Ticker Tape: A Scalable 3D Particle System with Wind and Air Resistance

Introduction

You sneak down the hall and pause just outside the office door, where two enemy guards wait inside. You pull the pin on a grenade and toss it into the room. Blam! After the blast, you enter the office and find the guards dead, but something just isn't right. The papers stacked on the desks survived the blast without any dire effects. Shouldn't pieces of paper be swirling through the air? We certainly think so. So we created Ticker Tape, a particle system with 3D orientation, air resistance, and wind effects. You can see a video of Ticker Tape running, download the free code, or download the executable binary at www.intel.com/software/tickertape.

With microprocessors rapidly increasing in the number of cores, an enormous amount of compute power is available for games. Ticker Tape can use spare processor cycles to simulate ambient particles that interact with the environment. We were especially interested in how lightweight particles, such as paper or leaves, could better interact with air resistance and wind.

In this article, we discuss our approach to simulating 3D particle behavior and the modular design we used that encourages reuse and experimentation. Additionally, we look into the performance gains achieved through threading and use of Intel® Streaming SIMD Extensions (Intel® SSE).

Figure 1. Ticker Tape demonstrating falling leaves.

Modularity

A primary goal of this demo was to make it simple for developers to experiment with and integrate the effects into their own projects. Offloading necessary computation to properly designed middleware, an API, or a library is always welcome. The modular design of the Ticker Tape library makes it straightforward for developers to take the library and drop it

into their code. The library also provides hooks so the main code of the application can manipulate the particles through wind and collision callbacks. Because the library uses the application's existing thread pool, the application controls the threading aspects of the library; in fact, the library has no knowledge of the application's threading mechanism. We'll explore threading in more detail below.

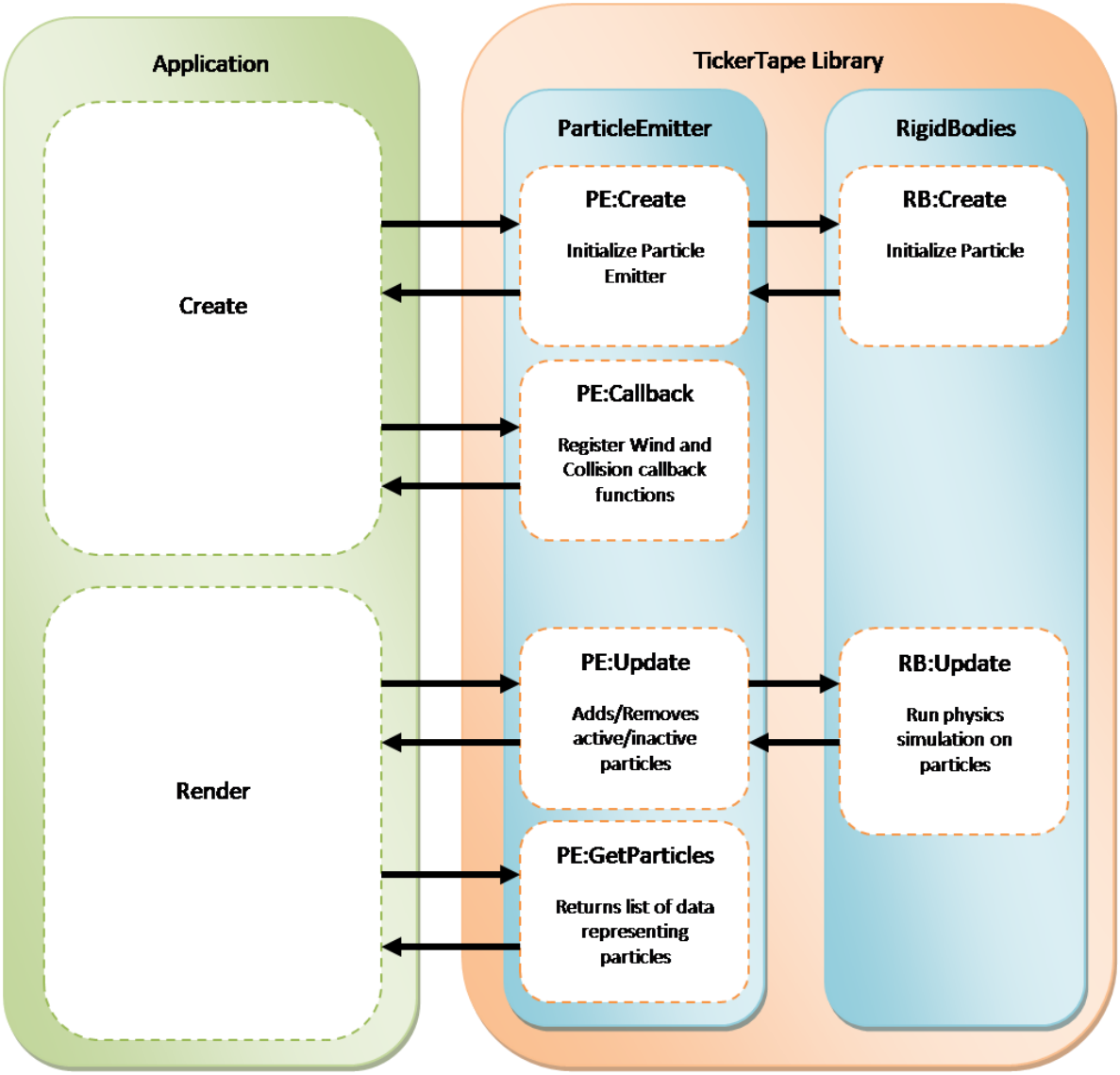

Figure 2. Diagram of Ticker Tape library and application interaction.

The wind and collision callbacks allow the application to manipulate the particles without managing them. In the demo, we have created a tornado-style wind, which picks up the particles and tosses them around in a vortex shape. The code for the wind is in the application and is called by the library through a callback. This allows the application to manipulate the particles after they have been initialized.

Collision works in a similar fashion. We integrated a height map for the demo and implemented simple collision with the ground. This could be extended to a more advanced

collision detection with the library callbacks. The callback also allows the Ticker Tape library to be easily integrated into the application's physics system, such as Havok*.

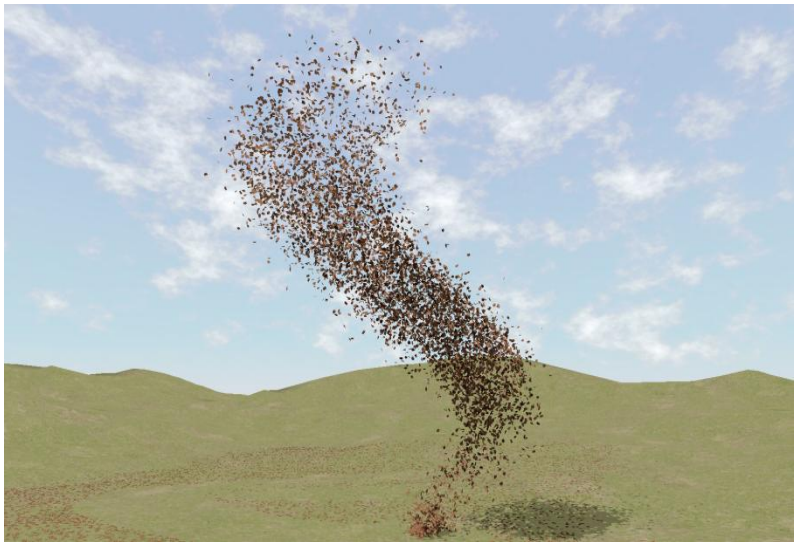

Figure 3. Particles in a tornado.

Simulation

The behavior of the particles in Ticker Tape can take one of two paths: a pre-computed behavior or a physics behavior computed in real-time (a path that is affected by forces). The pre-computed behavior is further split into two modes: a flutter behavior and a tumbling behavior[1].

- **Flutter**. The particle floats in the air moving back and forth with a pendulum-like effect.
- **Tumble**. The particle performs barrel rolls and falls in a straight line.

Particles on the pre-computed path cannot switch at runtime between the flutter and tumble modes. However, the particles have the ability to transition to and from the physics mode where forces (wind) can be applied to alter the particles' behavior. The mixture of the paths and behaviors in the demo makes the particles' behavior appear more realistic.

The two pre-computed behaviors are strictly artistic and are taken from observations of how a piece of paper behaves as it is falling. The algorithms for determining the various forces on the particles were derived from the "Beginner's Guide to Aerodynamics" available on NASA's Web site.

The hooks that are present allow the user to implement wind effects to manipulate the particles through the library. When the wind affects a particle, the particle will move in the direction stated by the wind force and continue its movement while forces are interacting with it.

The physics behavior takes wind, lift, drag, and torque into account when calculating movement. The particles are represented as quads, both visually and in the physics simulation. This minimizes the number of vertices processed by the graphics card and reduces the amount of data processed in the simulation. However, each particle also stores

four normals, one for each vertex. Typically, this would be unnecessary since one normal would suffice for a quad or could even be calculated when needed. However, by storing the normals separately, they can be manipulated to make the object behave as if it were curved.

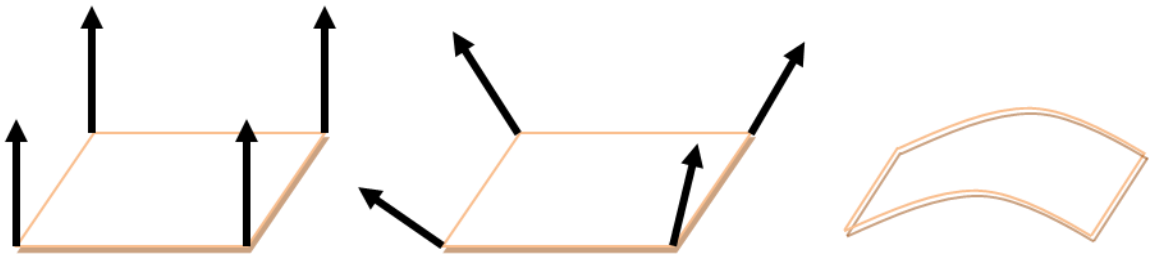

Figure 4. Splaying the normals creates a curved object.

Threading

The only way to make full use of any modern processor is to parallelize the application. Depending on the type and variety of work, breaking an application into parallel pieces can be very difficult. However, particle systems are very easy to parallelize because they comprise of many independent objects that can be grouped together in any way and processed in any order.

Even though Ticker Tape is basically "just" a particle system, there are some interesting aspects to how threading was implemented. In addition to simply highlighting the benefits of doing work in parallel, we wanted to demonstrate multiple systems using a shared task manager and the performance boost that can occur when the parallel work is done in conjunction with the rendering.

When using parallel middleware, there is a danger that the application can end up with multiple systems competing for processor resources. A program may have a well-threaded physics system, particle system, and AI system, each creating a number of threads equal to the number of cores. This can cause oversubscription and contention. With these three systems, there would be three times as many threads as processor cores. For a more detailed examination of this behavior, please see "Getting the Most From Your Middleware."

There is also the issue of "doing parallel work in parallel, not serially." What does this mean? In a threaded application a lot of the work is done in parallel, and then some of the work is done in serial. Typically, the serial work is rendering, which can be a substantial portion of work in the frame. Figure 5 shows the flow: First we set up the data needed for the parallel work, next we do the parallel work and then wait for the parallel work to finish (indicated by the red box). The wait is necessary because the results of the parallel work are used to render the scene. Once the wait is over, the results are copied into a vertex buffer.

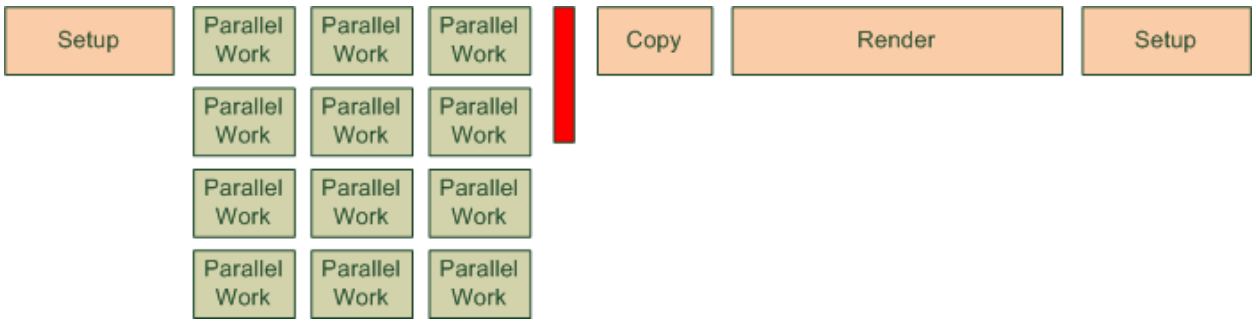

Figure 5. Doing parallel work in serial.

The way to improve this process is to do the parallel and serial work at the same time. Obviously, when doing this the serial work cannot use the results of the parallel work, as is shown in Figure 5. Instead, the serial work uses the results of the last time the parallel work was done, as shown in Figure 6. Here, the parallel work being done is for the next frame's render. When implemented in this fashion, a 1.26x performance improvement is achieved.

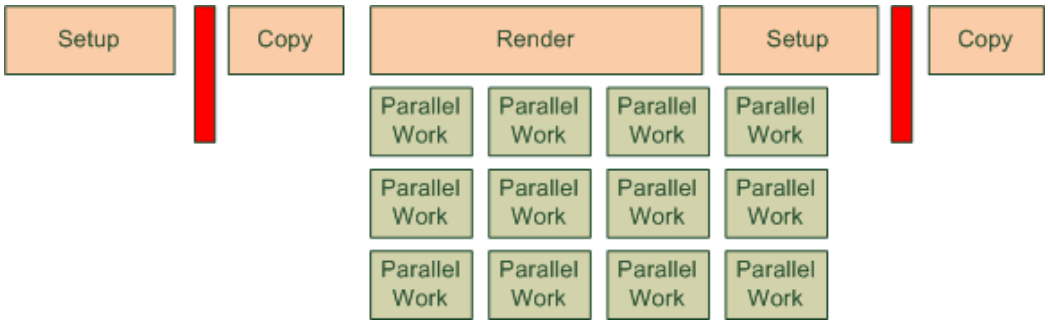

Figure 6. Doing parallel work in parallel.

Performance

The key to performance in the Ticker Tape demo, as with any modern application, is the ability to take advantage of all available processor resources. As discussed in the preceding section, the primary way to do this is by threading the computationally intensive portions of the application. The data in Table 1 was obtained from testing done on an Intel® Core™ i7 processor with Intel® Hyper-Threading Technology (Intel® HT Technology) on two workloads simulating 20,000 particles. One workload is the application with no wind and the other has wind enabled.[2]

[2] Testing was completed on an Intel® Core™ i7 processor-based machine running at 3.2 GHz with 6 GB of RAM using an NVIDIA GeForce* 9800 GX2 graphics card.

	2 cores	2 cores + SSE	4 cores	4 cores + SSE	4 cores + HT	4 cores + HT + SSE
No Wind	61	77	117	150	155	195
Wind	43	53	86	105	119	142

Table 1. Frames-per-second benefits of threading and Intel® Streaming SIMD Extensions.

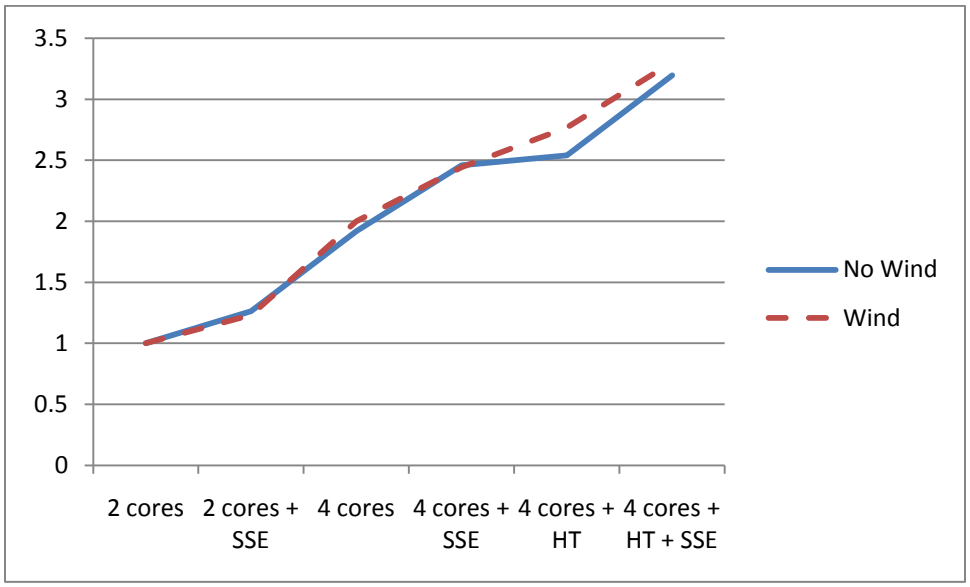

Figure 7. Scaling benefits of optimizations versus two cores.

As Table 1 shows, scaling from two to four cores results in just under a 2x performance gain. This doubling of performance corresponds with a doubling of processing resources. Going from four to eight threads, we see a smaller performance gain of about 1.3x. This is expected because these additional four threads are logical (not physical) cores provided by Intel HT Technology. The virtual cores share resources with the physical cores and as a result do not provide the same amount of processing improvement as a full physical core.

The performance considerations also included the use of Intel SSE instructions. We did some profiling of the application and found that the majority of the time was being spent in the function Newton(), responsible for calculating the effect of various forces on the particles such as wind, drag, and lift. These calculations are all computed four times, one per corner for every particle. The function was changed to use Intel SSE instructions instead of scalar instructions.

The advantage to this approach is that each Intel SSE operation computes four operations at once instead of just one. As Figure 7 shows, using Intel SSE gives a good boost to application performance with the specified numbers of cores. On eight cores we obtained 1.26x without wind and 1.19x with wind (The wind function was not optimized for Intel SSE). With the introduction of the Intel® Advanced Vector Extensions instruction set, eight concurrent operations will be possible for even greater speedup. The second and third articles in this series look at the process of optimizing this demo with Intel SSE.

Future Work

The code could be modified and extended in several interesting ways. A first step to creating more interesting behavior might be to enable particles to switch between the flutter and tumble pre-computed movement paths. Another possible enhancement is to make the particles actually bend and fold. Particle movement could be enhanced with different wind patterns and by letting dynamic objects directly affect the particles. The cascaded shadow maps could also be optimized to create a more realistic scene. We believe there are many other possibilities and hope that developers will take the code and run with it.

Conclusion

Ticker Tape illustrates a method to use extra CPU computational power to produce a nice looking effect without heavily impacting graphics. The code is available to download at http://www.intel.com/software/tickertape and can be freely used and distributed. We hope that game developers will use, modify, and extend the work. We can't wait to hear and see what you do with it!

About the Authors

Mike Yi is a software engineer in the Intel Visual Computing Software Division. He focuses on creating samples that help developers achieve better scaling and performance on Intel® architecture.

Quentin Froemke is a software engineer with Intel's Visual Computing Software Division where he works with the game industry to more effectively utilize multicore and other Intel® technologies.

References / Resources

[1] Belmonte, Eisenberg, Moses. "Flutter and Tumble: The Physics of Falling Paper." Physics News Graphics. <http://www.aip.org/png/html/paper.htm>. 6/15/09

"Beginner's Guide to Aerodynamics" < http://www.grc.nasa.gov/WWW/K-12/airplane/bga.html> 11/11/09

http://software.intel.com/en-us/avx