Positioning and Moving Sprinkler Systems for Irrigation

Team #836

February 6, 2006

Abstract

This document describes an optimal solution to the problem of irrigation on a rectangular field where the sprinkler systems water dispersion pattern is elliptical and the length of the irrigation system is not adequate to cover the entire field.

We have derived a model from the stated empirical data, and constructed two separate computer simulations of the model, each with model with it's own approach to the problem. We have verified the results of our simulations in a third manner using that wonderful branch of mathematics, the Calculus. We have analyzed our results statistically, and found an optimal solution. In our optimal solution, we have placed four nozzles unevenly along our sprinkler system, one at each end and one five meters from each end. This configuration allowed the farmer to have to only move the sprinkler system once every seven hours, for twenty-eight hours. The field received the requisite amount of water, and at no time did the water reaching the field per hour exceed the allotted maximum. Water distribution was also highly uniform, with only a 5.35% standard deviation from the mean.

Contents

Chapter 1

Introduction

A farmer has a field 30 meters in width by 80 meters in length. The farmer has a series of pipe segments with which he can construct a pipe system of 20 meters in length. What is the most efficient method for the farmer to water his field? Specifically, how many nozzles should the farmer incorporate into his pipe system, and how far apart should the nozzles be spaced? Also, where should the farmer put the sprinkler system in order to minimize the number of times the sprinkler system has to be moved to water the field. These are the questions which we have attempted to answer with our model.

Chapter 2

The Model

2.1 Assumptions

In constructing our model, we have made several defensible assumptions.

1. Water is a non-viscous, non-turbulent, irrotational, and incompressible fluid.

   This assumption is taken for granted in many pieces of literature. The effect of the properties of water we have left out is negligible in many applications, and leaving out the calculations of these properties will greatly simplify the derivation of our model.

2. Pressure loss in a 20 meter length of pipe is negligible.

   Before neglecting pressure loss, we first produced a calculation for the pressure loss in the length of pipe. Initially, we calculated the Reynolds number, given by

   $$\mathrm{Re} = \frac{\rho v_s D}{\mu}$$

   where ρ is density of water, v_s is mean fluid velocity, D is diameter of pipe cross section, and μ is absolute dynamic fluid viscosity. Finding $\mathrm{Re} = 3.177$, we determined it to be a laminar flow. We then applied our results to the Swamee-Jain Equation,

in order to determine the friction factor, f, in a full flowing circular pipe:

$$f = \frac{1.325}{\left[\ln\left(\frac{e}{3.7D} + \frac{5.74}{Re^{0.9}}\right)\right]^2}$$

where e is average roughness height, and Re is the Reynolds number. This we found to be the unit-less constant 2.650. Given the previous values, we were then able to calculate h_f, the head loss, given in the D'Arcy-Weisbach formula as:

$$h_f = f \cdot \left(\frac{L}{D}\right)\left(\frac{v_s^2}{2g}\right)$$

where L is length of pipe, and g is the acceleration force due to gravity. Finally, we used conversion factors to render the head loss into pressure loss using the following formula: $P = h_f \cdot\ 9.790$. Thus, we determined that the pressure loss due to friction along the pipe's length was 26.80 kPa, leaving our pipe with over 390 kPa of pressure to force the water through the nozzles.[6]

3. The Newtonian physical model holds for a droplet of water.

   Of course, we know that Newtonian physics does not hold for a droplet of water, nor for any other particle in the universe. But it does provide a simple and fairly *accurate* model for kinematic systems dealing with macro-particles at sub-light speeds.

4. The field is topographically uniform and at the surface, pressure is one atmosphere (101.325 kPa).

   Farmers tend to keep their fields as flat as possible, precisely because it creates an easier watering situation. According to Evans [2], irrigation systems like on our farm generally operate on land with less than a 6% slope. Although possible to account for the many environmental factors globally experienced by farmers (atmospheric pressures, temperature ranges, and wind velocities for example), such details would increase the complexity of our model beyond necessity. It is enough to assume moderate wind speeds, arable land, and note that as the elevation drops from 8850 meters to -400 meters, the atmospheric pressure rises from 31 kPa to 106 kPa. Such

a difference, though large, is only slightly more than half of the amount of pressure flowing within our irrigation pipes.

From these basic assumptions we have constructed our model.

2.2 Initial Derivations

Given F_{in}, the water flux through the inlet, and d_{in}, the diameter of the inlet, then v_{in}, the velocity of water through the inlet, is

$$v_{in} = \frac{4F_{in}}{\pi d_{in}^2}.$$

If n is the number of sprinklers, and $k = 1, 2, \ldots, n$, the continuity equation [3] gives that

$$A_{in}v_{in} = A_1v_1 + A_2v_2 + \cdots + A_nv_k.$$

That is, the product of the area and the fluid speed at all points along a pipe is constant for an incompressible fluid. Notice, $A_1 = A_2 = \cdots = A_n$. Thus,

$$A_{in}v_{in} = A_s(v_1 + v_2 + \cdots + v_k).$$

Because we have assumed the pressure loss along the length of the pipe negligible, we assume the velocity to be equal across all nozzles. The velocity out of each sprinkler nozzle then is

$$v_s = \frac{A_{in}v_{in}}{A_s n}.$$

By plugging in the known values for area, we find $v_{in} = \frac{1}{\pi}\frac{m}{s}$, and $v_s = \frac{2500}{9n\pi}\frac{m}{s}$.

If θ is the angle of the sprinkler nozzles from the horizontal, then we can determine the maximum vertical velocity v_u of the water stream out of a sprinkler head using the trigonometric identity,

$$v_u = v_s \sin\theta.$$

Similarly, the horizontal velocity v_h is

$$v_h = v_s \cos\theta.$$

The maximum time t_m that a droplet of water can spend in the air is given by the kinematic equation of a particle under constant acceleration

$$t_m = 2 \cdot \frac{v_u}{g}$$

where g represents the acceleration of gravity. Then r_s, the maximum radius of water distribution for a sprinkler, and A_s, the area of the water distribution for a sprinkler, are given by

$$r_s = v_h t_m \text{ and } A_s = \pi r_s^2.$$

By combining the terms within v_h and t_m, we discover $r_s = \frac{v_s^2}{g}$. Then the water flux F_s to the ground around a sprinkler is given by

$$F_s = \frac{F_{in}}{nA_s}.$$

The worst case for water accumulation around a particular sprinkler s is given by the following. Given n sprinklers, we can find the radius r_s for each sprinkler as before. We can also find the distance between each sprinkler ΔS. If we take

$$m = \left\lfloor \frac{r_s}{\Delta S} \right\rfloor \pmod{n+1},$$

then we have the number of sprinklers within one radius of s. Thus, the ground immediately around s is receiving spray from $2m+1$ sprinklers. Then F_σ, the largest possible amount of water accumulation for any point along the pipe system, is given by

$$F_\sigma = F_s(2m+1).$$

2.3 Optimization

2.3.1 Max Flow

Our first condition for determining an acceptable irrigation system was that any given nozzle must put out the most water it can. Since the radius of spray is proportional to the

square of the velocity ($r_s = \frac{v_s^2}{g}$) with the area proportional to the radius squared, and the velocity v_s inversely proportional to the number of sprinklers, the maximum spray velocity would come from the least possible number of sprinklers. Since one is the least nontrivial number of sprinklers possible in the system, we determined v_s and the area to both be maximized for $n = 1$, and hence the flow, or product of the area and velocity, would also be maximal.

2.3.2 Minimum Saturation

Given the condition that the entire field must be sprayed with .02 meters over the course of 4 days. This translates to an average flow of $5.787 \times 10^{-8} \frac{m}{s} \cdot A_s$ With just one sprinkler, we can ideally cover an area of $\pi \cdot (\frac{v_s}{g})^2 m^2$, or about $1.996 \times 10^6 m^2$. In reality, this number is impossible, because such a velocity would increase the wind velocity relative to the water, decreasing droplet size, and thus greatly increasing the quantity of water lost due to evaporation.[5] Also, we already know that the total flow leaving the sprinklers must equal the amount entering the pipe, which was given as $150 \frac{L}{min}$. According to the Rain Bird catalog, a high quality high pressure rotary nozzle is capable of releasing $34.8 \frac{L}{min}$, which is less than a quarter of what would be necessary in this system.[7] Disregarding such impossibilities for the moment, we find that the rate of saturation would be

$$\left(\frac{.0025 \frac{m^3}{sec}}{1.996x10^6 m^2} \right) = 1.25 \times 10^{-9} \frac{m}{s}$$

Note, this rate occurs over the entire area, of which our plot of land only represents .15%. Therefore, it would be safe to state that

$$.15\% \times 1.25 \times 10^{-9} \left(\frac{m}{s} \right) \times 86400 \left(\frac{sec}{day} \right) \times 4(days) = 6.5 \times 10^{-9} \left(\frac{m}{s} \right)$$

Since this figure is well below our intended minimum precipitation, there must be more than one sprinkler head on our irrigation pipe.

2.3.3 Maximum Saturation

With more than one sprinkler placed along our pipe, we know that any radius greater than 10m will lead to some overlap. We also know, from the kinematic equations, that a radius of less than 10m will have a velocity $v_s = \sqrt{10 \times 9.81} = 9.90\frac{m}{s}$. Note, from the continuity equation, the sum of the sprinkler velocities must be proportional to our input velocity:

$$\frac{A_i v_i}{A_s} = v_1 + v_2 + ... + v_n,$$

thus $88.419\frac{m}{s} = v_1 + v_2 + ... + v_n$. The greater the variance of nozzle velocities, the more varied the radii of the circles to which they project water. Smaller circles would pour greater amounts of water closer to the pipe, while greater circles would saturate larger areas with less water. Whether this would eliminate all possibility of uniform saturation or just create potentially more work in number of times the pipe must be moved will be discussed further. The one case in which no extra work is required to symmetrically overlap with covered areas, is when the nozzles are equally spaced apart. From a theoretical standpoint, it seems reasonable to guess that this is the best system. First, because if two nozzles are closer to one another than to their other neighbors, water should accumulate to a greater degree in their areas of overlap, and to a lesser degree in areas where they do not. Secondly, it would certainly entail less consideration on the part of the workers setting up the pipe. Consequently, it is reasonable to restate the continuity equation in this simplified form,

$$v_s = \frac{88.419}{n}$$

Taking discrete n-values, spacing them evenly along a 20 meter pipe, and determining the associated radii, we were able deduce that $A_o = A_s - A_{bt}$, where A_o is the area of overlap, A_s is the area of the circle around a sprinkler, and A_{bt} was the area between the "top" and "bottom" circles. To find the area between circles, we set the center of the pipe as the Origin of the x-y plane, in order to take advantage of radial symmetries. From there, we calculated the equations for circles spaced along the origin. For ease of integration, the pipe ran from $y = -10$ to $y = 10$, so the equations for the circles were of the form

$y_n = \sqrt{r^2 - x^2} + b$, keeping in mind the negative values. Using calculus, the area between the "top" curve and the "bottom" curve was written as

$$4\int_0^r y_1 - y_n dx = 4\int_0^r \left[\sqrt{r^2 - x^2} + 10 - \sqrt{r^2 - x^2} - 10\right] dx$$

Canceling the square roots, we end up with a linear function with respect to radius, $A_{bt} = 80r$. Thus, the area shared by all the sprinklers is simply $A_o = \pi r^2 - 80r = r(\pi r - 80)$. Finally, we must divide the total flow by the overlapped area, $\frac{.0025 left(\frac{m^3}{sec} right)}{A_o}$, to determine the rate of saturation in meters per second.

In Table 2.1, we see that four equally spaced nozzles provides the highest rate of flow without exceeding the value 2.083×10^{-6}. Finding the intersection between the function $f(x) = \frac{.0025}{r(\pi r - 80)}$ and $f(x) = 2.083 \times 10^{-6}$, we find that only those values beneath $x = 4.666$ satisfy our requirements. So long as the soil can properly absorb the moisture, higher influxes of water are preferable to lower ones. Since x is a continuous representation of the discrete n-values, we can safely choose $n = 4$ as the optimal number of nozzles. Consequently, so long as we have 4 or less sprinklers, we can let the pipe sit at any location for an arbitrary amount of time without surpassing the .75 cm/hour saturation limit. Rates beyond this amount can have many deleterious effects on the soils and plants being hydrated. To name a few, supersaturated soil will expel oxygen, causing to anaerobic conditions that can lead to root rot; decreased nitrogen uptake severely stunts plant growth; and water will build up. If lacking a retainer systems will erode the topsoil.[**?**].

n	r	A	$\left\lvert\frac{.0025}{\Delta A}\right\rvert$
2	199.24	108860	2.298×10^{-8}
3	88.584	17565	1.423×10^{-7}
4	49.829	3814.0	6.555×10^{-7}
5	31.890	643.75	3.883×10^{-6}
6	22.146	230.90	1.083×10^{-5}

Table 2.1: Rates of Flow.

Upon closer inspection of the possible placements of the sprinklers, we find that regardless of the placement of the middle nozzles, the area being pumped with water from all sources will remain unchanged. This means that we can arbitrate the locations of the inner nozzles to mediate the size of the areas receiving spray from only one source, two sources, and three sources. Depending on the location of the irrigation system within the field, specifically the distance from the system to the boundary of the farm plot, we may need more flow to cover further out. In which case we would locate the middle nozzles equidistant from the midpoint of the pipe and quite near one another. This flow distribution would minimize the areas of land receiving water from two sources, while increasing the areas receiving one and three sources by an equal amount. Alternately, placing the pipes very near the boundary of the land would necessitate a decrease in the amount of water flowing radially out and prompt placement of the inner nozzles to be nearer the endpoints. In either case, the fixtures would be need to be equidistant to the midpoint of the pipe for the sake of uniform flow. Because the area of land receiving differing outpourings of water changes linearly with the distance between the inner points, calculus alone cannot help in distinguishing between the solutions. In effect, all possible spacings are equally good, so long as the farmer places the pipe the proper distance from its previous locations and the edge of the field.

2.4 Dispersion Profile

In the construction of our model, we at first considered an constant dispersion profile. That is, we assumed that the flow from the nozzle would disperse itself evenly across the the entire dispersion area. The dispersion profile is diagrammed in Figure 2.1.

From the very start we realized that this constant dispersion profile was an unrealistic assumption. The literature backed this. So, in addition to this constant dispersion profile, we considered another in which the hourly water accumulation decreases linearly with the distance from the nozzle. From our research, it seems that this is the most sought after type of dispersion profile as it allows sprinklers to be spaced in such a way that all areas

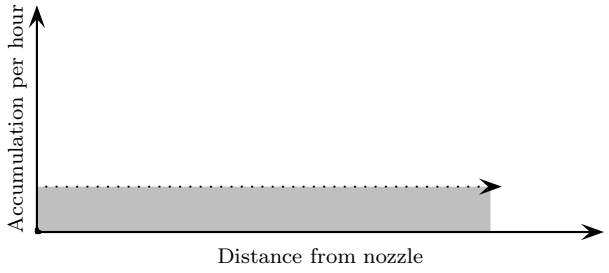

Figure 2.1: An constant dispersion profile.

receive an even amount of precipitation. A diagram of the linear dispersion model is shown in Figure 2.2.

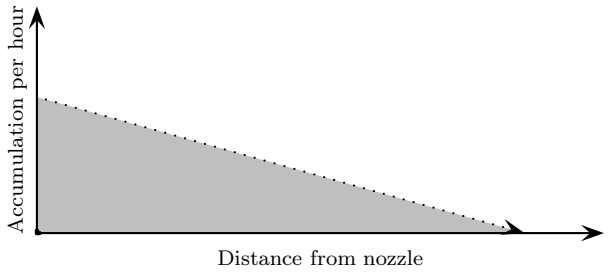

Figure 2.2: A linear dispersion profile.

When adjusting the dispersion profile, it is important to maintain the volume formed by the rotation of the profile around the y axis. If the volume of the rotation is not maintained, the dispersion profile is inaccurate because water has then been introduced into the system. A simple calculation shows that in the linear dispersion profile, the depth of water accumulation at the center must be exactly three times that of the constant dispersion profile. To wit,

$$\pi r_1^2 h_1 = \frac{1}{3}\pi r_2^2 h_2.$$

But, because the maximum radius of the dispersion profile is the same for all dispersion profiles with a fixed number of nozzles, $r_1 = r_2$, thus we have that

$$3 \cdot h_1 = h_2.$$

That is, the center height of our linear dispersion profile is exactly 3 times the height of our constant dispersion profile.

Chapter 3

The Simulation

Two separate computer simulations were constructed based on our Newtonian model. A simulation was written in the C programming language. This simulation provides data on relative water distribution; i.e, it gives no figures in centimeters of water. It is designed only for finding a configuration that applies water as uniformly and efficiently. A second simulation in Python attempted to accurately model the amount of water delivered to the field per hour, and to measure exactly how much time was required for full water dispersion onto the field. Both of these simulations include a 10% loss of water delivered to the ground, to account for possible water loss due to evaporation, plant canopy, weather, and other such factors with the intent of providing a model of the worst-case scenario.

3.1 A Simulation in C

3.1.1 A Relativistic Approach

This simulation only provides data on relative water distribution; i.e, it gives no figures in centimeters of water. It is designed only for finding a configuration that applies water as uniformly and efficiently (fewest pipe positions) as possible. It was decided that finding a good configuration was more important that finding a specific duration, as a variety of factors in practice will affect the duration; weather, season, crop, while the factors that

affect the position are less likely to change (the exact configuration depends primarily on the nozzle type). To this end, the simulation calculates the standard deviation as a percentage; this made it easy to compare different configurations (vs. eyeballing output images).

This simulation uses a file with positions for nozzles on the pipe, and locations for the pipe in the field enabling many different configurations to be easily tested. The code relating to the actual simulation is entirely in the *simulate()* and *spraywater()* functions. The remainder of the code is for reading in the configuration file, and outputting contents of the field array to a PNG image. The source code of this simulation can be found in *irrigation.c* this file is included in Appendix B.

3.1.2 simulate()

The *simulate()* function determines the positions where a nozzle will be in the field, by walking the list of pipe positions and then calling *spraywater()* for every nozzle on the pipe. Pipes may only be aligned horizontally or vertically; it was deemed unlikely that an angled configuration would be optimal in a rectangular field. A two dimensional field array is maintained for holding the amount of water applied to the field; it uses 20 elements per meter.

3.1.3 spraywater()

The bulk of the work is done in the *spraywater()* function; this function calculates water flow (from the number of nozzles), then calculates the velocity and radius that the nozzle will cover. The spraywater function then iterates over the field array, calculating the distance of each element from the nozzle and applying some amount of water proportional to the radius if the element is within the radius.

We used a linear dispersion model, with the highest dispersion next to the nozzle, decreasing to zero at the maximum radius. This seemed to be the preferred sprinkler type for irrigation in the literature, and gave us the best results. We also tried reverse linear

dispersion (zero at the nozzle, highest at the maximum radius - we were able to find one sprinkler advertised as having this dispersion profile) as well as constant dispersion, but the results with these models were not nearly as uniform.

3.1.4 writeimage()

The *writeimage()* also calculates statistics, as the array is outputted to an image. It calculates the maximum and minimum values and their ratio, as well as the average and standard deviation. The standard deviation uses the usual formula, first calculating the average, then iterating over the array a second time to calculate the variance.

3.2 A Python Simulation

The Python simulation attempted to accurately model the quantifiable variables in our model. It measured the amount of water precipitated to all parts of the field per hour, and also the total time required to ensure that all parts of the field would receive the appropriate amount of water.

This simulation functioned by providing a class for all components in our model. A class was created to model the field, the sprinkler system, and for each sprinkler. A fourth class drives the simulation. The source code of this simulation can be found in *irrigation_sim.py* and *irrigation_tools.py*. Both of these files are included in Appendix A.

3.2.1 The Field Class

The *Field* class has four attributes: *max_hourly_water* describes the maximum allowable water accumulation in one hour for any portion of the field, *width* and *height* describe the height and width of the field, *sectors* is an array containing the total water accumulation for points on the field, and *resolution* describes the number of points held in the array *sectors* for each unit of field dimension.

The *Field* class contains only one method important to the simulation. This is the *apply_water* method. The *apply_water* method receives an instance of a Sprinkler class object, and from it deduces how much water will fall were on the field in one hour. This method is the primary driver of the Python simulation.

3.2.2 The SprinklerSystem Class

The *SprinklerSystem* class is initialized with n, the number of sprinklers in the system, and *theta* the angle of the sprinkler nozzles. The *SprinklerSystem* class has five attributes, *inflow* is the water flux into the system, x and y describes the sprinkler system position on the field, *phi* describes the the angle of the sprinkler system in the plane of the field, *sprinklers* is a python list containing all of the *SprinklerSystem*'s child sprinklers.

The *SprinklerSystem Class* class has two methods of importance. The method *run_sprinklers* is a Python construct known as a generator. It yields sprinkler objects iteratively so that not all sprinkler objects must be returned at once. Instead the are returned as needed by the caller. The method *move* simply updates the *SprinklerSystem*'s position and all of the child sprinkler's positions as well.

3.2.3 The Sprinkler Class

The *Sprinkler* class is the core of the simulation. The class is initialized with the nozzle flow of the sprinkler and the sprinkler's angle over the field. The *Sprinkler* class has several attributes: *distance_from_root* describes the sprinklers distance from the sprinkler system root point, x, y, and *phi* give the root point of the sprinkler system, *nozzle_diameter* is the nozzle diameter given in the problem description, *nozzle_flow* is the water flow through the sprinkler. From the nozzle flow and nozzle diameter the following parameters are computed and saved as attributes: *nozzle_velocity* is the water velocity out of the nozzle, *dispersion_radius* is the maximum range of a the water jet, *dispersion_area* is the area computed from the sprinkler's dispersion radius.

The *Sprinkler* class contains several important methods. The most important being

the *dispersion_profile* method. This method returns the hourly water accumulation as a function of the distance from the sprinkler nozzle. We have implemented this method in the two distinct ways described in the earlier section describing the derivation of our model. The *Sprinkler* class also has *move* as a method. This method simply updates the sprinklers reference to the root point of the sprinkler system. The *coords* method returns the actual field coordinates of the sprinkler nozzle in a tuple.

3.2.4 The Simulation Class

The *Simulation* class drives the simulation. The *Simulation* creates two instances of the *Field* class, one for hourly water accumulation and one for the total water accumulation, and also creates one *SprinklerSystem* class.

The *Simulation* class's primary method is *run_simulation*. This method contains a list of sprinkler system positions. Each position in the list corresponds to one hour of time. The *run_simulation* method iterates through each position, first setting the sprinkler to the position, and then iterating through each sprinkler in the sprinkler system, applying water to a temporary field class. The temporary field is checked for the maximum water application for the hour, and the water application for the hour is added to the total water accumulation for the field. After all the sprinkler system positions have been completed, the minimum accumulation on a sector is deteremined as well as the maximum, and the simulation is terminated.

Chapter 4

Results

We have reached the conclusion that 4 nozzles is the optimal number.

4.1 Simulation in C

With the C-Simulation, we tried a variety of nozzle configurations and sprinkler system positions, optimizing each by hand to attain the lowest standard deviation possible. The results were not quite what we expected, in particular that a uniform nozzle placement was not always ideal. The uniformity of water dispersion was quite a bit better than we expected, and is quite dependent on using a linear distribution model; we tried a few with constant distributions and reverse linear distributions, and the dispersion uniformity was always significantly worse. Below are some of our best results; most of these would never

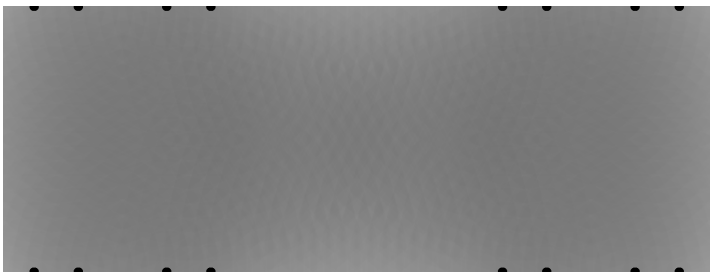

Figure 4.1: Four nozzles, four positions, 5.36% standard deviation.

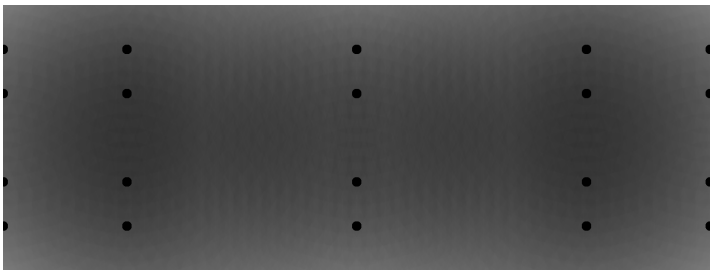

Figure 4.2: Four nozzles, five positions, 7.87% standard deviation.

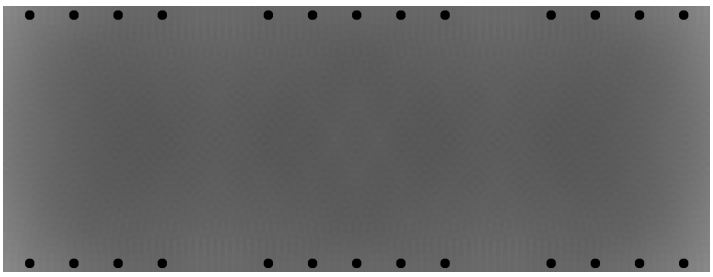

Figure 4.3: Five nozzles, six positions, 5.86% standard deviation.

be used in practice (and some have too many nozzles to be practical; it turned out to be necessary to use no more than 4 nozzles to cover enough area and not oversaturate the field) but they show patterns that might be useful in larger fields.

Figure 4.1 depicts the nozzle arrangement with the smallest standard deviation of water accumulation with a useable number of nozzles.[1] Curiously, the ideal placement on the pipe didn't have the nozzles equally spaced. This would also be a very efficient arrangement (in water useage) if nozzles can be found that will spray in half circles, and the fact that the positions are on the edge of the field ought to make it easier to move the pipe around in practice.

Figure 4.2 depicts the other general configuration we tried. As you can see, with four nozzles it is less optimal in every way than the first configuration. However, if water is scarce and nozzles that spray in half circles cannot be used, this might be preferable to

[1] In all of these figures darker color indicates greater accumulation of water.

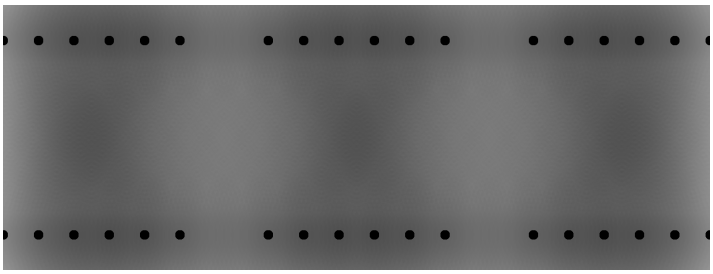

Figure 4.4: Six nozzles, six positions, 8.91% standard deviation.

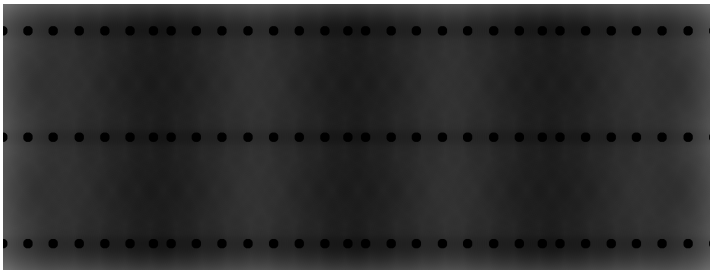

Figure 4.5: Eight nozzles, twelve positions, 5.85% standard deviation.

save water.

In Figure 4.3, we begin to see how as the number of nozzles goes up, the coverage decreases and the number of positions must increase. The coverage is quite uniform, but in practice there will likely be no reason to use this configuration over the first. Figure 4.4 illustrates a continuation of the trends in the five nozzle configuration. Six nozzles are not optimal for this field size, and the standard deviation is higher. While the configuration

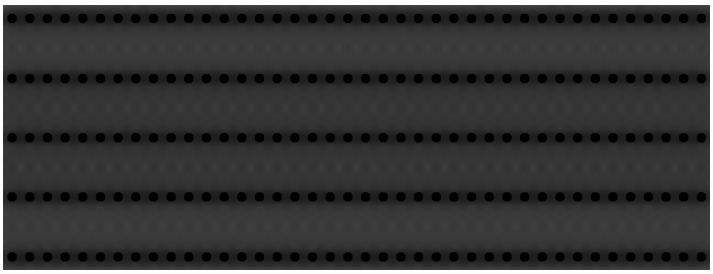

Figure 4.6: Ten nozzles, twenty positions, 4.6% standard deviation.

in Figure 4.5 would be useless on a 80m x 30m field, it does show what a configuration on a larger field would look like. Figure 4.6 shows the densest configuration we tried. As you can see, there seems to be an optimal spacing in large fields dependant only on the sprinkler radius.

4.2 Python Simulation

With the Python simulation, the optimal nozzle configurations and sprinkler system positions found by the C simulation were verified. Additionally, it was found that the field could completely be watered to a two cm depth in approximately twenty-eight hours. The simulation showed that the unevenly spaced four nozzle configuration could apply at most 0.25 cm of water per hour. However, due to the large dispersion radius of the sprinklers, the needed two centimeter accumulation was dispersed in just seven hours.

Figure 4.7 shows the unevenly spaced four nozzle configuration after seven hours of water application. The same field is seen again after fourteen hours in Figure 4.8, after twenty-one hours in Figure 4.9, and after twenty-eight hours in Figure 4.10. It was necessary that the sprinkler system spend at least 7 hours in each of the four position shown in order for full a full two centimeter application of water. However, only four moves of the sprinkler system are required for this configuration of nozzles and sprinkler system positions. In this four nozzle, four sprinkler system position configuration, the most water dispersed to any portion of the field was 2.95 centimeters, the least dispersion was 2.07 centimeters, and the average 2.68 centimeters with a standard deviation of 6%. These results indicate that the farmer should be able to have his field entirely watered in two fourteen-hour days.

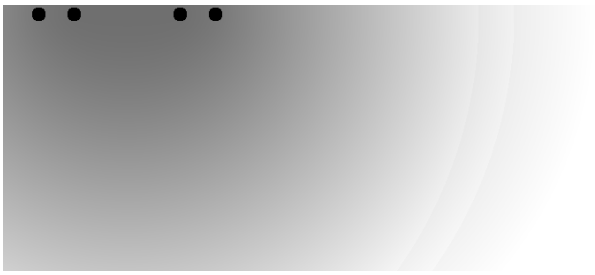

Figure 4.7: Unevenly spaced four nozzle configuration after seven hours.

Figure 4.8: Unevenly spaced four nozzle configuration after fourteen hours.

Figure 4.9: Unevenly spaced four nozzle configuration after twenty-one hours.

Figure 4.10: Unevenly spaced four nozzle configuration after twenty-eight hours.

Chapter 5

Review

5.1 Strengths

Our model is based on the solid foundation of Newtonian physics, which while not being a perfect model for the universe has provided a solid and long-standing foundation for many of the achievements of the past few centuries.

Another strength of our model is that it gives an expected variation in water distribution, which may be useful if there are crops that have a relatively tight tolerance for water requirements.

Both of our computer simulations can be run very quickly and provide an easy mechanism to vary the input parameters. This can be used to test a high volume of nozzle configurations and sprinkler system positions in order to definitively nail down a good solution for any field shape or size.

5.2 Weaknesses

We believe our model to be of significant practical use, but ultimately it rests on a few assumptions which could use more practical testing. Primarily the specific nozzle type will have an impact on the dispersion radius and water dispersion profile, but we currently have

no data on how or even how much. The dispersion radius will also depend on how much a stream of water is affected by air friction; this would be extremely difficult to model due to its chaotic nature, but a small amount of empirical testing could either validate our assumptions or provide new ones. The linear dispersion model is better supported by the literature; it is regarded as being ideal for irrigation purposes by current manufacturers, and in practice the actual water dispersion profiles should be close to our linear model. Any variation may decrease uniformity, but the variation would have to be extreme to signifantly alter our figures. It is also unlikely that any variation in dispersion profile and not dispersion radius would change the optimal placement for nozzles and the pipe.

Chapter 6

Future Work

While our model has provided some insight into the problem of the farmer and the field, there are a several of additional factors that we would like to investigate. The first being variations in field size. We would also like to explore the effects of non-uniform terrain on the irrigation system; farmland tends to be flat or gently sloping in one direction, but we only consider flat terrain. Also, it would be interesting to investigate the effect of weather on the sprinkler system, though this would certainly require emperical data. Prevailing wind would likely have a similar effect as terrain slope, but wind, humidity and temperature would likely all have effects on the amount of water required. It may also help to incorporate specifics of different nozzle designs into our model; perhaps some nozzle configurations would be better suited for different purposes.

We would like to some day write an evolutionary algorithm to determine the optimum positioning of the sprinklers on the field. Such an algorithm would provide automated generation of highly optimized solutions to the irrigation problem.

Bibliography

[1] Google. "Google Calculator". 2006. http://www.google.com/help/features.html#calculator.

[2] Evans, Robert G. "Center Pivot Irrigation". Biological Systems Engineering Department, 1999.

[3] Serway, Raymond and John Jewett. Physics for Scientists and Engineers. Belmont: Brooks and Cole, 2004.

[4] Anton, Howard. Calculus, A New Horizon. New York: John Wiley and Sons, Inc.

[5] Smajstral, A.G. and F.S. Zazueta. "Evaporation Loss During Sprinkler Irrigation". Institute of Food and Agricultural Sciences. University of Florida. 2003.

[6] Wikipedia. 2006. http://www.wikipedia.org.

[7] Rain Bird 2006-2007 catalog. http://www.rainbird.com.

Appendix A

Simulation in Python

A.1 irrigation_sim.py

Python simulation code.

```
#
#   irrigation_sim.py  -  irrigation simulator
#

# Standard Library
from math import *

# Additional Packages
from Numeric import *
from MLab import *
import pylab
import Image

# User-defined Packages
import irrigation_tools as tools

class Sprinkler:

    distance_from_root = 0.0 # distance from the root point.


    # position vector of the root point.
    x = 0.0
    y = 0.0
    phi = 0.0
```

```
    arc = [1]  # water a half circle
    # arc = [1,-1]
    # arc = [-1]

    # constants
    nozzle_diameter = 0.6 # cm
    percent_water = 0.9 # percentage of water retained

    # derived values
    nozzle_flow = 0.0
    nozzle_velocity = 0.0
    dispersion_radius = 0.0
    dispersion_area = 0.0

    def __init__(self, nozzle_flow, theta):
        self.nozzle_flow = nozzle_flow * tools.in_conv["liters_per_minute"]
        self.nozzle_velocity = tools.velocity(self.nozzle_flow,
         self.nozzle_diameter,
         in_units=["cubic_meters_per_second", "centimeters"])
        self.dispersion_radius = tools.radius(self.nozzle_velocity, theta)
        self.dispersion_area = tools.area(self.dispersion_radius)

    # water depth as a function of distance from the nozzle.
    @tools.units
    def dispersion_profile(self, distance_from_nozzle):
        return 3.0*(-distance_from_nozzle/self.dispersion_radius + \
         self.percent_water)*(self.nozzle_flow/self.dispersion_area)

    @tools.units
    def dispersion(self, dc):
        return self.dispersion_profile(dc)

    def move(self, x, y, phi):
        self.x = x
        self.y = y
        self.phi = phi

    def coords(self):
        return (self.x + self.distance_from_root * cos(self.phi),
         self.y + self.distance_from_root * sin(self.phi))

class SprinklerSystem:
```

```
    inflow = 150.0 # liters per minute
    x = 0.0 # coords of pipe end
    y = 0.0
    phi = pi/2 # angle of the pipe

    def __init__(self, sprinkler_positions, theta):
        self.sprinklers = []
        n = len(sprinkler_positions)
        for ii in range(n):
            s = Sprinkler(self.inflow/float(n), theta)
            s.distance_from_root = sprinkler_positions[ii]
            s.x = self.x
            s.y = self.y
            s.phi = self.phi
            self.sprinklers.append(s)

    def run_sprinklers(self):
        for sprinkler in self.sprinklers:
            yield sprinkler

    def move(self,x,y,phi):
        self.x = x
        self.y = y
        self.phi = phi
        for sprinkler in self.sprinklers:
            sprinkler.move(x,y,phi)



class Field:

    max_hourly_water = 0.75 # cm
    width = 80.0
    height = 30.0
    sectors = array([])

    def __init__(self, resolution, max_hourly_water):
        self.resolution = resolution
        self.max_hourly_water = max_hourly_water
        self.sectors = zeros((self.resolution*self.height,
         self.resolution*self.width), Float)
```

```
    def apply_water(self, sprinkler):
        # adjusting coordinates to array indexes
        x, y = sprinkler.coords()
        ix = self.resolution * x
        iy = self.resolution * y
        ir = self.resolution * sprinkler.dispersion_radius
        for dx in range(int(ix-ir), int(ix+ir)):
            for dy in range(int(iy-ir), int(iy+ir)):
                if 0 <= dx and dx <= self.sectors.shape[1]-1 \
                 and 0 <= dy and dy <= self.sectors.shape[0]-1:
                    dc = sqrt(float(dx-ix)**2 + float(dy-iy)**2)
                    if dc <= ir:
                        try:
                            self.sectors[int(dy)][int(dx)] =  \
                             self.sectors[int(dy)][int(dx)] + \
                             sprinkler.dispersion(dc/self.resolution,
                              out_units="centimeters_per_hour")
                        except IndexError:
                            raise str((dy,dx))

    def clear(self):
        self.sectors = zeros(self.sectors.shape, Float)

    def render_field(self, sprinkler_coords, path = "irrigation.png"):
        print "Drawing Image..."
        sectors = self.sectors * 100.0
        for x,y in sprinkler_coords:
            x = int(x * self.resolution)
            y = int(y * self.resolution)
            for ix in range(x-10, x+10):
                for iy in range(y-10, y+10):
                    if 0 <= iy and iy <= sectors.shape[0]-1 and \
                     0 <= ix and ix <= sectors.shape[1]-1:
                        try:
                            sectors[iy][ix] = 0.0
                        except:
                            raise str(((iy,ix), sectors.shape))
        im = Image.new("RGB",
         (int(self.width*self.resolution), int(self.height*self.resolution)))
        img_array = zeros((self.width*self.resolution**2*self.height, 3), 'l')
        img_array[:,0] = ravel(sectors).astype("l")
```

```
        img_array[:,1] = ravel(sectors).astype("1")
        img_array[:,2] = ravel(sectors).astype("1")
        im.putdata(tools.tupleize(img_array.tolist()))
        im.save(path)


class Simulation:

    resolution = 10.0 #  number of array elements per square meter
    max_hourly_water = 0.750 # cm

    def __init__(self, sprinkler_positions, theta):
        self.f = Field(self.resolution, self.max_hourly_water)
        self.hourly = Field(self.resolution, self.max_hourly_water)
        self.s = SprinklerSystem(sprinkler_positions, theta)

    def run_simulation(self):
#         positions = [(0, 5.0, pi/2),
#                      ( 0, 5.0, pi/2),
#                      ( 0, 5.0, pi/2),
#                      ( 0, 5.0, pi/2),
#                      (20, 5.0, pi/2),
#                      (20, 5.0, pi/2),
#                      (20, 5.0, pi/2),
#                      (20, 5.0, pi/2),
#                      (40, 5.0, pi/2),
#                      (40, 5.0, pi/2),
#                      (40, 5.0, pi/2),
#                      (40, 5.0, pi/2),
#                      (60, 5.0, pi/2),
#                      (60, 5.0, pi/2),
#                      (60, 5.0, pi/2),
#                      (60, 5.0, pi/2),
#                      (80, 5.0, pi/2),
#                      (80, 5.0, pi/2),
#                      (80, 5.0, pi/2),
#                      (80, 5.0, pi/2)]

        positions = [( 1.0,  5.0, 0.0),
                     ( 1.0,  5.0, 0.0),
                     ( 5.0,  5.0, 0.0),
                     (30.0,  5.0, 0.0),
```

```
                    (30.0,  5.0, 0.0),
                    (55.0,  5.0, 0.0),
                    (59.0,  5.0, 0.0),
                    (59.0,  5.0, 0.0),
                    (59.0, 25.0, 0.0),
                    (59.0, 25.0, 0.0),
                    (55.0, 25.0, 0.0),
                    (30.0, 25.0, 0.0),
                    (30.0, 25.0, 0.0),
                    ( 5.0, 25.0, 0.0),
                    ( 1.0, 25.0, 0.0),
                    ( 1.0, 25.0, 0.0)]

        print "Starting simulation."
        for hours in range(len(positions)):
            print "Hour: %i, position: %s" % (hours, str(positions[hours]))
            self.hourly.clear()
            self.s.move(*positions[hours])
            print "  Running sprinklers."
            sprinkler_coords = []
            for sprink in self.s.run_sprinklers():
                print "   ", sprink.coords()
                sprinkler_coords.append(sprink.coords())
                self.hourly.apply_water(sprink)

            print "Maximum hourly accumulation on a sector: %f" % max(max(self.hourly
            if max(self.hourly.sectors) > self.max_hourly_water:
                print "Too much water per hour on some sectors..."
            self.f.sectors = self.f.sectors + self.hourly.sectors
            self.f.render_field(sprinkler_coords, "irrigation_%i.png" % hours)
        print "Most accumulation on a sector: %f" % max(max(self.f.sectors))
        print "Least accumulation on a sector: %f" % min(min(self.f.sectors))


if __name__ == "__main__":
    s = Simulation([0, 4, 16, 20], pi/4)
    s.run_simulation()
```

A.2 irrigation_tools.py

Tools for the python simulation.

```
G = 9.81 # m/s^2

import types
from math import *
from matplotlib.patches import Circle, Rectangle, Polygon
from matplotlib.transforms import Value
from matplotlib.backends.backend_agg import RendererAgg


# all conversions to meters and seconds
in_conv = {"liters_per_minute": 1.66666666666e-5, # cubic meters per second
           "cubic_meters_per_second": 1.0,
           "centimeters": 1.0e-2, # meters
           "square_meters": 1.0, # leave alone
           "radians": 1.0,
}

# all conversions from meters and seconds
out_conv = {"meters_per_second": 1.0,
            "centimeters_per_second": 1.0e2, # from meters per second
            "centimeters_per_hour": 1.0e2 * 60.0 * 60.0 # from meters per second
}

def units(f):
    def wrapper(*args, **kwds):
        if kwds.has_key("in_units"):
            result = f(*[in_conv[c] * a for c, a in zip(kwds["in_units"],args)])
        else:
            result = f(*args)
        if kwds.has_key("out_units"):
            result = out_conv[kwds["out_units"]] * result
        return result
    return wrapper

@units
def velocity(flowrate, diameter):
    """Velocity as a function of flowrate and tube diameter."""
    # flowrate is in liters per minute (10^3 cm^3 / minute)
    # diameter in centimeters
    # returns meters per second
    return (4 * flowrate)/(pi * (diameter)**2)
```

```
@units
def radius(velocity, theta):
    """Radius as a function of velocity and angle."""
    # velocity in meters per second
    # theta (angle from horizontal, radians)
    # returns meters
    v_u = velocity * cos(theta)
    tm = 2 * v_u / G
    return velocity * sin(theta) * tm

@units
def area(radius):
    return pi * radius**2

@units
def depth(sprinkler_flow, area):
    return sprinkler_flow/area

@units
def find_m(n, radius):
    ds = 20.0/n
    m = 2 * floor(radius/ds) + 1
    if m >= n:
        return n
    else:
        return m

def tupleize(alist):
    return list([tuple(item) for item in alist])

def print_params(n, theta):
    # generates a parameter table
    f_in = 150.0 # L/min
    d_in = 0.6 # cm
    print " # _flow_ n_vel radius ___area___ _depth_ _day_ _4days_  m _m*d_"
    for ii in range(1, n+1):
        F = f_in/ii
        V = velocity(F, d_in, in_units=["liters_per_minute", "centimeters"])
        R = radius(V, theta)
        A = area(R)
        D = depth(F, A,
         in_units=["liters_per_minute", "square_meters"],
```

```
 out_units="centimeters_per_hour")
M = find_m(ii, R)
print "%2i %6.2f %5.2f %6.2f %10.2f %5.5f %5.2f %7.3f %2i %4.3f" \
 % (ii, F, V, R, A, D, 24*D, 4*24*D, M, M*D)
```

Appendix B

Simulation in C

B.1 irrigation.c

C simulation code.

```
/*
 * Irrigation.c - irrigation simulator
 */

#define _GNU_SOURCE

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <Ilib.h>

const double pi = 3.14159265358979323846433832975;

struct lines {
    char *line;
    struct lines *next;
};
```

```
struct positions {
    int duration;
    int vertical;
    float x;
    float y;
    struct positions *next;
} positions;

uint32_t field[1600][600];

double theta = 3.14159265358979323846433832795/4;

int nozzles;

float nozzlepositions[30];

void parserules(char *stuff);
struct lines *parselines(char *stuff);
void simulate();
void spraywater(float x, float y, int duration);
void writeimage();

int main(int argc, char **argv)
{
    int i, j;

    if (argc < 2) {
        printf("Please enter a rules file\n");
        exit(-1);
    }

    parserules(argv[1]);

    for (i = 0; i < 1600; i++)
        for (j = 0; j < 600; j++)
            field[i][j] = 0;

    simulate();
    writeimage();

    return 0;
```

```
}

void writeimage()
{
    long long avg = 0;
    double variance = 0;
    int i, j, l = 255, m = 0, sum = 960000;
    IColor c;
    IGC g = ICreateGC();
    //IImage out = ICreateImage(1600, 600, IOPTION_GREYSCALE);
    IImage out = ICreateImage(1600, 600, 0);



    for (i = 0; i < 1600; i++)
        for (j = 0; j < 600; j++) {
            if (field[i][j] > 255) {
                field[i][j] = 255;
                sum--;
            } else {
                avg += field[i][j];
                if (field[i][j] > m)
                    m = field[i][j];
            }
            if (field[i][j] < l)
                l = field[i][j];
            c = IAllocColor(255 - field[i][j], 255 - field[i][j], 255 - field[i][j]);
            //c = IAllocColor(0, 0, 255 - field[i][j]);
            ISetForeground(g, c);
            IDrawPoint(out, g, i, j);
        }
    printf("Greatest acculumation: %i\nSmallest accumulation: %i\n", m, l);
    float rat = (float) m / l;
    double a2 = (double) avg / sum;
    printf("Ratio: %f, Average: %f\n", rat, a2);

    for (i = 0; i < 1600; i++)
        for (j = 0; j < 600; j++)
            if (field[i][j] < 255)
                variance += (a2 - field[i][j]) * (a2 - field[i][j]);

    double v2 = variance / sum;
```

```
    printf("Standard deviation: %f, percent %f%%\n", sqrt(v2), ((sqrt(v2) / a2) * 100)

    FILE *fp = fopen("irrigation.png", "w");
    IWriteImageFile(fp, out, IFORMAT_PNG, 0);
}

void simulate()
{
    int i, n = 0;
    struct positions *p;

    p = positions.next;
    while (p) {
        for (i = 0; i < nozzles; i++) {
            if (p->vertical)
                spraywater(p->x, p->y + nozzlepositions[i], p->duration);
            else
                spraywater(p->x + nozzlepositions[i], p->y, p->duration);
        }

        p = p->next;
    }
}

void parserules(char *stuff)
{
    int i;
    struct stat statbuf;
    char *file, *p;
    struct lines *lines;
    struct positions *q = &positions;

    i = open(stuff, O_RDONLY);

    if (i < 0) {
        printf("Can't open rules file\n");
        exit(-1);
    }

    fstat(i, &statbuf);

    file = malloc(statbuf.st_size + 1);
```

```
    read(i, file, statbuf.st_size);
    close(i);
    i = 0;

    lines = parselines(file);

    while (lines) {
        if (strncmp(lines->line, "nozzle", 6) == 0) {
            lines->line += 6;
            for (nozzles = 0; ; nozzles++) {
                nozzlepositions[nozzles] = strtof(lines->line, &p);
                if (lines->line == p)
                    break;
                lines->line = p;
            }
        } else if (strncmp(lines->line, "position", 8) == 0) {
            lines->line += 8;
            q = q->next = malloc(sizeof(struct positions));

            q->duration = strtol(lines->line, &p, 10);
            lines->line = p;
            while (isblank(*lines->line))
                lines->line++;

            q->vertical = 0;
            if (*lines->line == 'v' || *lines->line == 'V')
                q->vertical = 1;
            lines->line++;

            q->x = strtof(lines->line, &p);
            q->y = strtof(p, NULL);
        }
        i++;
        lines = lines->next;
    }
    q->next = NULL;
}

struct lines *parselines(char *stuff)
{
    struct lines l, *p = &l;
```

```
    while (1) {
        p = p->next = malloc(sizeof(struct lines));
        p->line = stuff;

        stuff = strchr(stuff, '\n');
        if (!stuff)
            break;
        *stuff = '\0';
        ++stuff;
    }
    p->next = NULL;
    return l.next;
}

void spraywater(float x, float y, int duration)
{
    // decimeters, seconds
    int i, j, ix, iy, a, b;

    ix = x * 20;
    iy = y * 20;

    double flow = 2.5 * 1000 / nozzles;

    double velocity = (4 * flow)/(pi * .36 * 100);

    double vu = velocity * cos(theta);
    double tm = 2 * vu / 9.81;
    double radius = velocity * sin(theta) * tm;
    radius *= 20;
    radius *= .9;
    double r2 = radius * radius;

    for (i = 0; i < 1600; i++)
        for (j = 0; j < 600; j++) {
             a = i - ix;
             b = j - iy;
             if (a * a + b * b < r2) {
                 if (a * a + b * b < 120)
                     field[i][j] = 256;
                 field[i][j] += (((radius - sqrt(a * a + b * b))) ) * flow *40/ r2;
                 //field[i][j] += (((sqrt(a * a + b * b))) ) * flow *10/ r2;
```

```
            }
        }
}
```

B.2 irrigationrules

Configuration file for the C simulation.

```
nozzle 0 5 15 20 # 5.4% SD
position 120 h 3.5 0
position 120 h 56.5 0
position 120 h 3.5 30
position 120 h 56.5 30

nozzle 0 5 15 20 # 7.9% SD
position 120 v 0 5
position 120 v 14 5
position 120 v 40 5
position 120 v 66 5
position 120 v 80 5

nozzle 0 5 10 15 20 # 5.9% SD
position 120 h -2 1
position 120 h -2 29
position 120 h 30 1
position 120 h 30 29
position 120 h 62 29
position 120 h 62 1

nozzle 0 4 8 12 16 20 # 9.2% SD
position 120 h 0 4
position 120 h 0 26
position 120 h 30 4
position 120 h 30 26
position 120 h 60 4
position 120 h 60 26

nozzle 0 2.9 5.8 8.7 11.6 14.5 17.4 20 # 5.9% SD
position 120 h -3 3
position 120 h -3 15
position 120 h -3 27
position 120 h 19 3
```

```
 position 120 h 19 15
 position 120 h 19 27
 position 120 h 41 3
 position 120 h 41 15
 position 120 h 41 27
 position 120 h 63 3
 position 120 h 63 15
 position 120 h 63 27

nozzle 0 2 4 6 8 10 12 14 16 18 20 # 4.6% SD
position 120 h -3 1.5
position 120 h -3 8.3
position 120 h -3 15
position 120 h -3 21.7
position 120 h -3 28.5
position 120 h 19 1.5
position 120 h 19 8.3
position 120 h 19 15
position 120 h 19 21.7
position 120 h 19 28.5
position 120 h 41 1.5
position 120 h 41 8.3
position 120 h 41 15
position 120 h 41 21.7
position 120 h 41 28.5
position 120 h 63 1.5
position 120 h 63 8.3
position 120 h 63 15
position 120 h 63 21.7
position 120 h 63 28.5
```