Real Time Dynamic Wind Calculation for a Pressure Driven Wind System

Criss Martin*
Dept. of Computer Science & Engineering
University of North Texas

Ian Parberry†
Dept. of Computer Science & Engineering
University of North Texas

Abstract

We describe real time dynamic wind calculation for a pressure driven wind system. This simple and elegant approach allows us to perform visual effects in real time using a minute fraction of the CPU's processing power over and above what is required for static wind. Experiments were performed with a real-time application to render wind-driven snow over a 3D terrain to verify these claims.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation;

Keywords: Dynamic wind, snow, real time animation

1 Introduction

There are many ways that developers have implemented wind, or wind like effects, in 3D real time environments. The easiest solution is to set a global static wind vector that remains constant regardless of obstructions from the terrain and objects. Static wind is most often used as a difficulty modifier (for example, kicking a football on a windy day taking into account wind direction), but it adds little visually to the scene. Wang et al. [2005] attempt to render realistic snowfall with visual effects created by dynamically manipulating textures. However, since the texture is a two-dimensional image, there is no way to have wind-driven visual effects such as snowflakes blowing over and around features of the terrain.

Other applications go to the other side of the spectrum by using complicated equations to calculate all of the eddies that form from wind flowing around objects in the world (see, for example, Wei et al. [2003], Wei et al. [2004], Stam and Fiume [1993], Stam and Fiume [1995], Stam [1999]). While the end product is visually stunning, its calculations are extremely demanding of CPU power. Real time applications including games simply cannot afford the luxury of solving complicated mathematical equations. A high frame rate and high user responsiveness must be maintained at all times, which means that only a fraction of the CPU power can be used for wind calculations.

We propose a method for controllable real time dynamic wind calculation for a pressure driven wind system. Our method allows realistic wind-driven visual effects, and has the following properties:

- Fast set-up time.
- Negligible effect on rendering speed.
- Allows the designer to specify the positions and pressure values of a high and a low pressure system to drive the wind.

Setup is performed when new terrain is loaded and when the pressure values are changed or moved. It consists of the calculating the following values at each terrain vertex:

1. The pressure gradient.
2. The wind shadows.
3. The wind pressure.
4. The wind uplift.

The wind vector for each particle is then interpolated from the terrain vertex values on each frame of animation.

The main part of this paper is divided into four sections. Section 2 describes the calculation of the pressure gradient. Section 3 describes the calculation of the wind shadows. Section 4 describes the calculation of the wind pressure and uplift vectors. Section 5 describes the application of our dynamic wind algorithm to a wind-driven snow demo, and justfies the claims made for its performance.

2 Pressure Gradient

Wind always blows from a centers of high pressure to centers of low pressure, with a speed determined by the pressure difference and the rate of air exchange between the pressure centers. Change of pressure over unit distance is called *pressure gradient*, and the greater this value the faster the wind will blow (see, for example, [Pidwirny 1999]).

We will set a pressure gradient in our 3D world by choosing two points on our terrain, one a high pressure center $p_h = (x_h, z_h)$, the other a low pressure center $p_\ell = (x_\ell, z_\ell)$. We will compute and store the pressure at each vertex of our terrain. We start by interpolating the pressure values on a line between p_h and p_ℓ. The pressure at all the other points p that do not fall on the line between p_ℓ and p_h will be calculated by finding the point of intersection between the pressure gradient line and a line perpendicular that runs through p. Figure 1 shows a pressure gradient with p_h at the upper left corner and p_ℓ at the lower right corner of the image, respectively, and a grayscale background indicating the interpolated pressure value.

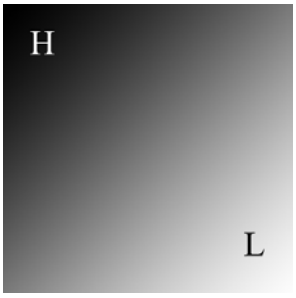

Figure 1: Sample pressure gradient.

We will assume that $p_h \neq p_\ell$. If the slope of the line between p_h and p_ℓ (called the *pressure gradient line*) is zero, all vertices on

*Email: nitramssirc@yahoo.com
†Email: ian@unt.edu

a column will have the same pressure value, so we only need to calculate the pressure for a single row. Similarly, if the slope is infinite, all the vertices on a row will have the same pressure value. In either case, we use the following formula to compute the pressure value at a given point:

$$f_a = f_\ell + p(f_h - f_\ell)/d$$

where f_a is the calculated air pressure, f_ℓ is the pressure at p_ℓ, f_h is the pressure at p_h, p is the position on the row or column depending on the slope, and d is the length of the row or column.

Now suppose that $p_h \neq p_\ell$ and the slope of the pressure gradient line is strictly between zero and infinity. To find the pressure value for point p in this case, we will need to find the pressure at the point of intersection between the pressure gradient line and the line perpendicular to it that runs through p.

Let m_1 be the slope of the pressure gradient line, m_2 be the slope of the line perpendicular to it, b_1 be the z-intercept of the pressure gradient line, and d be the distance between the two pressure centers.

$$\begin{aligned} m_1 &= (z_\ell - z_h)/(x_\ell - x_h) \\ m_2 &= -(1/m_1) \\ b_1 &= z_\ell - (m_1 * x_\ell) \\ d &= \sqrt{(x_\ell - x_h)^2 + (z_\ell - z_h)^2} \end{aligned}$$

We perform the following calculations for each vertex $p = (x_p, z_p)$ in the terrain. First, compute b_2, the z-intercept of the perpendicular line.

$$b_2 = z_p - (m_2 * x_p)$$

Then we compute the intersection point $p_i = (x_i, z_i)$ between the pressure gradient line and the line perpendicular to it that runs through p.

$$\begin{aligned} x_i &= (b_2 - b_1)/(m_1 - m_2) \\ z_i &= (m_1 * x_i) + b_1 \end{aligned}$$

Let d_ℓ be the distance from p_i to p_ℓ, and d_h be the distance from p_i to p_h. We then interpolate the pressure to point p.

Let d_1 be the Euclidean distance between p_h and p_ℓ. If $d_h > d_1$,

$$f_a = f_\ell - (d_\ell/d_1) * (f_h - f_\ell),$$

otherwise,

$$f_a = f_\ell + (d_\ell/d_1) * (f_h - f_\ell)$$

We do this last check at the end to see if the point of intersection lies between the two pressure centers. If it does not, we interpolate the pressure value as if the line continues. That is, if the point is behind p_h, the pressure will be higher, and if it is behind p_ℓ, the pressure will be lower.

3 Wind Shadows

Wind shadow zones are areas on the map that are blocked from the wind by the terrain. In wind shadow zones the speed of the wind at (x, y, z) depends on y/y_s, where y_s is the height of the wind shadow at (x, z). These areas will have a lower pressure and slower wind speed than they would if they were exposed to the wind because air is not moving in these zones as it is elsewhere.

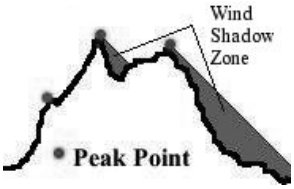

Figure 2: Wind shadow zones.

To determine the wind shadow zones we examine each terrain vertex to determine *peak points*, which are points on the terrain that are higher or equal to a point behind it, and higher than a point after it in the plane of the wind direction. (See Figure 2.) If a point is a peak point, we will calculate the wind shadow it creates in the plane of the wind direction. The wind shadow will include all points that fall in the triangle between the peak point, the point directly under the peak point at height 0 and the point at height 0 that is at some distance d_s away. To calculate d_s we will use projectile motion to determine how long it takes the wind to fall all the way to height 0 from the peak point and use that to determine how far the wind travels before it hits height zero.

$$d_s = \sqrt{h(b_x^2 + b_z^2)/4.9}$$

where h is the height of the peak and $v_b = (b_x, b_z)$ is the base wind velocity, which is described more fully in the next section (specifically, it will be the component of v_p due solely to a_p, without the contribution u_p due to the terrain).

Once we know the location of the shadow zone, we can compute which vertices in the shadow zone using a simple array lookup.

4 Calculating the Wind Pressure Vector

The wind pressure is the pressure the wind exerts on objects due to the wind's velocity [Vent-Axia 2005]. We will compute the wind pressure as a vector at each terrain vertex. Before we calculate anything, we must first determine whether the given vertex falls in a wind shadow zone. If the vertex is in a wind shadow zone, then we linearly scale the pressure at that point as described in the previous section.

There are three steps to determining the wind pressure vector at terrain vertex p, which we will denote s_p:

1. Find the total wind acceleration vector at terrain vertex p , which we will denote t_p, due to the pressure gradient and uplift from the terrain.
2. Convert t_p into a velocity vector v_p.
3. Convert v_p into the wind pressure vector s_p.

Step 2, the conversation from acceleration to velocity, is simple mechanics, which we will omit. To convert wind velocity to wind

pressure in Step 3, we use the following formula from [Vent-Axia 2005]:

$$s_p = 0.6v_p$$

where 0.6 is a constant derived from the density of the air at 20°C and average relative humidity.

It remains to describe Step 1, the calculation of wind acceleration. Pidwirdny [1999] states that the wind acceleration $A(p_1, p_2)$ at point p_1 due to point p_2 is given by

$$A(p_1, p_2) = 0.775(P_1 - P_2)/d,$$

where the constant term 0.775 is the reciprocal of the density of air (for which we will use a constant value of 1.29), P_1 is the pressure at p_1, P_2 is the pressure at p_2 and d is the Euclidean distance between p_1 and p_2.

We discretize the forumula from Pidwirny as follows. The wind acceleration vector at terrain vertex p , $a_p = (x_a, z_a)$, will be computed componentwise as the average of the accelerations due to neighboring terrain vertices. For example, suppose terrain vertices are separated by distance d, and let q and r be the two neighbors of p in the x dimension. Then x_a is computed as

$$x_a = 0.775(P_r - P_q)/2d,$$

where P_r is the pressure at r and P_q is the pressure at q. The z component z_a is calculated similarly.

Once we have the acceleration caused by the pressure gradient, we add the acceleration due to the uplift from the terrain. The uplift will allow the wind to roll over the terrain. To determine the uplift at point p, we must calculate the resultant vector from the collision between the wind and the terrain. Let n_p be the bilinearly interpolated surface normal of the terrain at point p, and v_w be a normalized vector from the low pressure to the high pressure center. Then u_p, the uplift acceleration vector at point p, is given by:

$$u_p = v_w - 2(v_w \circ n_p)n_p$$

The total wind acceleration vector at point p is then given by $t_p = a_p + u_p$. Finally, if point $p = (x_p.y_p, z_p)$ falls in a wind shadow zone, we scale t_p by a factor of y_p/cy_s where y_s is the height of the wind shadow zone at point p, and c is a tunable constant ($c = 2$ was arbitrarily chosen for our demos). This causes the wind to blow slower by a linear amount the lower you get in the wind shadow zone.

5 Applications

There are many visual effects that could benefit visually from using a dynamic wind model, including cloth [Cordero 2005], flames [Lamorlette and Foster 2002], grass [Ramraj 2005; Wang et al. 2005; Perbet and Cani 2001], and object aerodynamics [Rhodes 2005]. The pressure system could also be used to simulate pressurized spacecraft and render the effects of rapid decompression. The only real adjustment that would need to be made would be the calculation of the pressure gradient in three dimensions, rather than the two-dimensional approach used in this paper.

Figure 3: Screen shot showing simple terrain.

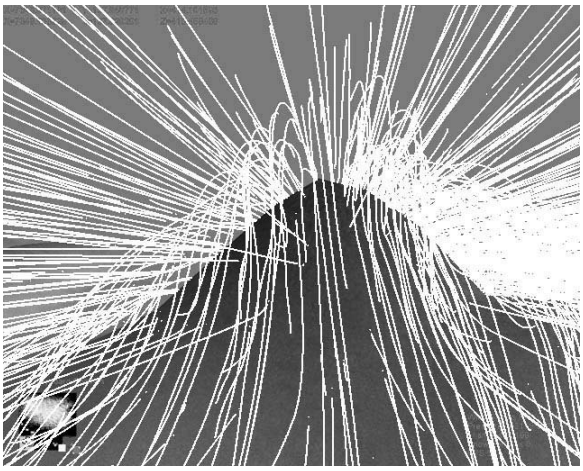

Figure 4: Screen shot showing paths of snow particles for terrain in Fig. 3. Notice how the snow particles are blown up and over the hill.

We have implemented our dynamic wind system in an application that shows wind-driven snow particles. Figure 3 shows some simple terrain. Figure 4 shows the paths of wind-driven snow particles over that terrain. Observe how the particles are driven by the wind up and over the ridge. Figure 5 shows some more complicated terrain. Figure 6 shows the paths of wind-driven snow particles over that terrain. Observe the clustering of particles in the wind shadow behind the peak.

We ran experiments that varied the number of snow particles from 250 to 4000 and measured the average frame rate for rendering falling snow with dynamic versus static wind. The experiments were performed on a Windows PC with a single-core 3.2 GHZ Pentium 4 processor, and an Nvidia 6800 Ultra graphics card. The frame rate overhead from dynamic wind was found to be negligible, typically less than or equal to one frame per second delay. Setup times for a 64K terrain (which include calculation of pressure and wind speed at all terrain vertices, and computation of wind shadows) were typically 1 or 2 lost frames while simultaneously rendering at 20 fps.

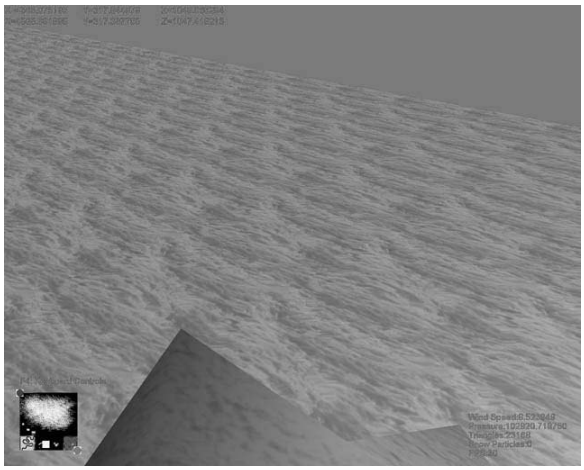

Figure 5: Screen shot showing more complicated terrain with a mountain overlooking the ocean.

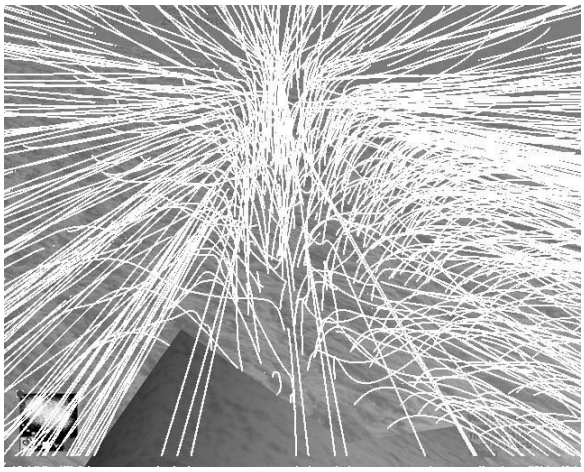

Figure 6: Screen shot showing paths of snow particles for terrain in Fig. 5. Notice the effect of the wind shadow at lower right.

6 Conclusion

We have described real time dynamic wind calculation for a pressure driven wind system. Calculations are performed for all vertices of the terrain grid, and then interpolated as needed. We compute for all vertices a pressure gradient, from which we derive a wind pressure vector with components from the base wind vector and terrain uplift. Wind shadows are then computed. Our simple and elegant approach has been implemented to drive a real time simulation of snow blowing over a terrain.

Remaining open problems include efficient real-time modeling of snow accumulation, building on the work of Fearing [2000], and application of our technique replacing static wind in related animation applications such as cloth [Cordero 2005], flames [Lamorlette and Foster 2002], grass [Ramraj 2005; Wang et al. 2005; Perbet and Cani 2001], and object aerodynamics [Rhodes 2005].

References

CORDERO, J. M. 2005. Realistic cloth animation using the mass-spring model. In *Game Programming Gems 5*, K. Pallister, Ed. Charles River Media, 421–433.

FEARING, P. 2000. Computer modelling of fallen snow. In *SIGGRAPH '00: Proc. 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 37–46.

LAMORLETTE, A., AND FOSTER, N. 2002. Structural modeling of flames for a production environment. In *SIGGRAPH '02: Proc. 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 729–735.

PERBET, F., AND CANI, M.-P. 2001. Animating prairies in real-time. In *SI3D '01: Proc. 2001 Symposium on Interactive 3D Graphics*, ACM Press, New York, NY, USA, 103–110.

PIDWIRNY, M. 1999. Introduction to the atmosphere. In *Fundamentals of Physical Geography*, 2nd ed. http://www.physicalgeography.net/fundamentals/7n.html, ch. 7.

RAMRAJ, R. 2005. Dynamic grass simulation and other natural effects. In *Game Programming Gems 5*, K. Pallister, Ed. Charles River Media, 411–419.

RHODES, G. 2005. Back of the envelope aerodynamics for game physics. In *Game Programming Gems 5*, K. Pallister, Ed. Charles River Media, 395–409.

STAM, J., AND FIUME, E. 1993. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH '93: Proc. 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 369–376.

STAM, J., AND FIUME, E. 1995. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH '95: Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, NY, USA, 129–136.

STAM, J. 1999. Stable fluids. In *SIGGRAPH '99: Proc. 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128.

VENT-AXIA. 2005. Wind pressure and flow around buildings. http://www.vent-axia.com/sharing/windflow.asp.

WANG, N., AND WADE, B. 2005. Let it snow, let it snow, let it snow (and rain). In *Game Programming Gems 5*, K. Pallister, Ed. Charles River Media, 507–513.

WANG, C., WANG, Z., ZHOU, Q., SONG, C., GUAN, Y., AND PENG, Q. 2005. Dynamic modeling and rendering of grass wagging in wind: Natural phenomena and special effects. *Comput. Animat. Virtual Worlds 16*, 3-4, 377–389.

WEI, X., ZHAO, Y., FAN, Z., LI, W., YOAKUM-STOVER, S., AND KAUFMAN, A. 2003. Blowing in the wind. In *SCA '03: Proc. 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 75–85.

WEI, X., ZHAO, Y., FAN, Z., LI, W., YOAKUM-STOVER, S., AND KAUFMAN, A. 2004. Lattice-based flow field modeling. *IEEE Transactions on Visualization and Computer Graphics 10*, 6, 719–729.