Interactive Tornado Simulation

Advanced Computer Graphics 2010 Final Project

Nick Coppola and Lincoln Tahara

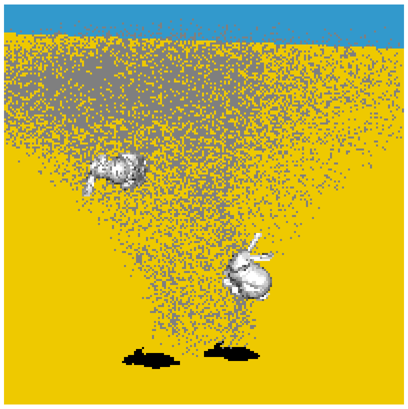


Figure 1 - *Bunnies caught in the tornado*

1 – Abstract

We present a method to quickly and efficiently simulate a tornado and its interaction with various objects. This simulation is designed to run in real-time. As a result, the interaction between the tornado and the objects seen in the simulation will not be an exact representation. Despite that, this system provides a good and convincing depiction of a tornado.

2 – Introduction

This system was created mainly for its novelty value, rather than a perfectly accurate and realistic representation of a tornado. That being said, a real-time and interactive representation became an option.

3 – Background

An actual tornado would not be terribly difficult to implement, but simulating it would require large amounts of resources which are not readily available. A full Navier-Stokes implementation [1] could easily be adapted to simulate a tornado. This would ensure that the simulation would remain accurate and it would reduce the amount of attention needed for the forces (only a few constant forces would be needed).

Of course, in order to be able to see the tornado, we would need particles. Most often, a tornado is only visible because of the dust and debris it picks up. This can be represented accurately through the use of particles; the more particles the better. The problem is the number of particles is limited by the bandwidth between the GPU and CPU, limiting us to less than ten thousand particles in our simulation. However, using the Million Particle System [2] we can get around this bandwidth by storing information in textures, allowing us to transport more data faster.

Since we needed a real-time simulation, a full Navier-Stokes implementation was out of the question, at least with our current hardware. We then settled for a particle system, instead, that would produce a tornado. This particle system consisted of a force field, a velocity field, and rules for spawning and killing particles.

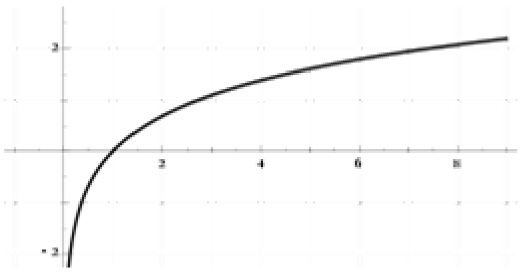
4.1 – Tornado

a) Force and Velocity

The force and velocity fields are both based on the natural log function:

$$y = \ln(x) + c$$

which yields the following graph:

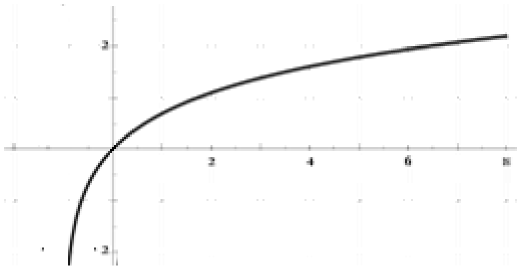


The force is strongest towards the center of the tornado, and non-existent within the eye of the tornado. As the particles move further away from the center, the forces acting upon them decrease in strength. In order to achieve our desired funnel shape, we exponentially decreased the rate at which the forces decay as we move further away from the eye.

The logarithm function finds us the fraction of the force (*y*) that will act on a given particle given the distance (*x*) from the center. But first, we need to perform a few transformations in order to get the desired output.

We translate the function to the left one unit so that we get an intersection at the origin:

$$y = \ln(x + 1) + c$$



Now, the difference between *y* = 2 and our logarithm function will give us the proper force decay at the top of the tornado:

$$y = 2 - \ln(x + 1) + c$$

c, the constant, translates the graph vertically, allowing us to decrease or increase the rate at which the force decays. A value of 2 will cause the force to instantly become 0, which we want at the base of the tornado. We want to linearly increase the value of *c* with respect to the tornado's height in order to complete our funnel shape. So we get:

$$c = \frac{particle_{height}}{tornado_{height}} * 2$$

Putting it all together gives us the fraction of the force (*y*) acting on a given particle:

$$y = 2 - \ln(x + 1) + \frac{particle_{height}}{tornado_{height}} * 2$$

x	= distance from the center
$height_{particle}$	= distance from ground
$height_{tornado}$	= height of the tornado

With the addition of a direction, our force field will be completed. The direction is the perpendicular clockwise direction from the vector starting at the center and ending at the particle location. Because tornados have a tendency to attract particles toward the center, we had to adjust the force so that it would point more towards the center, rather than being tangent to the tornado.

b) Particles

The particles are an integral component to this system. The particles in our system are broken up into three categories: the main vortex particles, noise particles, and object particles.

Vortex Particles

The vortex particles provide the visualization for the main structure of the tornado. The particles are spawned along the bottom of the tornado and are carried up and around by the force field. To prevent an excessive number of particles from being in the system, they are killed if their net force is zero. This set of particles alone created a funnel shape, but it looked unnatural. To remedy this situation, we added noise particles.

Noise Particles

Noise particles are spawned randomly in any location that is affected by the tornado's forces above a threshold. This way, we can achieve the chaotic look for our tornado. These particles are affected by gravity and disappear once they go outside the bounding box of the tornado.

Object Particles

The object particles represent a 3-Dimensional object that can be rotated and moved around by the tornado. A set of object particles consists of seven particles: top, bottom, left, right, front, back, and center. We bind these particles to their respective positions on a mesh so when the set of particles move and rotate, the mesh does the same.

c) Other Components

The tornado has various parameters which are read in from a text file. These are height, width, strength, gravity, speed, initial position, and the environment size.

Height/Width

Height and width determine the size of the tornado. Height, as the parameter suggests, affects the height of the tornado. The position at which the forces stop acting on particles depends on this Width refers to the tornado's width at its base. The width at the top is the square of the width at the base. The width affects the bounding box of the tornado, which in turn, affects what the tornado's forces have an impact on. In order to simplify the system, we have no forces, outside of gravity, act on positions outside the bounding box.

Strength

The strength parameter determines the maximum force the tornado can apply. This is the scalar that is multiplied by the fraction and direction mentioned earlier to obtain the force given a position. The code relies on the user to provide a good balance between height and strength to create a good tornado. If the strength is too low, relative to the height, the tornado will be more of a gust, having little impact on the environment. On the other hand, if the strength is too high, we will not be able to properly see the tornado; the particles will move too fast.

Gravity/Speed

Gravity refers to the only other force acting on the system besides the tornado. It's pretty straight forward. In addition to the forces, if any, imposed on a position, there is also a downward force. Speed, is also another simple parameter. Its only effect on the system is how fast the tornado moves. These values currently have no units and are just arbitrary scales.

Other

Initial position is where the tornado will start. The position can never be outside the environment. In the event that this does happen, the tornado will be snapped back into the environment.

Bounding box creates the boundaries for the environment. The bounding box prevents the tornado and any other object from escaping the environment. It can be as large as the user wishes, but never smaller than the tornado.

4.2 – Objects

Meshes

Meshes are placed into the simulation after being read in from an object file (.obj). After parsing the file, the data is stored into a series of containers: one for faces, one for edges, and one for vertices. Once all that is read in, the dimensions of the bounding box for the mesh are sent to the tornado particles function to create a set of object particles for the mesh.

Rotation/Translation

The rotation and translation of the object is based on the movement of the object particles. Yaw, pitch, and roll describe the angle of rotation about the x, y, and z axes, respectively, from the object's initial orientation.

The dot product of two vectors gives us the cosine of the angle between them. Therefore, the inverse cosine of the dot product of two vectors gives us the angle between them.

For example: vector A is the vector from the front to the back. Vector B is the axis which we are measuring the angle from, which in this case, is the Y-axis.

$$\cos(\theta) = \vec{A} \cdot \vec{B}$$

$$\theta = \cos^{-1}(\vec{A} \cdot \vec{B})$$

After calculating the angles, we sum up all the angles about a particular axis to find the net rotation. We label these yaw, pitch, and roll.

Display

The rotation/translation function figures out the exact position and orientation of the object and passes the information into the rendering function.

Translation is easily handled by adding the change in distance to the vertices of the mesh prior to drawing.

Rotation is handled using the matrix stack. Prior to drawing, the rotation is applied. After drawing, the matrix is reset. Doing it this way allows us to have multiple objects on screen with different orientations.

5 – Interactivity

What would be the point of rendering in real-time if the simulation was not interactive? We offer the user various ways to tweak the tornado to his/her liking.

Config File

The config file, "tornado.txt", offers various metrics (explained previously) which the user can adjust to change nearly all aspects of the simulation.

Mouse

The following describe the mouse controls available to the user:
Left click [hold] – change the camera's viewing direction. The camera moves with the tornado.
Right click [hold] – move the mouse up to zoom out and down to zoom in.

Keyboard

The following describe the keyboard actions available to the user:
W – move the tornado away from the camera.
S – move the tornado towards the camera.
A – move the tornado to the left.
D – move the tornado to the right.
G – spawn a gourd.
H – spawn a humanoid block
B – spawn a bunny.

6 – Results

We have about three hundred particles in the system at any given time and it runs without lag. It takes about fifteen simultaneous objects before a 4.0GHz processor shows any signs of slowing the system down.

The particle system is a convincing representation of a tornado. Though still in the early stages, this system works well. The pictures below show our progress over the course of the project.

The physics applied to the objects are also not quite right. There is a lack of acceleration. This is the only real bug for now.

7 – Future Work

There is still a lot to do before we can call this a complete tornado simulation. We can further refine the particles around the tornado in order to make it look more realistic. This is more important for the top of the tornado and the point at which the tornado makes contact with the surface. Including more noise particles in those areas would help.

Tornados are usually accompanied by a cloud above them. Adding a cloud mesh right above the tornado would probably be our next step forward.

An actual tornado would create winds extending much further than its immediate vicinity. We can achieve this effect in our simulation by adding more residual forces surrounding the tornado. In order to depict this, we would need more particles. This is where the Million Particle System [2] method would come in.

In addition to improving the tornado, we can also expand the environment which it resides in. Instead of having it be a box, we can have hills, valleys, and such. It would also be nice to add other environmental elements such as fire and water to create fire whirls, water spouts, and other interesting forms of tornadoes.

Finally, we can improve the meshes and their behaviors in interacting with the tornado. Adding a light source and creating shadows through shadow volumes would be the first thing to do. Then we would add textures. Lastly, we would probably implement breakable objects.

8 – Conclusion

We have presented our interactive tornado simulation which runs in real-time. The user has plenty of control over the tornado and its basic properties. Through particle effects and kinematics, we have successfully created a simple tornado simulation.

Our tornado simulation is far from complete, but we have provided a good starting point. We have a basic implementation of a tornado and its interaction with objects.

Acknowledgements

Thanks:
Barbara Cutler for base codes and files.
Nick and Lincoln for being awesome.
Tornadoes for existing.

References

[1] Nick Foster and Dimitri Metaxas. Realistic Animation of Liquids. *Graphical Models and Image Processing,* 58(5):471-483, 1996.

[2] Lutz Latta. Building a Million Particle System. In Game Developers Conference, 2004.

[3] Karl Sims. Particle Animation and Rendering Using Data Parallel Computation. *ACM Computer Graphics (SIGGRAPH '90)*, 24(4):405-413, 1990.

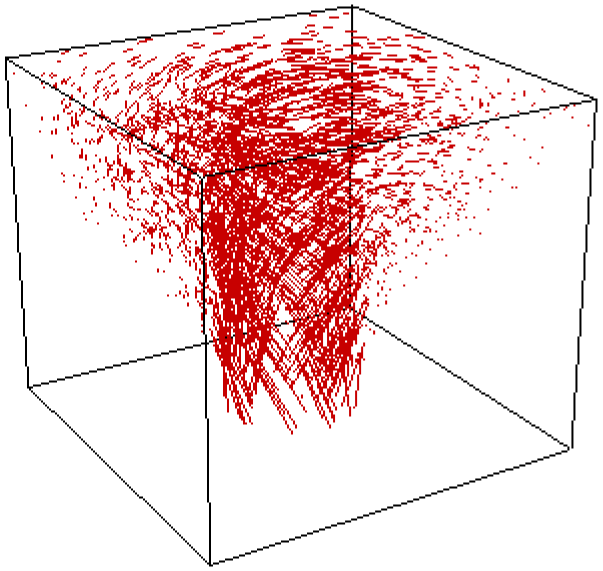


Figure 2 – *Tornado forces*

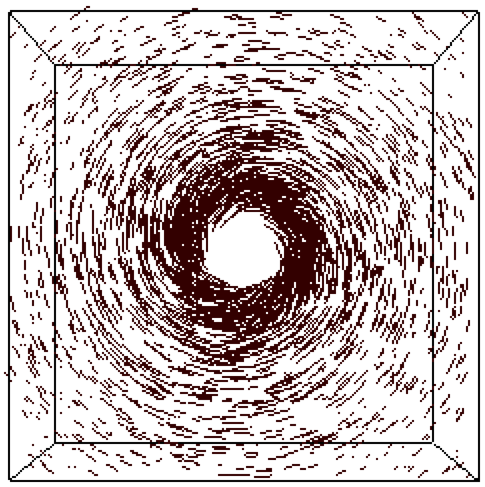


Figure 3 – *Tornado forces (top view)*

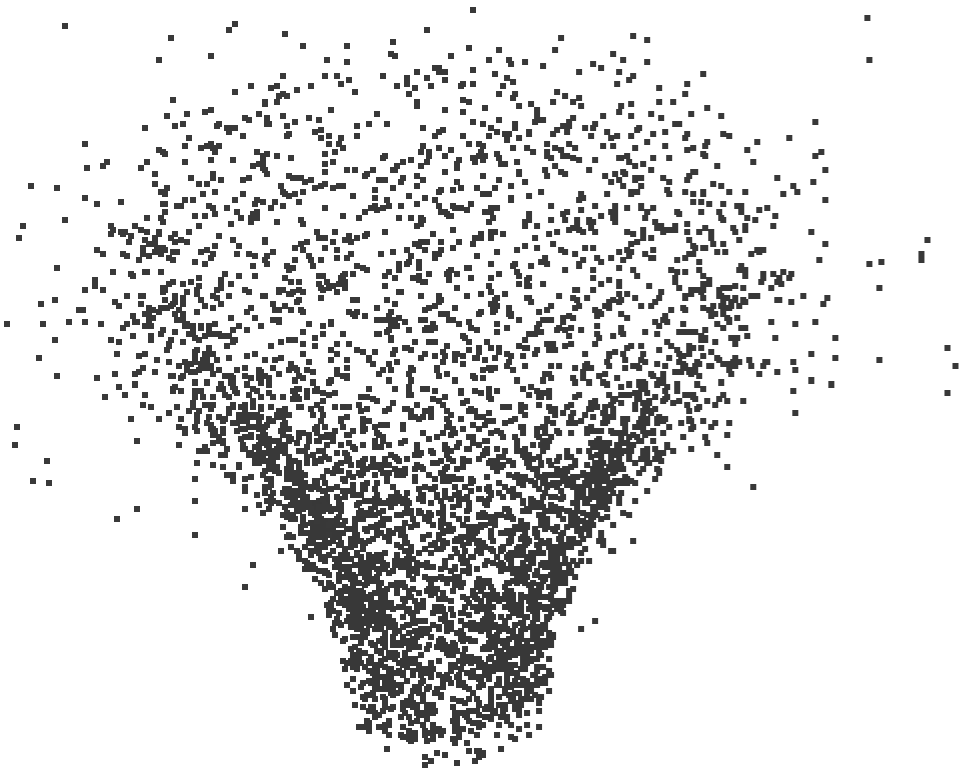


Figure 4 – *Particles after being spawned and moved by the forces*

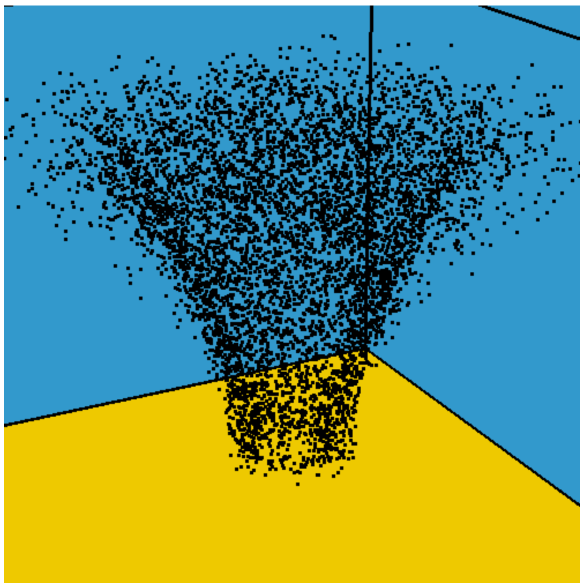


Figure 5 – *Tornado in the environment*

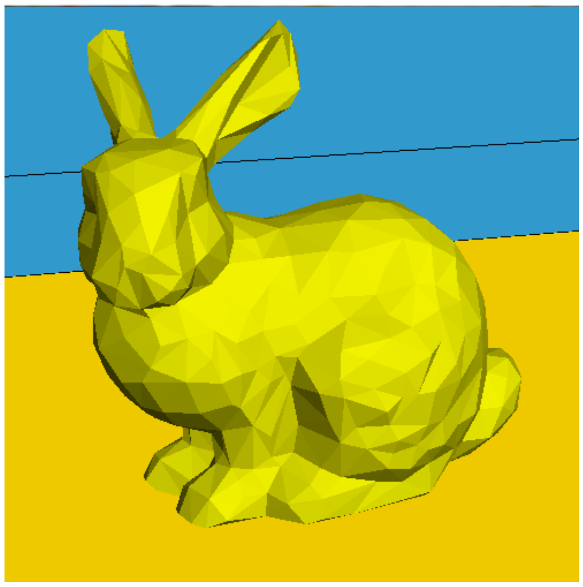


Figure 6 – *Bunny mesh (colored yellow)*

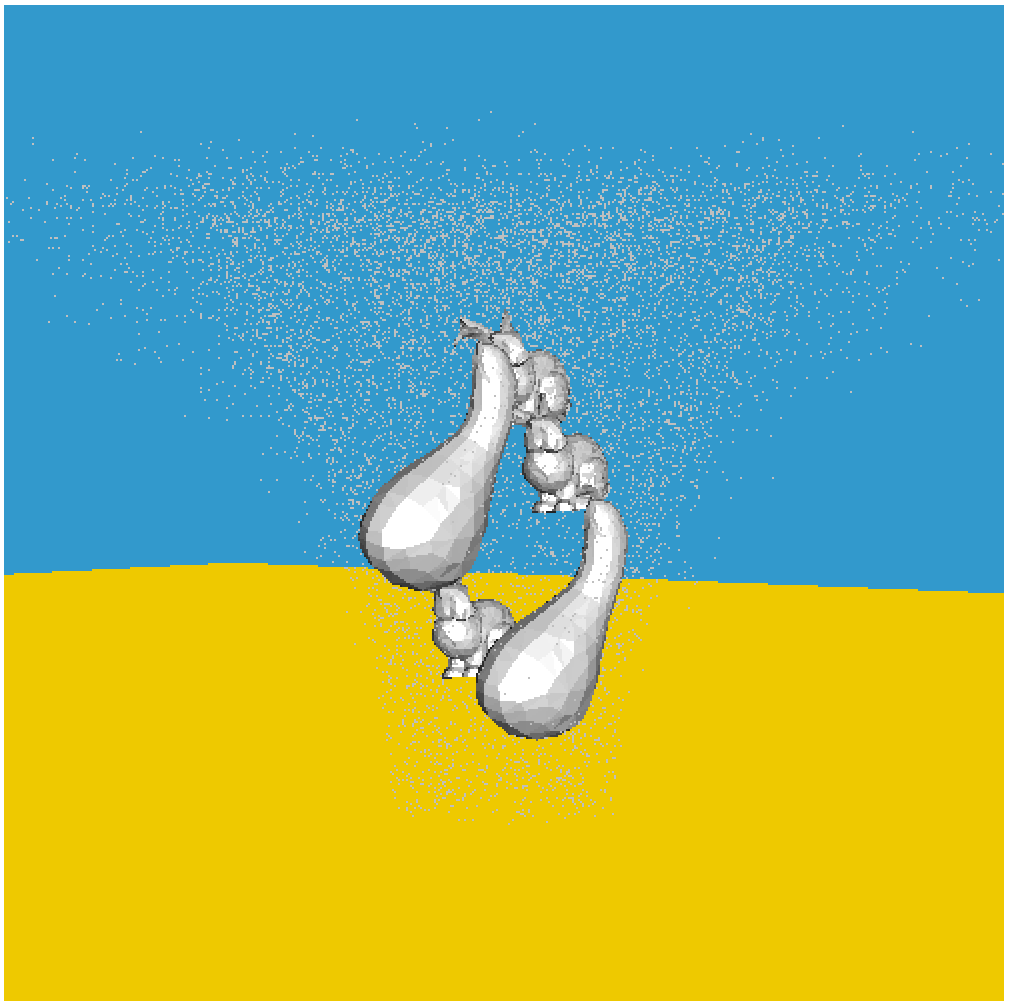


Figure 7 – *Two gourds and three bunnies caught in the tornado (rotation not implemented)*

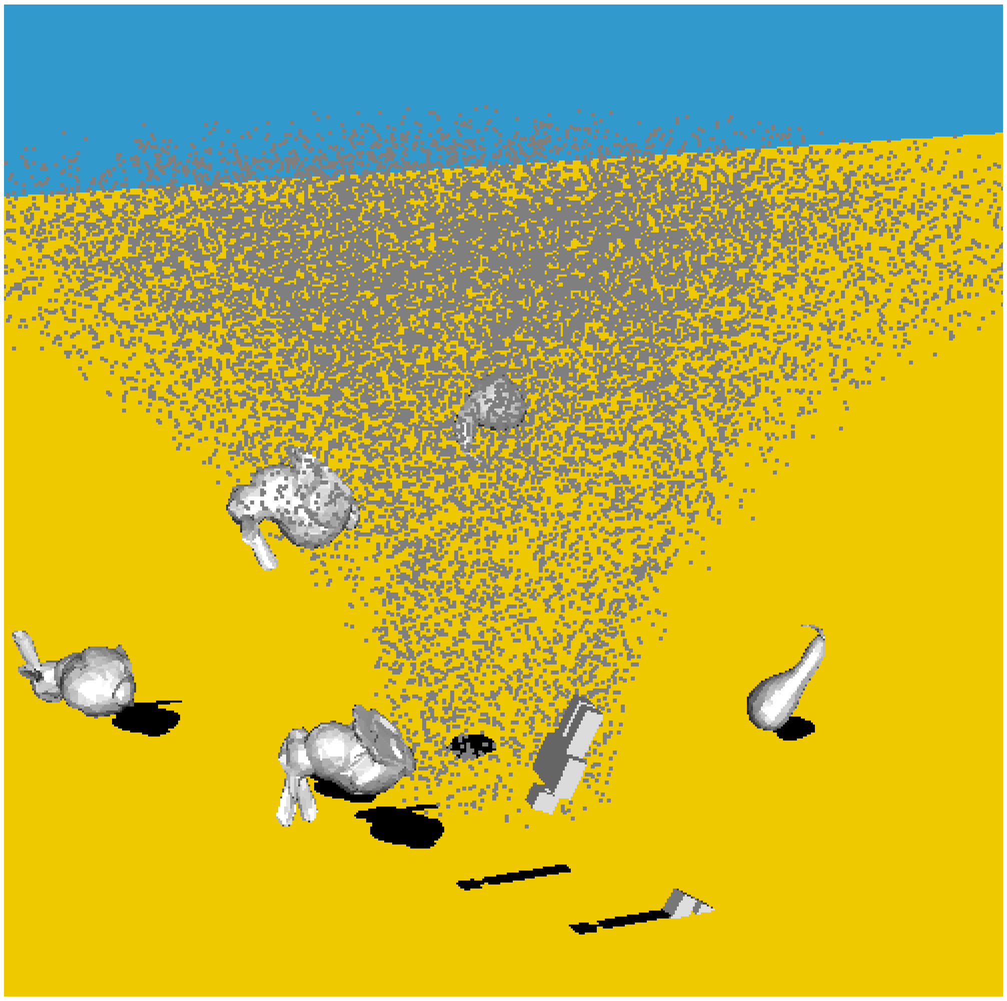


Figure 8 – *Various objects interacting with the tornado (with rotation and planar projection shadows)*