Particle System with High-Level Elements

Jonathan W. Decker*
University of Maryland, Batimore County

Abstract

We present a particle system with simple mass spring elements and propose a complete GPU implementation of the system in shader programs. In this system, there is a large number of small meshes we refer to as molecules. Each particle is moved independently according to external forces that model fluid, then restoration forces are applied onto every particle along the springs incident on it. The end result is a large set of objects noticeably aligned with their local flow and interacting with the environment.

Keywords: graphics hardware, particle systems, mass spring, vector fields, fluid simulation

1 Introduction

Particle systems are often used to create effects where group motion is more important than the appearance of individual objects. However, there are many examples of phenomena where individual objects are important as well, such as wind-tossed leaves, or schools of fish. It is in these instants that single particles, even when drawn as more complex primitives than point sprites, such as quads, do not convey the subtle nature of fluid motion. A solution would be to allow multiple particles to represent a single element.

We propose a particle system where each individual element is actually a series of particles that move together as one, and can be rendered together as complex shapes. We will refer to these particle structures as molecules. Each molecule is a simple mass spring system that can interact with height maps and each other. We feel that provides a large visual advantage over simple particle systems. As the molecules are deformed and restored according to the forces acting on them, they align themselves in the direction of these forces. Thus the pattern of movement is instantly recognizable, whereas in pure particle systems with textured point sprites, the direction of the motion cannot be discerned locally. Even when the elements in a particle system are given angler motion, there is then only two degrees of freedom over the look of the primitive. However, in our system, using molecules of the appropriate complexity can allow for an arbitrary level of expressiveness.

The particle system will be almost entirely implemented using shaders, which will involve collision detention with elastic forces, fluid motion, particle management and particle sorting for alpha blending. An interface has been developed to allow users to apply forces on the molecules, change the quantity of molecules in the system, as well as the complexity of the molecules.

The remainder of the paper is divided into the following sections: Section 2 discuses related work, Section 3 describes the details of our implementation, Section 4 provides our results, and in Section 5 we discuss future work and the conclusions we have drawn.

*e-mail: j6@umbc.edu

2 Related Work

A particle system are used to simulate a wide variety of complex phenomena not easily represented with geometry and texturing, and their use for model various effects was first explored by Reeves [Reeves 1983]. More recently particle systems have been researched which implement all of their operations in fragment programs on the GPU, thus allowing systems to iterate over a large number of particles [Kipfer et al. 2004] [Kolb et al. 2004].

Related to particle system are mass spring systems. A mass spring system is one where the individual elements under consideration are meshes that are allowed to deform. The overall shape of a mesh is maintained using restoration forces from elasticity theory, bought into the field of graphics be research such as Terzopoulos et al. [Terzopoulos et al. 1987]. Terzopoulos et al. concludes in their work that the dynamic nature of this technique combines the concepts of shape and motion, which is essential to our work where molecules are meant to emphasis motion.

Just as particle systems have been implemented on the GPU, so have mass spring systems. The Nvidia implementation of cloth simulation, particles positions are updated according to collisions in one pass, and then two further passes apply restoration forces [Zeller 2005]. Cloth geometry is then repositioned using a vertex shader and vertex texture fetch (VTF). This simulation is only of flat meshes, but other implementations of mass spring system on the GPU have been created to handle 3D meshes as well as 2D [Georgii and Westermann 2005] [Echtler 2004].

Although every molecule in our system is a mesh with elastic constraints, we feel it is more appropriate to describe our system as a particle system rather than a mass spring system, since at a high level all molecules are identical and only together do they form the complete picture. In mass spring systems, objects can have very different make up and the implementation must account for this generality.

Navier-Stroke equations are useful for simulating stable fluids [Stam 1999]. These equations have been implemented on graphics hardware for a number of real-time applications such as cloud [Harris et al. 2003] and smoke [Fedkiw et al. 2001] simulation. These methods generate flow fields, which are discrete volumes of vectors. To be able to determine the force acting on a particle at any position, some type of interpolation is needed or the motion will be choppy.

However, even with a GPU implementation, this method is expensive and yields only interactive rates for 3D fluid fields [Harris et al. 2003]. Since our goal with this research is to produce large numbers of high-level primitives with realistic motion at real-time rates, we will settle for a less computationally complex approach. One such method is to use measured or precalulated vector fields. When used in this manner, our method can be seen as a variation of a fluid flow visualization technique that simulations the movement of a small quantity of visible material through a field. Telea and Wijk present a GPU accelerated method that insertions and advects a dye through the field [Telea and van Wijk 2003]. Similar in concept,

Kruger et al. utilize the GPU to cast millions of a particles into a fluid flow at real-time rates [Jens Krüger and Westermann 2005].

3 Implementation

The following is an outline of the implementation section of the paper. First we describe our general framework for processing positions with the GPU. Next we describe how molecules are represented and what types of external forces we have applied to them. Then we how elasticity forces are used to restore molecule shape.

3.1 GPU Implementation

Graphics hardware is used to initialize and manipulate particle positions, as well as render them directly from texture data. Positions and velocities are stored in textured that are that are bound to screen-aligned quads and then processed in a fragment shader. The output is rendered to a texture that is then used as input in the next pass. In this way, different operations can be done on the particles, such as initialization, movement based on external forces, elastics forces between particles in each molecule and sorting.

When position data is stored in a texture on the GPU, as we have done in this work, there needs to be treating the texels as vertex data. For this, Kobi et al. and Kipfer et al. use superbuffers, which allow control over whether they represent texture data or vertex data [Kipfer et al. 2004] [Kolb et al. 2004]. This method is specific to the OpenGL API. Another method, used by Zeller for the nVidia GPU-accelerated cloth demo, is to use Vertex Texture Fetch (VTF), support by vertex shader 3.0 [Zeller 2005]. First, geometry is defined and initialized to some arbitrary position, such as the origin. Then this geometry is rendered with a vertex shader that resets the positions and normals of the vertices to values stored in textures. This method can also be used in OpenGL using the ARB_vertex_shader extension [Kolb et al. 2004]. In our implementation we use VTF, but either approach would work with the rest of our system.

Figure 3 provides a visualization of how the vertex texture fetch works. In our system, each vertex of within each molecule is positioned at some arbitrary location, possibly the origin. Along with its position, normal, and the texture coordinates, each vertex has an extra set of texture coordinates where it can look up its attributes within the position and normal textures. When the geometry is rendered, it is sent through a vertex shader where each vertex sets its position and normal to these values.

3.2 Molecule Construction

Each molecule is a set of unique vertex positions that when connected with the correct geometry form some shape. Figure 1 shows several examples, as they would be drawn by triangles. Although there are no restricts on the shape of the molecules, increasing their complexity greatly reduces the possible number of molecules that can be manipulated and rendered in real-time. Furthermore, our collision method is based on particle intersection, and not triangle intersection. Thus, for our tests we have chosen molecules that are small and dense so that they do not commonly go through objects in the scene.

Since the unique vertex list, or vertex buffer, for each molecule always as the same order, index buffering can be used to render them. Furthermore, since every molecule in the system is the same, we can use a single vertex buffer to store the vertices and a single index buffer to dictate what triangles will be drawn.

Each vertex stores a position, a normal and a texture coordinate.

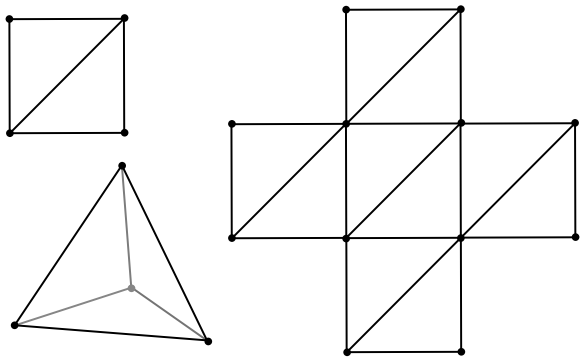

Figure 1: Triangle layout of several molecules.

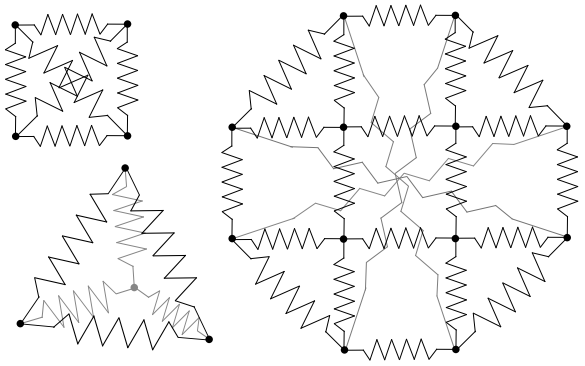

Figure 2: The springs of the example molecules shown in Figure 1. Notice that springs need not correspond to unique triangle edges, and can be defined according to the desired effect.

The texture coordinate is chosen at initialization and does not change. The position and normal must change according to the movement of the molecule.

The particles in our system are the vertices of the molecule constructs, so in the following text we use the words *vertex and* particle interchangeability.

3.3 External Motion

There is a wide variety of external forces that can effect the particles, depending on the desired motion. Predefined volumetric vector fields stored in 3D textures allow use to simply look up the force direction for a given molecule. However, this method of a look-up into a single grid cell of the volume may not be enough if the flow varies greatly from cell to cell. This would cause particles to switch direction sharply, creating unrealistic motion. To avoid this, we should find all grid cells surrounding a particle and move it by an average force.

Another method that is much less expensive in terms of storage is to move a particle by a function of time and its previous position.

3.4 Elasticity Forces

Once the particles of a molecule have been moved according to external forces, restoration forces need to be applied in order that the molecule sustain shape. To this end we use Hooke's law, which maintain an initial distance between the particles within each molecule. These springs are defined between certain pairs of par-

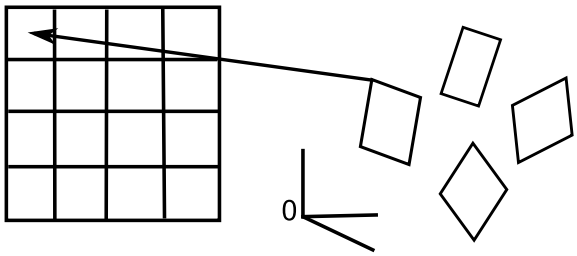

Figure 3: This diagram shows the idea behind a vertex texture fetch. The vertices in the scene are fixed at arbitrary values (for example, the origin). They retrieve their actual position and normal from textures, using texture coordiates provided to them at intialization.

ticles (Figure 2). Every pair of particles need not have a spring between them, and in fact the placement of springs depends on the desired motion of the molecule. The springs maintain their size by applying forces to the particles they are incident on. The force on a particle along a spring is the product of a stiffness constant and the signed difference of the length of the spring from its initial length. The particle is thus moved in the direction of the other particle attached to this spring by the force.

We did not implement Elasticity Forces in our GPU implementation, but we propose the following method for doing so. Currently, on the CPU, we store the spring information as an array of floating vectors. This could be easily stored in a constant buffer within a shader. We store the spring adjacency information for each molecule vertex as an array of indices into the spring array. These could also be constant buffers, since there will only need to be a small number, given the relative small size of the molecules. In this way we can implement any molecule shape with any spring configuration on the GPU.

3.5 Molecule Normals

To be able to correctly light the molecules when they rendered, we will need to calculate their normals after the vertex positions have been moved. This can be done simply by traversing the position texture in a pixel shader and determining the sum of the triangle normals adjacent to each vertex. This adjacency information can be hard-coded into the shader or stored in constant buffers. This requires some computational redundancy since the triangle normals are computed multiple times, but this is unavoidable given the parallel nature of the hardware.

4 Results

The following section demonstrates our implementations with the use of images. It is broken up into two parts, one for our cpu implementation and the other for our gpu implementation. The performance results given where obtained from testing done on a PC running Windows XP with a Intel Pentium D CPU 3.2 GHz and a NVIDIA GeForce 7900 GT/GTO.

4.1 CPU Implementation

We implemented our molecule system on the CPU with two types of molecules, quad shaded and cross shaded (See Figures 1 and 2). In Figure 4, textured cross molecules are manipulated by a single timestep of the wind speeds of Hurricane Bonnie. The motion of the molecules within a volume has the same properties as the motion dye through such fields, as has been proposed in previous work for

Figure 4: In this image we see the cross shaped molecules in action. Each of these molecule is made of 10 triangles, and even from a still image some characteristics about the nature of the vector field can be observed.

Figure 5: Each of these simple quad molecules has been positioned on random vertices of the smallest branches of this tree model. Then, the bottom two vertices have been set so that they cannot move. The resulting effect is that the leaves are attached to the tree, but still align themselves with the direction of the wind.

visualizing volumes. However, simulating dye spreading through a volume would not be able to reveal the overall direction of the field in a still image. However, it is clear from both Figures 4 and 5 that our molecules system is more than capable of conveying volume shape within still images.

We also experimented with attaching our system to fixed objects in a scene. In Figure 5, we have initialized quad molecules to random vertices on the smallest branches of a tree model. We then fixed the bottom two vertices of each quad, so they they are updated by neither external or elasticity forces. The result are leaves fixed to the tree at the stem, but aligned with the direction of the wind.

Given the expressiveness of our system, we wish to show that it is viable for real-time applications. At 55 frames per second, we were able to manipulate and render 300 of cross molecules, or 1500 quad molecules. Given the complexity of these molecules and the expressiveness of each, this is a fair number, although it could improve a great deal using a efficient GPU implementation.

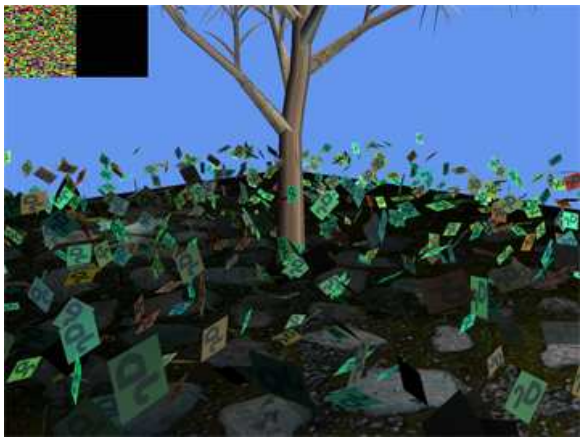

Figure 6: This image is from our GPU implementation. Here, vertices have been positioned and lit using the positions and normals in the textures in the upper left.

4.2 GPU Implementation

Our GPU implementation proved to perform worse then our CPU implementation, even without the proposed elasticity forces and normal calculation passes. 1024 particles only update at 20 frames a second, whereas they would run at an unhindered 60 frames per second with the CPU implementation. We will explain this in the conclusion section. This is not because of the overhead of using vertex texture fetch. When the molecules are not being updated per frame, thousands of molecules can be rendered to the screen at an unhindered 60 frames a second.

5 Conclusions and Future Work

Given the poor performance of our GPU implementation in comparison with our CPU implementation, it is clear our approach is inefficient. We determined after inspection that the reason for the loss in performance is that our implementation is not handling the textures efficiently. Instead of rendering using textures on the GPU and rendering to textures on the GPU, we are sending and receiving texture data across the bus several times each frame. In future work we would like to correct this implementation so that texture data are left on the GPU after it is first initialized, and manipulated only indirectly by the CPU. After we have done this we hope to achieve the performance gains from previous GPU accelerated work in this field.

We feel that this paper is merely proof of concept. We wish to extend it in order to create an advanced molecule system that could be used within larger systems without hindering them. Once we have improved the efficiency of our GPU implementation, examples such as the one shown in Figure 5 will be far more impressive, since many more leaves are required to cover a tree than our CPU implementation can handle. We would also like to use more complex molecules for the leaves which will be more expressive, and implement more algorithms for conveying wind motion. The results in the figure to do not animate in a convincing way since the vector field used has fixed forces at each cell. Using noise we would achieve some of the complexity that one would normally see in the subtle motion of leaves on a tree. We could also explore some locally evaluated equation methods, such as Fournier and Reeves have used to simulate realistic ocean waves [Fournier and Reeves 1986]. Given a depth value at each vertex, we can determine the position of that vertex as part of a wave. This has been used in cloth simulation and could also be utilized in simulating wind.

Furthermore, we would like to implement collision response into our application. This could include collision with terrain and fixed objects using depth maps, as well as self collisions between molecules and triangles within molecules. Should features would further cement our system as part of a virtual environment, where the individual molecules interact with the scene.

6 Acknowledgments

We would like to thank Alark Joshi for his support and providing us with the Hurricane Bonnie data.

References

ECHTLER, F. 2004. *Efficient Realization of Mass-Spring Systems on Graphics Hardware*. PhD thesis, University of Munchen.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 15–22.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 75–84.

GEORGII, J., AND WESTERMANN, R. 2005. Mass-spring systems on the gpu. *Simulation Modelling Practice and Theory 13*, 693–702.

HARRIS, M. J., BAXTER, W. V., SCHEUERMANN, T., AND LASTRA, A. 2003. Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 92–101.

HARRIS, M. 2004. *GPU Gems: Programming Techinques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley Professional, ch. 38, 637–665.

JENS KRÜGER, PETER KIPFER, P. K., AND WESTERMANN, R. 2005. A particle system for interactive visualization of 3d flows. *IEEE Trans. Visualization and Computer Graphics 11*, 6, 744–756.

KIPFER, P., SEGAL, M., AND WESTERMANN, R. 2004. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 115–122.

KOLB, A., LATTA, L., AND REZK-SALAMA, C. 2004. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 123–131.

LANDER, J. 1998. The ocean spray in your face. *Game Developer 5*, 7 (July), 11–16.

REEVES, W. T. 1983. Particle systems - a technique for modeling a class of fuzzy objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 359–375.

STAM, J. 1999. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128.

TELEA, A., AND VAN WIJK, J. J. 2003. 3d ibfv: Hardware-accelerated 3d flow visualization. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, Washington, DC, USA, 31.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 205–214.

ZELLER, C., 2005. Nvidia sdk white paper : Cloth.