Mark Blunk, Sam Coskey, and Erik Curre
University of Washington

Fountain Spray as a Particle Model

Team Number 357

February 11, 2002

Abstract

We discuss a simple particle model for a decorative fountain. We give a method by which to control the water spread of a fountain using statistics generated by the model. Symbolic and computational solutions are explored. We apply the methods of our model to a typical fountain example. Finally, we summarize the relative benefits and drawbacks of our solution.

Contents

1 Introduction **2**

2 Prerequisites **2**
2.1 Definition of the fountain . . . 2
2.2 Definition of the anemometer . . . 3
2.3 Simplifications to physics . . . 3
2.4 Additional assumptions . . . 3
2.5 Restatement of the problem . . . 3

3 Physical Model **3**
3.1 Forces considered . . . 3
3.2 Symbolic calculations . . . 4
3.3 Numerical Method . . . 5

4 Experimental Extension of the Model **6**
4.1 The fountain simulation algorithm . . . 6
4.2 The data collection scheme . . . 7
4.3 The testing procedure . . . 9
4.4 Summary . . . 10

5 Sample Applications **10**
5.1 A typical fountain . . . 10
5.2 Another fountain . . . 11
5.3 Other fountains . . . 13

6 Analysis of the Model **13**
6.1 Strengths of the model . . . 13
6.2 Weaknesses of the model . . . 13

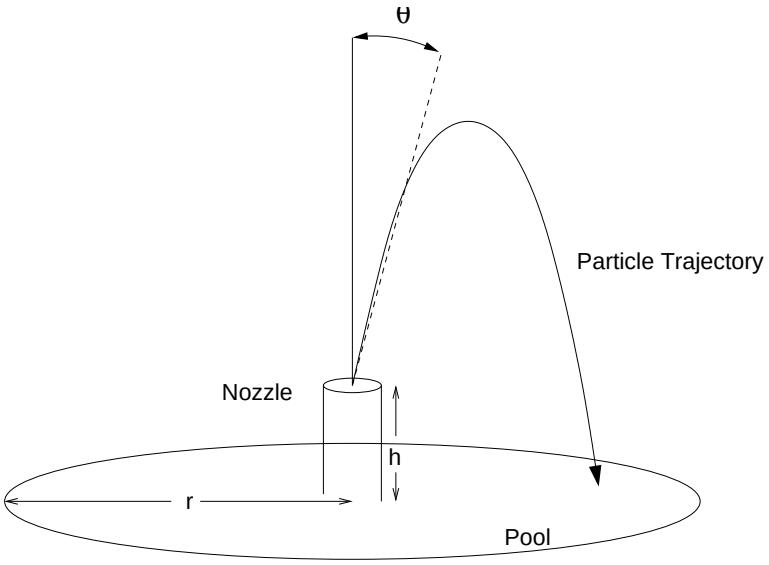

Figure 1: A typical fountain setup. In this picture, there is no ambient wind.

1 Introduction

From our very own University of <expletive deleted>, to the grand Bellagio Hotel and Casino in Las Vegas, institutions worldwide have discovered the attraction of a frivolous display of water. Typically, a fountain will have several nozzles that shoot water upwards or at an angle into the air, and a pool to collect the water as it falls. Unfortunately, one cannot control the weather, and wind can cause the water to soak nearby walkways and pedestrians.

Using only the data from a nearby anemometer, we wish to control the flow of water under high wind conditions. If we reduce the height of the fountain water as winds pick up, we can hopefully retain most of the water in the pool, preventing the fountain from running dry and keeping pedestrians safe.

We are thus called upon to model the effects of wind on a stream of water. To do this, we will adopt a particle model of water flow. That is, we divide the water into a number of spherical droplets which can be modeled individually.

2 Prerequisites

In order to present our solution, we have made numerous simplifications to the given problem. We have made several assumptions regarding either the problem domain itself, or the mechanics and physics that affect the fountain. At the end of this section, we restate the problem in terms of the assumptions that we have made.

2.1 Definition of the fountain

For the sake of simplicity, we will generally assume in our discussions that a fountain consists of a single nozzle centered in a circular pool of water, having radius r. The nozzle is at a height h above the pool. The water is expelled from the nozzle no more than some θ radians from the vertical. Such a setup is exemplified in Figure (1).

The fountain controls are limited to just one parameter. This parameter will essentially be a dial that somehow determines the intensity at which water flows from the nozzle.

Symbol	Description	Value
m	mass of water particle	1.413×10^{-5}kg
g	strength of gravitational field	9.81N kg^{-1}
μ	dynamic viscosity of air	1.798×10^{-7}kg m s^{-1}

Figure 2: Symbols used

2.2 Definition of the anemometer

In this problem, we are given as input only data from a nearby device which measures wind speed and heading, called an anemometer. In reality, the measurements from this device may not be accurate enough to allow us to approximate the wind force near the fountain. For this reason, we must make several assumptions.

The anemometer is assumed to be near enough to the fountain that its data is representative of the wind speed at and near the fountain. We also assume that the wind provides a uniform force on all of the particles at any given time. Additionally, we suppose that the data recorded by the anemometer may be used by our algorithm without significant delay.

2.3 Simplifications to physics

To greatly simplify our treatment of the water flow, and to avoid issues such as fluid dynamics and surface tension entirely, we have adopted a particle model for the fountain water. More specifically, we have assumed that the water expelled from the fountain is merely a collection of spheres, with radius 1.5mm [3]. The particles are assumed not to interact with each other, and do not deform during their flight through the air.

Since we assume spherical water drops, the mass of each particle may be calculated by $m = \text{density} \times \frac{4}{3}\pi r^3$. The value of m is shown in Figure (2).

We may now narrow our definition of the fountain's control dial. We suppose that this dial sets the initial speed at which water particles are ejected from the nozzle. This parameter is called v_0.

2.4 Additional assumptions

We have assumed that the viscosity of the atmosphere is constant everywhere. In particular, we have assumed that the fountain rests at sea level. This will allow us to choose the dynamic viscosity of the air which is taken from [4], and shown in Figure (2), below.

2.5 Restatement of the problem

All of these assumptions and simplifications necessitate a restatement of the problem:

> Suppose that the nozzle angle θ, and the radius of the pool r are fixed. Given the current wind velocity at the fountain, find the maximum speed v_0 such that if the water particles are ejected from the nozzle at v_0, the water jet will fall (except perhaps for a suitable error) within the pool.

3 Physical Model

In this section, we describe the components of our fountain model in terms of elementary kinematics.

3.1 Forces considered

The most important element of our model is the effect of various forces on a single water particle. The forces which we consider are gravity (mg), air drag (D), and wind (W). This is reflected by the free body diagram in Figure (3). We now proceed to describe the equations we will use to model these forces.

The air drag is a force which acts against the direction of motion of a particle. The magnitude is dependent on the velocity of the particle, and computed by

$$D(t) = 6\pi\mu r_p v(t) \tag{1}$$

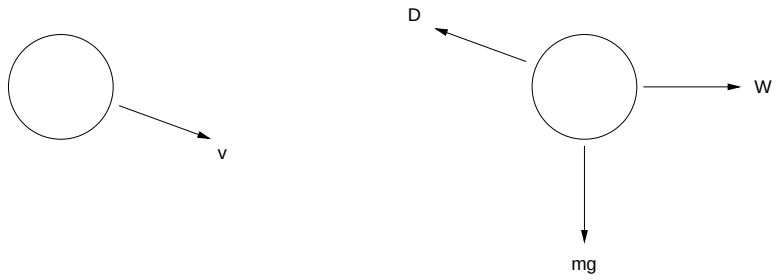

Figure 3: (Left) Schematic of a typical water particle, with velocity shown. (Right) A free body diagram of the water particle.

Here, r_p denotes the radius of the particle, and $v(t)$ denotes the magnitude of its velocity at time t. This expression for the drag force was taken from [1].

The wind is a constant force, which is assumed to be horizontal. It is similar to the drag force equation above, except that it does not depend on the velocity of the particle.

$$W = 6\pi\mu r_p v_w \tag{2}$$

Here, v_w represents the velocity of the wind. This natural expression for the wind force was taken from [2].

3.2 Symbolic calculations

Ultimately, we want to find an expression for the initial speed of the water out of the fountain in terms of the ambient wind velocity. First, we will derive an equation for the vertical position (5) of the particle by applying Newton's 2nd law.

$$ma_{\text{vert}}(t) = \Sigma F = -mg - 6\pi\mu r_p v_{\text{vert}}(t) \tag{3}$$

Solving (3) for $v_{\text{vert}}(t)$, with the initial condition $v_{\text{vert}}(0) = v_0 \sin\theta$ (the vertical component of the initial velocity of the particle) yields

$$v_{\text{vert}}(t) = \frac{\exp(\frac{-6\pi\mu r_p t}{m})(6\pi\mu r_p v_0 \sin\theta + mg) - mg}{6\pi\mu r_p} \tag{4}$$

We may now compute an expression for the vertical component particle position (5), from (4). We take as the initial condition $s_{\text{vert}}(0) = h$, where h is the height of the fountain nozzle. Our result is the following expression.

$$s_{\text{vert}}(t) = \frac{-\exp(\frac{-6\pi\mu r_p t}{m})[6\pi\mu r_p m v_0 \sin\theta + m^2 g] - 6\pi\mu r_p mgt + m^2 g + 6\pi\mu r_p m v_0 \sin\theta + (6\pi\mu r_p)^2 h}{(6\pi\mu r_p)^2} \tag{5}$$

Now, when (5) is zero, our particle has struck the ground. In order to compute the horizontal distance from the nozzle that the particle has traveled, we must compute the root of Equation (5). This root shall be denoted t_0.

Next, we derive the equation expressing the horizontal position of the particle as a function of time. Again applying Newton's 2nd law, we have that

$$ma_{\text{horiz}}(t) = \Sigma F = 6\pi\mu p_r v_w - 6\pi\mu r_p v_{\text{horiz}}(t) \tag{6}$$

Solving (6) with the initial condition $v_{\text{horiz}}(0) = v_0 \cos\theta$ yields the following expression for the horizontal velocity of the particle.

$$v_{\text{horiz}}(t) = -(v_w - v_0 \cos\theta) \exp\frac{-6\pi\mu r_p t}{m} + v_w \tag{7}$$

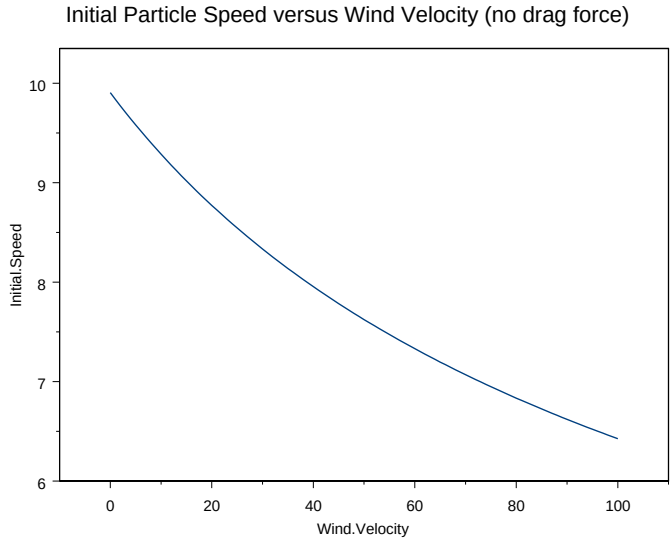

Figure 4: A graph of desired initial particle speed versus current wind velocity. Here, the drag force is not taken into account. The parameters are $r_p = 5\text{m}$, and $\theta = 15°$.

We now solve (7) to drive an expression of the horizontal position of the function.

$$s_{\text{horiz}}(t) = \frac{m(v_w - v_0 \cos\theta)\exp(\frac{-6\pi\mu r_p t}{m}) + 6\pi\mu r_p v_w t}{6\pi\mu r_p} \tag{8}$$

Here, θ represents the most extreme angle with which a particle might leave the fountain. Now that we have our time taken for the particle to strike the ground, t_0, and our expression for the horizontal position of the particle, (8), we now compute $s_{horiz}(t_0)$, deriving the maximum distance that a particle with initial speed v_0 (launched at angle θ) will travel.

Now, if we wish to restrict this distance to be no bigger than the radius r of the pool, we set $s_{horiz}(t_0) = r$, and then solve for v_0. This leaves us with a relationship between the speed of the wind, v_w, and the initial velocity v_0. When drag force is not taken into account it is possible to find the root for (5) for abstract v_0. Hence we are able to find the express this relationship between v_0 and v_w symbolically by the following equation:

$$v_0 = \sqrt{\frac{mg^2 r}{12\pi\mu r_p v_w \sin^2\theta + mg\sin 2\theta}} \tag{9}$$

With the exception of drag force considerations, this equation is a final solution. We can graph this equation with some sample values, and such a plot is shown in Figure (4).

3.3 Numerical Method

Unfortunately, when drag is considered, the equations become too difficult to solve explicitly for v_0. We must turn to a numerical method to solve this problem, outlined in Algorithm (1). Instead of finding the root of (5) for an abstract value v_0, We choose a value for the wind speed v_w. We then select increasing values of v_0, and compute the horizontal position of the particle when it hits the ground, stopping once our particles are landing outside the pool.

Thus, it is possible to numerically obtain a correspondence between wind velocity, v_w, and the initial speed v_0 required to keep the particle inside the pool. This relationship determines, for a given wind velocity, what is the maximum initial speed that the particles may have so that they stay inside the pool.

Algorithm 1 The numerical method algorithm

```
for v_w from 0 100, by increments of 1 do
  for v_0 from 0 20, by increments of .1 do
    t_0 = root(s_vert(t))
    d = s_horiz(t_0)
    if (d < r_pool) then
      desired initial speed := v_0
    end if
  end for
  return v_w and the desired initial speed
end for
```

4 Experimental Extension of the Model

We now propose a way to extend our physical model using a computer simulation of a fountain.

A natural way to model a fountain of the type we have described is with a computer simulation called a particle system. The term particle system was first used by Reeves [5] to describe a technique he used to create a special effects sequence for the movie Star Trek II: The Wrath of Khan. The effect in question was that of a bomb exploding on the surface of a planet and expanding from the point of impact until the explosion eventually engulfed the entire planet. In this case, the expanding ring of fire was represented by thousands of individual points, or particles, which moved according to some governing rule.

A common characteristic of particle systems is that there is usually some degree of randomness in the way that the initial attributes of a particle are specified. For example, particles may be given a random initial velocity or may be emitted in a random direction from some initial source, or both. Uses of such systems include the simulation of explosions, smoke, fluid flow, flocks of birds, fire, clouds, and fountains.

To test our physical model, we developed a fountain simulator based on a particle system. The simulator was written in c++ and can be executed on several platforms. The program consists of three main components: a fountain simulation algorithm that drives the particle system, a data collection scheme to extract data from the particle system, and a testing procedure to obtain data in a systematic way so that it can be analyzed and compared to the results of our model.

4.1 The fountain simulation algorithm

Algorithm 2 The fountain simulation algorithm

```
while still running do
  spit out 40 new particles from the fountain nozzle
  assign an initial velocity to each new particle
  for each particle do
    force acting on particle := gravity + viscosity + wind
    use current force to calculate the change in the particle's position
    update the particle's position
    if the particle has hit the ground then
      remove the particle from the simulation
    end if
  end for
end while
```

Algorithm (2) shows the basic structure of the fountain simulation algorithm. This piece of the simulator drives the particle system from which data is collected.

The first question that arises is how many water "particles" might a typical fountain have? We address this question by making the assumption that it makes no difference whether we let our fountain have 5000 particles or 5 million. What we are really interested in is the proportion of particles that fall outside the

radius of the pool. We just need to use enough particles to get a good statistical sample that can be analyzed (see the later section on data collection). With this consideration in mind, we now fix the number of particles used by our simulator:

Design note: For the remainder of this paper, we use 20,000 particles for all fountain simulations. This means that the fountain nozzle emits 20,000 particles over some period of time, and the simulation halts and data collection stops once all 20,000 particles have hit the ground. This is a high enough number to get accurate data, and low enough that it does not take very long to do a single experiment with the simulator.

The operation of the particle system is defined by Algorithm (2). Each simulation loop updates the particle system to represent the changes that occur in a very small interval of time (say 0.001 seconds). At the beginning of each simulation loop, 40 new particles are emitted from the fountain nozzle. Each new particle is given an initial velocity to set it in motion. The magnitude of the velocity is specified as an input to the system, and a small random factor is added to it to represent the fact that no real fountain nozzle can spit out water at a precisely constant speed. Then the direction of the particle's velocity vector is allowed to vary randomly over the fountain nozzle angle.

After new particles have been added to the system, the force acting on each particle is updated according to Equations (1) and (2). The total force acting on each particle is the sum of the gravity, wind, and viscous forces as shown in Figure (3). Then the force is used to determine the particle's current acceleration, and hence its change in position.

Lastly, the simulator checks to see if any particles have hit the ground. Each particle that has hit the ground is removed from the simulation and the appropriate data about that particle is logged (see the section below on data collection). The simulation stops once all the particles hit the ground.

Design note: Throughout the simulation we assume that the wind speed is constant. This reflects the fact that the interval of time represented by the simulation is small enough that the wind readings from the anemometer do not change significantly.

4.2 The data collection scheme

Algorithm 3 The data collection algorithm

```
fountain height := 0
water spread := 0
while still running do
  for each particle do
    if the particle height > fountain height then
      fountain height := particle height
    end if
    if the particle has hit the ground then
      if distance from particle to fountain nozzle > water spread then
        water spread := distance from particle to fountain nozzle
      end if
      update particle density grid
      update density statistics
    end if
  end for
end while
```

The particle system itself is virtually useless without a way to extract data from the system. This is the purpose of the data collection algorithm, which is the second component in the fountain simulator. The basic structure of this component is shown in Algorithm (3). Every time the particles are updated, the data that we need to keep track of is also updated. There are three important pieces of information that are useful to know:

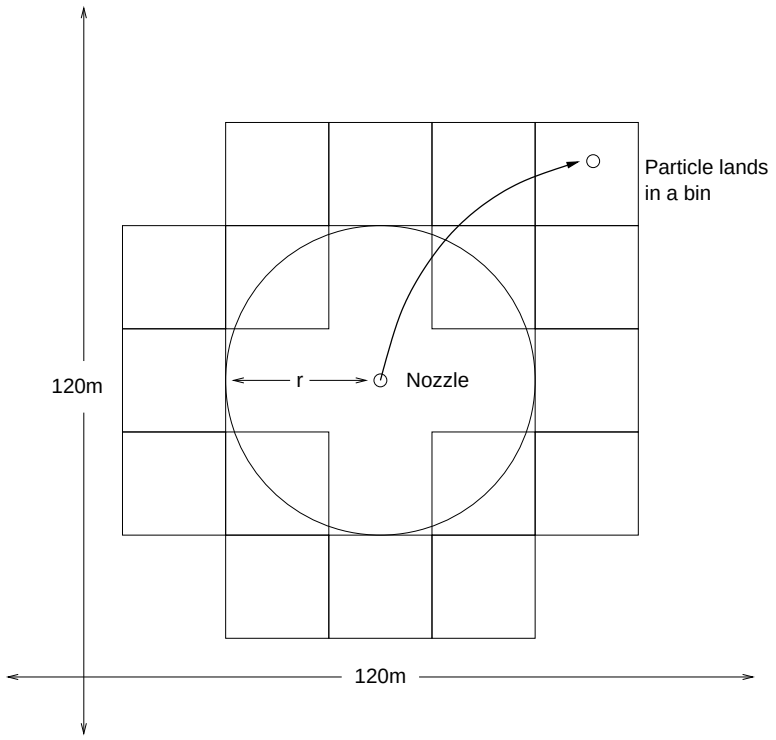

Figure 5: The area around the fountain is divided up into square bins. Each bin records the number of particles that have landed within it.

Data to keep track of:	Fountain height
	Estimated water spread
	Density information

The fountain height is simply the maximum height that any particle attains after being emitted from the nozzle. Every time a particle travels higher than our current fountain height, the fountain height is updated to reflect the new highest particle. The estimated water spread is the farthest distance from the nozzle that a particle reaches by the time it hits the ground. This information is updated every time through the loop in a similar manner to the fountain height.

The density information is the most important piece of data since it will be used by the testing procedure to construct the final output of the simulator program. By density information we mean the number of particles that have hit the ground in a certain area outside the pool radius. This is a measure of how much water spray is going outside the pool, and thus measures how soaked passersby are getting. The idea is to divide a large rectangular area of ground into discrete patches, or bins, using a grid as in Figure (5). Then the data collection algorithm keeps track of how many particles have fallen into each bin. So at the end of the simulation, all particles that fell outside the pool radius are distributed somehow across all of the bins. One bin might contain 10 particles, or hits, another might get 500, *etc.* Note that most bins will probably get 0 hits.

Design note: We only keep track of the particles that land outside the pool radius, because we do not care about what happens to the water that lands inside the fountain pool. So although all 20,000 particles will hit the ground, the density grid might register only 5000 hits if $15,000$ particles land inside the pool.

Design note: For all of our tests, we limit the particle spread to a 120×120 meter square centered on the fountain nozzle. Any particles that leave this square do not register as hits when they land. In addition, each bin is 1 square meter.

The bins provide a measure of the concentration of water spray in that particular area. The reason we care about spray density is that we can achieve a more spectacular fountain effect if we let some of the spray leave the fountain area. The problem is determining what an acceptable amount of spray is. If it is just a light misting, then passersby are not likely to mind, but if it is a torrential downpour, then people are going to complain! So we must define a threshold that serves as the maximum allowed number of hits per bin. We call this the soak threshold. So the problem we are trying to solve can be re-stated as follows:

> What is the maximum initial particle speed that we can use such that there are no bins that register a number of hits greater than the soak threshold?

It is evident that the soak threshold represents how much spray the passersby find acceptable.

Design note: For all of our tests, we use a soak threshold of 50 hits per bin. We found through trial and error that 50 hits per bin represents a reasonable light misting that we personally would not mind experiencing. It is obvious, however, that this is a highly subjective value that must be determined through experiment for each real fountain that our model is applied to. For example, some people might not want to get wet at all, in which case the soak threshold is 0 and all water is constrained to remain within the pool. But if the fountain is in the middle of a park on a hot summer day, people might enjoy a pleasant spray of 100 hits per bin. But as we will see, the value of the soak threshold does not affect the pattern of the final output.

4.3 The testing procedure

Algorithm 4 The testing procedure

```
wind speed := 0
initial particle velocity := 0
while not done do
  initial particle velocity := initial particle velocity + 0.1
  run a fountain simulation with current parameters
  output density information and fountain height/radius to a file
  if there exists a bin such that the number of hits in that bin > maximum allowed hits per bin then
    output current wind speed and initial particle velocity from previous loop to a file
    wind speed := wind speed + 1
    initial particle velocity := 0
  end if
end while
```

So now we have a particle system and a way of getting useful data out of it. But what do we do with our reams and reams of data? The answer is the third and final component of the fountain simulator: our testing procedure. The algorithm for the testing procedure is shown in Algorithm (4). The idea behind it is simple:

Testing Idea: Run the simulator several times for varying wind speeds ranging from a light breeze to gale force winds. For each wind speed, repeatedly simulate the effects of different initial particle speeds and determine the highest initial speed we can use without exceeding the soak threshold. Output this initial speed and the current wind speed to a file and then plot this data as a graph. The graph will tell you what initial speed to use for a given wind speed.

The goal of the testing procedure is to compute the data pairs required for a wind speed vs. desired initial particle speed graph. Then this graph can be compared to the results of the symbolic model to see how well the model agrees with our simulated fountain.

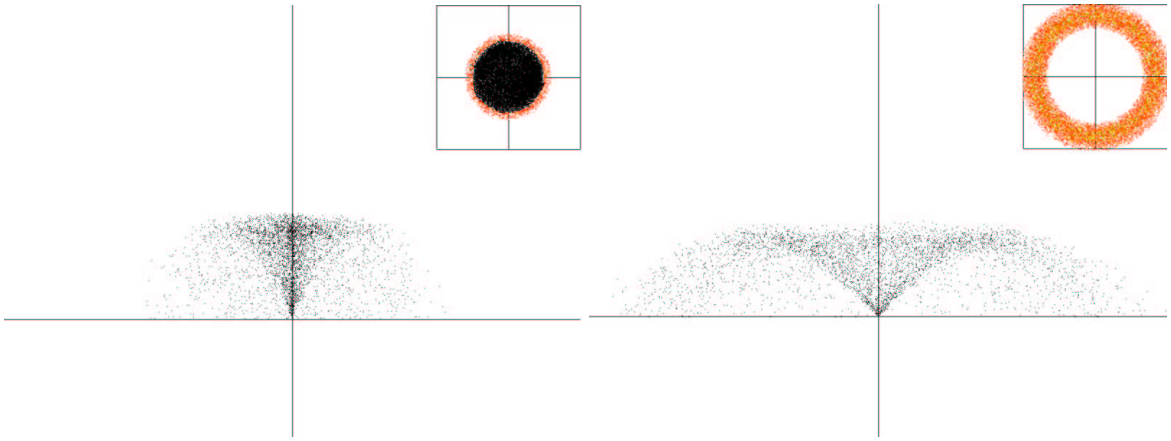

Figure 6: (Left) A screenshot of the fountain visualization. The fountain is depicted from a side view. The upper-right corner shows a top view, with darker particles representing those that fell within the pool.

Figure 7: (Right) A second screenshot of the fountain visualization, this time for the second fountain example.

4.4 Summary

Keeping the operation of the three components of the fountain simulator in mind, we can define a set of initial conditions that serve as input to the simulator:

Input Specifications: Fountain nozzle angle θ
Radius of the pool r
Soak threshold s

The fountain parameters are θ and r. s is the maximum allowed number of hits per bin. This number represents the acceptable amount of spray for a certain region outside the fountain pool. The higher this number, the more tolerant the passersby are of spray. If s is zero, then we have the special case represented by the symbolic model, where no water is allowed to fall outside the pool radius. The value of s would most likely need to be determined experimentally for any given real fountain. So given these input conditions, we have the following output from the simulator:

Output: A graph of wind speed vs. desired initial speed

That is, given a particular wind speed reading from the anemometer for the fountain specified by θ and r, the corresponding initial speed on the graph is the initial speed that should be given to the water particles being emitted from the nozzle so that the maximum number of hits per bin does not exceed s. The output of the simulation is a graph of the same type as the output from the numerical method developed in the physical model. The key difference is that here we take the soak threshold into account, whereas the numerical algorithm gave the special case that the soak threshold was equal to zero. Thus, the method described in this section is an extension of the method of Algorithm (1).

5 Sample Applications

5.1 A typical fountain

Suppose we have a fountain, as in Figure (1), with parameters $r = 5\text{m}$, $h = 0\text{m}$, and $\theta = 15°$. We can get an idea about where the water particles will fall by running Algorithm (2) with a representative initial water velocity. The associated visualization output, exemplified in Figure (6), gives us a good idea of how the water will be distributed.

Now, we wish to determine what the initial velocity of the water particles in the fountain should be for a given wind speed so that we will not soak nearby pedestrians. Algorithm (4) above computes exactly this.

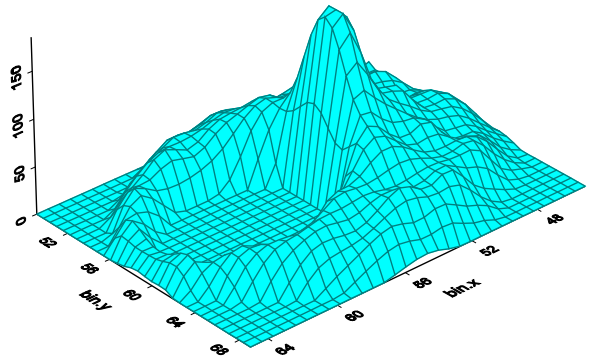

Figure 8: A density plot depicting bin coordinates on the x and y axes, and the number of particle hits in that bin on the z axis. The circular dip represents the pool itself—we did not count particles that fell into the pool (see the section on the data collection scheme). There is a 30m/s wind along the negative x axis.

However, in doing so, it will have to run a particle simulation for many sample wind speeds and many sample initial velocities. For each run, it divides the universe into bins and examines how many particles fall into each bin.

Figure (8) shows a density plot taken from a sample of the program's output. This particular example shows the particle density distribution for a wind speed of 30m/s in the $-x$ direction, and an initial particle speed of 13.2m/s.

Finally, we take a look at the data output by Algorithm (4). Shown in Figure (9) is the initial particle velocity needed to keep all of the bins at an acceptable soaking level for a given wind speed. Three possible soak thresholds are shown—a threshold of zero indicates that all of the water should fall within the pool, and thresholds 50 and 100 indicate varying levels of misting.

Figure (9) is then the final solution for this sample problem. Given any wind speed, it will return an initial velocity at which to set the fountain so that no area will be soaked more than some desired threshold. Note that the graph is quite similar to the graph without drag in Figure (4), as one would expect.

5.2 Another fountain

Now, we will proceed much as in the previous example, except that we will consider slightly different sort of fountain. We again set parameters $r = 5$ and $h = 0$, but this time we will not specify that the water is shot from the fountain between 0° and 15° (as when θ was 15°. Instead, we suppose that the water is shot between 30° and 40° from the vertical.

Once again, our Algorithm (2) and associated visualization gives us a good idea of what this fountain looks like. This is shown in Figure (7). We can see that this time almost all of the water falls outside of the pool at our "typical" initial velocity of 10m/s.

After producing all of its density data, we finally arrive at a plot that is similar to the last fountain, shown in Figure (10). As before, this plot depicts initial velocity in terms of the wind speed. As one might expect, the initial velocities are lower than their counterparts in the first example, even at low wind speeds. This is due to the lower angle at which the water is shot out of the fountain.

Another thing that we notice about this graph is that the initial particle velocity data is fairly coarse. At the expense of computing time, one might produce a much finer graph. The data from the graph should

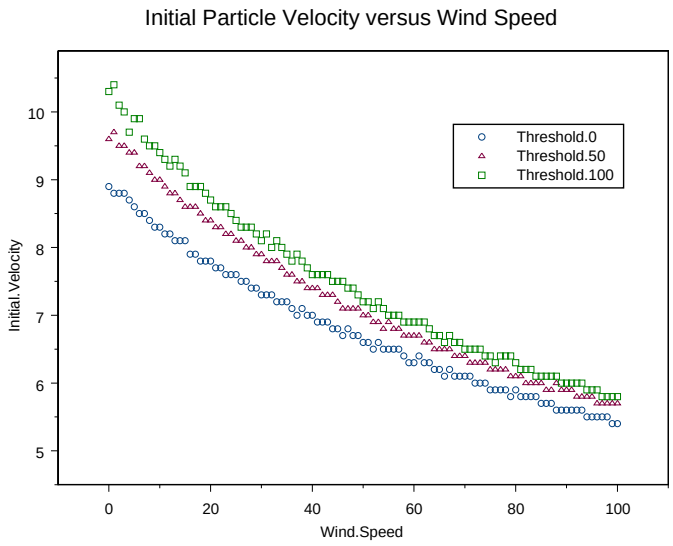

Figure 9: This plot gives the v_0 value necessary to keep fountain water in the pool in terms of wind speed, for three different soak thresholds.

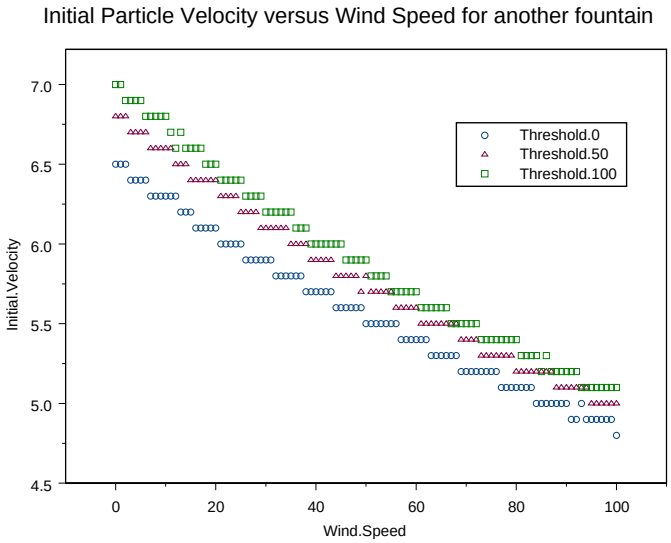

Figure 10: This is a second plot which gives v_0 in terms of wind speed, this time for the second fountain example.

be precomputed, and then hard coded into some kind of ROM on the controller for the fountain. Thus, the only limitation to the resolution of the data is the available storage space in the controller.

5.3 Other fountains

The methods used in this section will apply just as well, with only slight modification, to many other fountain designs. For example, one might have a square or oddly shaped pool. In this case, the wind direction as well as speed will need to be considered. Also, one may have several nozzles at different locations, each with its own initial particle velocity.

The modifications necessary to consider such kinds of fountains are very simple to apply to the computer algorithm, but would get quite messy when applied to the symbolic model.

6 Analysis of the Model

6.1 Strengths of the model

On the whole, we have found our model to be quite natural and easy to apply. Here, we list some of the advantages of our approach to the problem.

1. The particle simulator is flexible enough that it may easily be modified to provide a solution for other types and shapes of the fountain or the pool.
2. Our model is simple enough that the calculations can be performed by an on-board processor built into the fountain's control systems. The appropriate wind speed vs. desired initial velocity graph could be pre-computed and stored as a table on a small ROM for the fountain in question. Then the fountain control system would simply compare the current anemometer reading to the wind value in the table, select the corresponding initial velocity, and adjust the pressure of the fountain's pumps accordingly.
3. Since there is a relationship between the soak threshold and the radius used in the model, we can easily extend the model so it can be used for any soak threshold. In other words, we need not specify that all the water must fall inside the fountain. This allows us to use the same type of model to produce more spectacular fountains.

6.2 Weaknesses of the model

Of course, there are many ways to attack a problem such as this one. In this section, we discuss some of the drawbacks of our approach to the problem, and some things that could have been done to deal with the issues.

1. The most fundamental difficulty with our model and methods is that it is virtually impossible to test them against an actual fountain. More specifically, it is hard to be sure that the effects of parameters such as v_0 and v_w on the model accurately reflect the true behavior of water and wind.

   The best way to address this problem would be to calibrate constants in the software to a particular fountain. For example, the initial particle speed parameter is an abstraction of the true behavior of a fountain. What is needed is a constant multiplier so that the model's initial particle speed will match the true power of the fountain pump. Another example is the wind speed, which may not truly impart the force that we calculate on the water particles, especially since we have only guessed at their size.

   Determination of these constants may easily be accomplished with a small amount of data collection and statistical observation. The behavior of the model may be compared against the true behavior of the fountain, and the constants may be updated as is appropriate.
2. Another difficulty that we have all but ignored comes in knowing the true behavior of wind near a fountain. The speed and direction of wind at the anemometer several stories up may not be representative of the wind flows near the ground. This is because the wind may break as it hits nearby buildings, and swirl the water in unpredictable directions.

3. Our model does not take into account the complicated effects that wind can have on a water drpplet. For example, we did not model the possibility that the wind may cause lift in the water particles. It is conceivable that a strong wind will not just act laterally on a rain drop, but also impart a lift force on it. Moreover, it is the case that the water particles deform as they fly through the air, and eventually split into smaller water particles several times before they hit the ground.

References

[1] Dobler, Wolfgang. "Teaching Handout for Introduction to Astrophysical fluids." `http://antares.ncl.ac.uk/~dobler/teaching/MAS218/handout3.pdf`. January 25, 2002.

[2] Douglas, Craig. "Mathematical Models of Physical Problems." `http://mailhost.ccs.uky.edu/~douglas/classes/ma721-f98/douglas.090198/dust.ppt`. September 15, 1998.

[3] Halliday, Resnick, Krane. Physics, 4th edition. John Wiley & Sons, 1991.

[4] Lide, David. CRC Handbook of Chemistry and Physics. CRC Press: 2001.

[5] W. T. Reeves. "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects." Computer Graphics, vol. 17, no.3, pp. 359-376, 1983.