. 2009] we can see that optimization can take on the order of hours for real data.

## 2.2 GPU Implementation

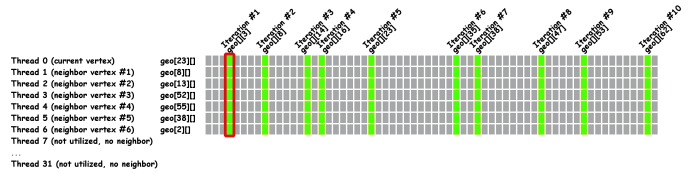
Triangular meshes present unique challenges in terms of data organization in memory and the problem of particle position optimization adds unique computational challenges of its own. This section discusses the challenges of mapping this problem to the GPU along with solutions unique to the problem at hand, and also provides details of the corresponding CUDA implementation.

A naïve CUDA implementation of the CPU algorithm will involve placing `geo[][]` into global memory and parallelizing the particle updates at each iteration. The pseudocode for such an algorithm is listed below:

```
__shared__ energy[7]; //Threads = neighbors
if (threadIdx.x < numNeighbors) {
  candidateVertex = neighbors[neighborsIndex
    [currentParticleVertex] + threadIdx.x];
  for (i = 0; i < numParticles; i++) {
    // skip current particle as it is moving
    if (currentParticleIndex == i) continue;
    accumulatedEnergy += exp(-0.5f*
      (geo[particles[i][candidateVertex]
        /sigma_squared));
  }
  energy[threadIdx.x] = accumulatedEnergy;
  __syncthreads();
  //Update = candidate with min. energy
}
```

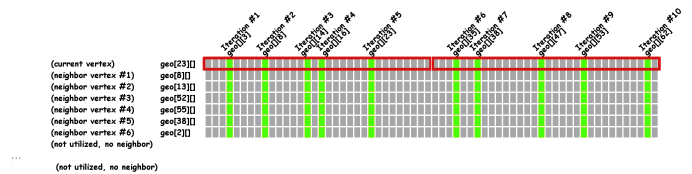
This algorithm exploits parallelism, but suffers from limitations related to memory coalescing and thread occupancy. The rest of the section provides insight into each of the challenges above and discusses potential solutions.

**Memory Coalescing.** Fig. 2 provides a visualization of the problem with an example situation from a mesh that has 10 particles, one of which is placed on vertex 23 and has 6 neighbors. In the naïve scheme, for every particle a thread is assigned to a



**Figure 2:** Visualization of memory layout for particle update on example mesh

candidate vertex and accumulates the energy at that vertex. The columns represent iterations. The rows represent arrays in memory. Note that the rows have been laid out figuratively and may not be contiguous in memory. The red box in Fig. 2 shows the situation for the given example on the first iteration. The “active” threads must access 7 different pieces of global memory. More specifically, these threads access locations contiguous along the second dimension of the `geo[][]`. Since the data is stored in row-major fashion, there is no coalescing. The optimized algorithm changes the order of memory accesses to a row-major fashion by allowing accesses contiguous along the first dimension of the `geomap[][]` structure. With the memory coalescing features in Compute Capability 1.2, this improves memory coalescing. Fig. 3 shows the pattern of memory accesses for the proposed improvement for compute capability 1.2. Using this scheme, global memory requests can be coalesced and data can be staged into shared memory. This scheme can reduce the time required for global memory fetches despite the lack of data reuse in shared memory.



**Figure 3:** Memory access pattern for improved GPU implementation on 1.2 capable devices for given example

**Thread Occupancy.** The naïve algorithm is limited by the number of active threads by design and leads to bottlenecks while accumulating results at the end of an update. Such a sparse population of active threads hinders the throughput of the streaming multiprocessor. The re-ordering of memory accesses also allows for the ability to tile the algorithm, which leads to increased occupancy of active threads in a streaming multiprocessor, consequently leading to faster performance.

Further optimizations were gained by (1) placing arrays which hold index information for other arrays into texture memory (2) limiting loop arithmetic calculations by unrolling and merging loop registers, and (3) tweaking the algorithm by testing for the optimal permutation of threads per block and tile size.

### 3 Results and Discussion

This section describes experiments designed to illustrate and validate the proposed method. First, we present an experiment with synthetically generated sphere and torus meshes to illustrate the applicability of the method. Next, we present an application to a triangular mesh generated as part of a statistical study of the human heart, specifically the left atrium. All CPU results were generated on a Linux machine with an Intel T6600 CPU with 4GB RAM, while the GPU implementations were run on an NVidia GeForce GTX 260 graphics card with 27 SMs (216 cores) and 876MB global memory.

**Synthetic Data.** Synthetic shapes of a sphere and a torus were generated using analytic methods and triangulated using an FEM meshing tool to generate meshes at resolutions commonly seen in practice. For the sphere, particle systems generated by the CPU and the naïve GPU algorithm are shown in images 1 and 2 of Fig. 1 respectively.

Table 1 documents the timing results for different number of particles for both the synthetic shapes. The time reported is the average time per iteration, computed over 1000 iterations. Each iteration includes an update for all particle positions in the system. Both the naïve GPU scheme (GPU1) and the optimized 1.2 capability scheme (GPU2) were tested. It is clear that the GPU versions scale well, and are clearly superior to the CPU implementation as the number of particles is increased. These timing results indicate a speedup in the range of 88X-176X for the sphere and 80X-128X for the torus over the CPU implementation. This can be improved further by optimization to specific GPU platforms.

sphere2 (~ 4k vertices)	#particles			
	128	256	512	1024
CPU	0.0177	0.0768008	0.313606	1.35615
GPU1	0.00020	0.00072	0.00222	0.00768
GPU2	0.00020	0.00046	0.00110	0.00397

torus (~ 1.5k vertices)	#particles		
	128	256	512
CPU	0.0153	0.0644005	0.287706
GPU1	0.00019	0.00073	0.00224
GPU2	0.00017	0.00038	0.00097

**Table 1:** Timing performance for synthetic shapes (avg. time over 1000 iterations)

**Real Data.** The proposed particle system implementations were further applied to a mesh representing the left atrium of the human heart. This mesh was generated using an FEM meshing tool and had ~10500 vertices. Images 3 and 4 in Fig. 1 depict a comparison of the particle distribution generated by the CPU and naïve GPU implementations respectively.

The distributions of particles generated by the CPU and GPU implementations are visually very similar. This is encouraging given that the parallel GPU implementation outperforms the sequential CPU implementation by several orders of magnitude in time. Table 2 provides details of the timing performance of the proposed implementations for real data. GPU acceleration provides speedups in the range of 100X-160X for the tested configurations.

LA (~ 10.5k vertices)	#particles		
	256	512	1024
CPU	0.0737007	0.309306	1.24028
GPU1	0.00073	NA <sup>1</sup>	0.00762
GPU2	0.00047	NA <sup>1</sup>	0.00457

**Table 2:** Timing performance for real shape (avg. time over 1000 iterations)

### 4 Discussion

This paper describes a particle system representation for shapes given by triangular meshes, which is further extended into a parallel algorithm and accelerated using GPUs. Speedups upto 341X were observed in tests on synthetic and real data. While the speedup obtained using parallel GPU implementations is desirable, it would also be interesting to compare the particle distributions generated by the various implementations. For synthetic shapes, parametric distribution of particles on the shape could be considered the gold standard and lead to a scheme for quantitative analysis of the particle distributions generated by various implementations. This work is beyond the scope of the current project, but will be an interesting avenue for future work.

Results on synthetic and real meshes indicate that the method can be applied to numerous topologies and provides particle system representations which can be used in statistical shape analysis applications. Such applications can further be used to study populations of meshes to derive variability and group statistics. Further work includes validation of mesh quality for individual shapes and an extension to population analysis similar to [Datar et al. 2009].

### References

- CATES, J., FLETCHER, P., STYNER, M., SHENTON, M., AND WHITAKER, R. 2007. Shape modeling and analysis with entropy-based particle systems. In *Information Processing in Medical Imaging (IPMI)*, LNCS, no. 4584, 333–345.
- DATAR, M., CATES, J., FLETCHER, P., GOUTTARD, S., GERIG, G., AND WHITAKER, R. 2009. Particle based shape regression of open surfaces with applications to developmental neuroimaging. In *Medical Image Computing and Computer Assisted Intervention*, MICCAI.
- JEONG, W.-K., AND WHITAKER, R. 2008. A fast iterative method for eikonal equations. *SIAM J. Sci. Comput.* 30, 5, 2512–2534.
- MEYER, M., GEORGEL, P., AND WHITAKER, R. 2005. Robust particle systems for curvature dependent sampling of implicit surfaces. In *In Proceedings of the International Conference on Shape Modeling and Applications (SMI)*, 124–133.
- PEYRE, G., AND COHEN, L. 2006. Geodesic remeshing using front propagation. *Int. J. Comput. Vision (IJCV)* 69, 1 (Aug), 145–156.
- ZHISONG, F., AND WHITAKER, R. 2009. Solving eikonal equations on triangulated surface mesh with cuda. In *NVIDIA Research Summit*, Poster 47.

<sup>1</sup>possible data processing error from the meshing tool prevented accurate timing measurement for 512 particles