Computing Curvature Using Amorphous Computing

Omari Carter-Thorpe

under the direction of
Radhika Nagpal
Massachusettes Institute Technology

Research Science Institute
August 5, 1998

Abstract

Amorphous Computing is the study of designing ultra-scale computing environments of smart sensors and actuators, where the individual processing elements are identical, randomly and densely distributed in a material, and communicate wirelessly within a small local neighborhood. Examples of such environments are bridge surfaces that can detect load or cracks, airplane wings that modify to reduce shear, etc. The objective of this project is to determine an effective method for individual amorphous computing processors to determine the curvature of the structure on which they are embedded by exchanging local information. This paper presents a solution which involves computing the number of elements within a local area and comparing it to the expected number of processors in a flat surface. The formulas and computer program are presented along with results from simulating the program on an amorphous computer embedded on a sphere. The paper discusses the sources of error in this technique and proposes other solutions.

1 Introduction

The idea of Gunk is a relatively new idea. The term Gunk refers to a substance that is filled with many cheaply produced micro-processors, that communicate with each other and attempt to accomplish a given task. An good example of Gunk would be paint and a bridge. Paint would be filled with these processors making Gunk Paint, that would then be spread across the bridge. The processors would then use different input sensors located on the bridge to monitor stress and detect weaknesses in the bridge. If a weakness were to be detected the processors would then either attempt to correct it, or notify someone capable of correcting the problem that arises.

There are many possible inputs, and many things that can be detected about the surface or structure on which the Gunk rests. Examples of these would be texture, maliability, altitude, and curvature. Of these different problems this paper deals specifically with detecting curvature. Detecting curvate of complex structures can be difficult, but any complex curved structure can be broken down into a compilation of simpler curved structers. Therefore the ability to detect the curvature of a simple surface like a sphere or plane without requiring coverage of the entire surface in gunk would be enough to measure the curvature of just about any surface.

This paper will discus the method used (should I put 1 more paragraph of intro? Maybe why curvature of a surface is important?).

2 Design

Being able to find the curvature of a sphere is an excellent test to of the abilities of Gunk, and also illustrate it's limitations. The particular method that was used to find curvature used surface area. The curvature of a sphere was determined by the the size of the radius reletave to the size and range of the processors in the Gunk. So the task for the system was to collectively return one value, as accurate as possible, of the radius of the sphere in the units of a. A is the distance (although it may vary from one Gunk system to another), that is equal to the radius of the sphere that represents the complete range of communication of a particular processor.

The formula for the surface area of the sphere is (convert to latex formula Surface Area = 2PI Rsqrd2 [1 - cos (a/r)]). Where as was said previously a is the maximum communication radius. Because the the processors already know a and will determine the surface area for themselves, this formula must be solved for r; the radius of the sphere. Solving for r complicates the situation a little because r is both a coefficient of the portion of the formula in brackets and is an input to the function cosine. A few different methods were determined to solve this problem including method 1, method 2, and using an approximation of cosine to achieve the value of r.

(talk about methods 1 and 2?)

The third method of determining r, which is the one that was finially decided on is the method of using an approximation for cosine. While this aproximation being the method of choice (Taylor series?), it wasn't alway the definitive choice, nor was it the primary choice. After looking at the amount of error of different accuracy, it was decided that formula that would be easiest to use, while still giving acuratate values of cosine was 1- t2/t! + t4/4!. After solving the formula with the approximation of cosine for r, the final formula is found that will be used to calculate the radius of a sphere from only knowing curvature: r = 12sasd + 323nssonf. After finding the formula that would be used calculate the radius of a sphere using as inputs the surface area of a portion of the sphere, and the size of the communication radius, the next problem was the method that would be used by the processors to obtain these inputs.

Part of this new group of problems was solved very quickly, because having a processor determing the second input (knowing the size of the communication radius) can be very easily solved. As a result of being a uniform static value, it can easily be given to all processors in the Gunk. This left the problem of determining the first input, the surface area of a portion of the sphere as the next challenge.

The final solution to determining surface area, turned out to be density. If processors are distributed randomly and uniformly on the surface of the sphere (or any object or structure for the matter) the average number of processors per a given area (Ex. Communication Area) should remain relatively regardless of the particular location on the surface that is chosen (Info on variance of avg D and actual d?). From here knowing that Surface Area * density = NumProcessors, surface area (hopefully with little error) by dividing the number of processors in a specific area by the average density of the area. So theoretically a group of processors could sum themselves, divide this number by the average density to calculate their surface area, then from there plug it into the approximated radius formula to find the radius of the surface on which they sit. Assuming that all goes well, the processors then also have the advantage of being able to calculate the curvature of only a small section, that is curved. for example if there were a bump or hill with constant curvature (a section of a sphere) on a perfectly smooth plane, and a group of processors were placed on this bump they would be able to calculate the curvature. If a formula were found that would require the entire surface of the imaginary sphere which has a section that is the bump, it would be impossible. As a result of this, entire structures can be mapped out, and divided into regions of different curvatures as well as eventually be mapped according to different attributes.

Now that the method has be determined to arrive upon the curvature of a surface (in this case a sphere) from knowledge of the number of processors, the next logical step is to determine the most effective method of counting the number of processors in total, and later, more specifically the number of processors in a particular area. So first what was done was to see of the total number of processors placed on a sphere could be calculated, then if only the number of processors within a certain distance of a specified location could be calculated.

The method chosen to me easiest and most efficient was the creation of a tree. As has been stated Gunk is essentially nothing more than a multitude of completely independent processors in a substance attempting to accomplish a task. They are given a means of communication without a method. There is no order, no structure, no differentiation, no selection nor specialization. This allows for a large amount of flexibility, and a tolerance for unexpected problems, but it doesn't give any direct line of communication or organization. To solve this problem and to eventually be able to total the number of processors, some method of communication but be implemented. The tree consists simply a central processor that is the top node of the tree. The processor (know as the source) knows all of its neighbors, which are placed onto the tree as it's children. Then from their each one of those processors then locates all of its neighbors that **aren't** already in the tree and adds them as their respective children. These newly found processors are generation 3 in the tree, their parents (the children of the source) are generation two, and the original source processor is generation 1. This process continues until the entire group of processors has its own place on the tree. One the tree has been constructed, communication between processors is simple. When a action needs to be done, the source processor sends the message to it's children the children then send the message to their children and so forth. Then when the data needs to be returned or information has to be sent that was read from a sensor or whatever the case may be, it is just sent by each processor to its parent until the message is received by the source.

There are many advantages of using this method of communication, as opposed to another. One of these is that it is very easy to create the tree of communication. Another advantage is that once the tree has been created it is also a simple and does not require processor specific code to pass on a message, except for the processor with no parent (the source), and those with no children (leaves). Finially the last reason why the tree method was chosen for as the method of communication is because it is easiest to then use to find the number of processors in a particular area. Figure BLah shows how processors created in this tree (colored according to their generation) create concentric (is that the word) circles with radii roughly equal to integer multiples of the radius of the innermost circle. This innermost circle has a radius equal to that of the communication radius, therefore the distance between each circle is 1 communication radius (or 1 a). From this, all that is required to measure the number of processors in a particular area is to send out a message to sum all of the processors within a certain distance (or generation) of the source processor.

(Also write problems that arise as a result of small densitys on tree distance accuracies)

3 Implementation

This section discusses implementation of the aforementioned ideas, as well as discusses the slight differences between the implementation and the original ideas as a result of attempting to map them to the limitations of scheme and the High-Level Simulator which is used to simulated the interactions between the processors in Gunk. blah blah blah **

All of the Amorphous Computing simulations were run using a simulator called The High-Level Simulator (HLSIM). This simulator allows the user to write the sources code is scheme, using special HLSIM constructs and functions to give the processors their instruction set. HLSIM then proceeds to create and area that will be painted with gunk, places the processors randomly on this area, then begins to simulate the actions of the specified number of processors (usually in the thousands or tens of thousands) running in parallel. HLSIM controls all message broadcasting, and receiving by processors and places the messages in a buffer to read at anytime my the processor. HLSIM also implements timeouts, so for example if a processors were to not want to do an action immeaditly, but were to want to wait five seconds, it would be able to using a timeout. The processor would continue doing its normal action, and then after five seconds elapses HLSIM would notify the processor and run the designated code.

The first problem that must be tackled when attempting to measure the curvature of a sphere is to have the processors placed randomly on a sphere. The major problem was being sure that the density of the processors at the sphere's equator was equal to the density of the processors at the caps. The following things were determined where x and y are width and depth in the HLSIM/Gunk environment, and z is height. The easiest method determined to distribute the processors evenly would be using the spherical coordinates of axial radius, bearing, and latitude. From there, convert the coordinates into the Cartesian coordinates of HLSIM. The distance from the axial radius has already been determined because it is simply the radius of the sphere, thus leaving only the bearing θ and the latitude ρ to be found. To accomplish this a random number between 0 and 2π was chosen to represent the bearing. From here only ρ needed to be chosen. If ρ where to be chosen using the same method, the distributation of the sphere would be non-uniform. Ignoring bearing if you just look at the total surface area for each particular value of ρ it can easily be seen that the circle that is formed from taking a horizontal cross section at position ρ the the surface area decreases as ρ nears 0 and 2 π, and increases as it nears π and $3/4\pi$. Because of this the probabilities of chosing ρ must be weighted so as to create and even distribution. To accomplish this a random number between -1.0 and 1.0 was chosen. Then the arcsine of this number was found and used as our latitude ρ. So the formulas used to determine the spherical coordinates are:

$$Axial\ Radius = Radius\ of\ Sphere \tag{1}$$

$$Bearing\ (\theta) = Random(bet.\ 0\ 2\pi) \tag{2}$$

$$Latitude\ (\rho) = \arcsin\ Random(bet.\ -1.0\ 1.0) \tag{3}$$

After finding these coordinates, the point must be maped to the Cartesian coordinates that HLSIM uses. Converting can be accomplished as follows:

$$Vector_X = \cos\theta * \cos\rho * AxialRadius \tag{4}$$

$$Vector_Y = \sin\theta * \cos\rho * AxialRadius \tag{5}$$

$$Vector_Z = \sin\rho * AxialRadius \tag{6}$$

A final topic that had to be addressed in the implementation of finding curvature is communication details. Although the general idea of communication in the tree structure had be devised, this still wasn't specific enough for HLSIM. Therefore, more detail must have been added in order to have the process work.

When communicating, processors relay messages back and forth from parent to child and vice versa. Without these messages all functionality of the Gunk system would be lost, and a failure in the relaying of these messages or incorrect delivery could lead to disastrous results. Therefore a reliable efficient method of communication must be devised in order for communication to exist. Some of the larger problems with this are, having the messages interpreted by only the particular processors for which they were directed and allowing the processors to be capable of identifying whether the message was sent to it from it's parent or child. The protocol of communication that was eventually adopted was very simple. First it was determined under what circumstances a particular processor would listen to a message, and when a message that it heard would be ignored. Following this, more information that a processor would need, would be the sender of the message (which would give the processors knowledge or whether the message was descending or ascending the tree, and who in particular was sending the message), and finially if the message is something that they should have heard, what is the sender's purpose. These three blocks of data were sent preceding every message send by a processor. Three variables were created named Target, Action and Sender, which when received preceding communication data, informed the receiver of the three important message blocks.

4 Results and Data

5 Discussion

The majority of the research and therefore the majority of this paper dealt with the creation and design of new methods of computing curvature, there still remain a few points for discussion.

Graph one is a graph of the change in area as a result of the change in Hops on a sphere of radius *this* with a hop length of *that*. A Hop is considered one unit of length equal to the communication radius of the processors. When in a communication tree, the distinctive circles that are created correspond to generation (figure: NotInYet) is one hop from the previous circle. This graph shows that if you go only two hops from the source processor there is very little difference in area, while if you go about twelve or 14 hops there is a large difference, and it would be relatively easy to tell the radius just from knowing the area (assuming there is no error in the area). From this graph, and from knowing that there will be variance in the density of the processors due to the random placement, a Hop Count of 14 was chosen to measure the majority of the data from.

The following graph is a little more complex. I keeps the radius of the sphere constant, but it lists a number of curves representing the different values for area that would be derived using different methods for each different number of hops from the source processor. The uppermost line represents the increase in area if the surface were completely flat, and the line just below it represents the computed area using the formula. This was placed to show how accurate (or inaccurate) the measured and computed areas were. This curve was computed using the approximation formula just as were the measured area curves. Using the approximation adds almost no error in the computed radius and is safe to use without fear of further complicating the data. Before the final four lines can be discussed, first the effects of density and the concept Kleinrock hops must be addressed.

As has been said that processors in the gunk are placed randomly on a sphere. From this a average density is computed, and later used. When the communication tree is constructed, the source processor (at every processor from there on) sends out a message to all of the processors in its communication radius. From here each of these processors then sends out a message to everyone in their communication radius. The reason for this is to measure distance from the source processor. If the communication radius (a) is a certain length, and a particular processor is generation four (which means it is three generations from the generation 1 source processor), then to calculate the distance between the source processor and this processor simple multiply a by three.

While this seems to be a perfectly working strategy, it is flawed. Lets say on average there were only three other processors in the communication area of a processor. The odds of these processors being on the very edge of the communication area (ie. the distance between the two processors is equal to the communication radius), is very small. So lets say that a particular processor Prs_X is only $\frac{1}{2}$ the communication radius away from the source processor (Prs_SRC). Now lets say that Prs_Xthen continues building the tree, and sends out a message. As was said before there is an average of only three other processors being in the neighborhood of the original processor, the odds of Prs_Y (a new processor that hears the message) being the full communication radius from Prs_X is also very small. So lets say again that it is 1/2 the communication radius from Prs_X. Now to keep things simple lets say that all of the processors are placed colinnearly. So now when Prs_Y attempts to find its distance from Prs_SRC it knows that it is two generations away, and it knows the communication radius, therefore according to the above formula: $Prs_Y\ Distance = 2 * Communication\ Radius$. But if we actually look at the distance between Prs_Y and Prs_SRC we can easily see that it is actually true that: $Prs_Y\ Distance = \frac{1}{2} * 2 * Communication\ Radius$ where the original formula is multiplied by $\frac{1}{2}$ because that is the actual average distance between processors of one generation and the next.

A man by the name of Mr. *Noname* Kleinrock wrote about this (Reference). Kleinrock researched the average distance between two different processors depending on the number of neighbours they had. He then created a formula that is used to compute the average distance if given number of neighbours. In the simulations that were run on the sphere, two different values for the number of neighbours were used on of 15 neighbours, and one of

30 neighbours. The two Kleinrock values were calculated for these neighborhood sizes and used. For the neigborhood size of 15, the value was *.79*, and for the 30 it was *.84*. Which means if on average you have 15 neighbours and you measure distances you will only travel about 79size, and at 30 neigbours you will only travel 84neighborhood size of 15, and one for 30. Each graph showes that expected area still using the cosine aproximation formula of before, but now also including the kleinrock corrections. The third curve is the klienrock area curve for a neighbourhood size of 15, and directly below that is the actual measured area curve for a neighboorhood size of 15. Curves five and six follow the same way with five being the Klienrock curve, and six being the measured area curve. As can be seen there is very little noticable difference between what would be expected according to the Kleinrock corrections and what was actually found, meaning that using the Klienrock corrections is a very effective method for calculating area from knowledge of the number of processors.

From what has been gathered so far the processors are capable of determining the surface area of the surface which they cover. The next part of this paper discuses the graphs related to converting this information into a radius, and finially discusses errors derived from calculation the radius.

communication Data... Kleinrock... Density...

6 Conclusion

Collisions and time differences, holes, regularizing,

7 Acknowledgments

References

[1] Stamey, Skye E.: Everything's Illegal in Virginia! *Norfolk Daily News*, 12 July 1992: 11(C).

[2] Szczepański, Amy F.: *100 Meals Using Ingredients That Your Mother Thinks Are Weird.* San Diego: Bench Press, 1997.

[3] Szczepański, Michael T.: Strange Things that I have Catapulted into our Neighbors' Yards. *Journal of Home Use of Medieval Siege Weapons.* **271** (10 December 1991): 19-76.

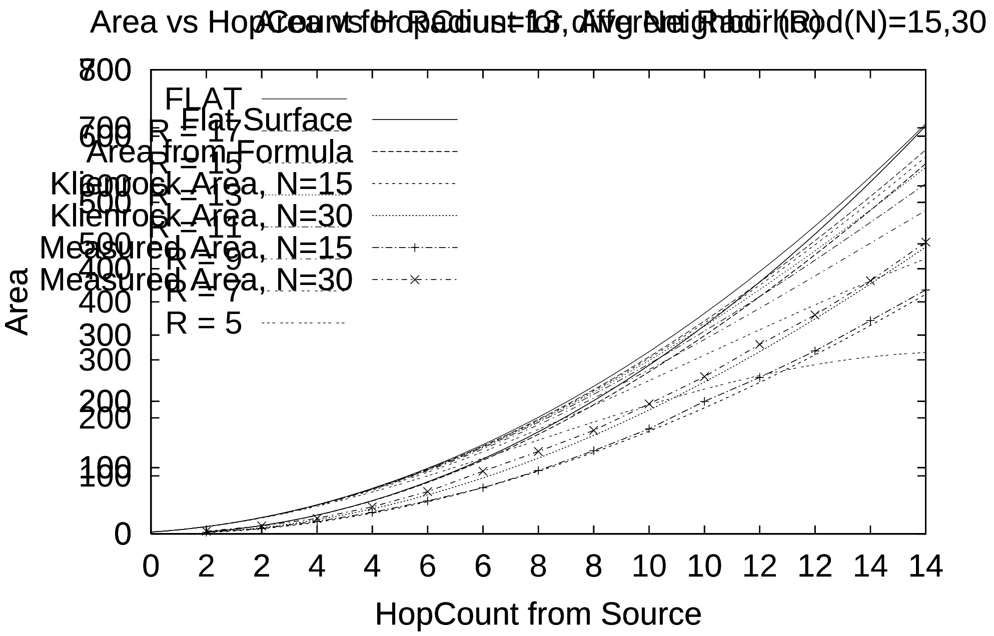


Figure 1: graph1

A Title of Appendix

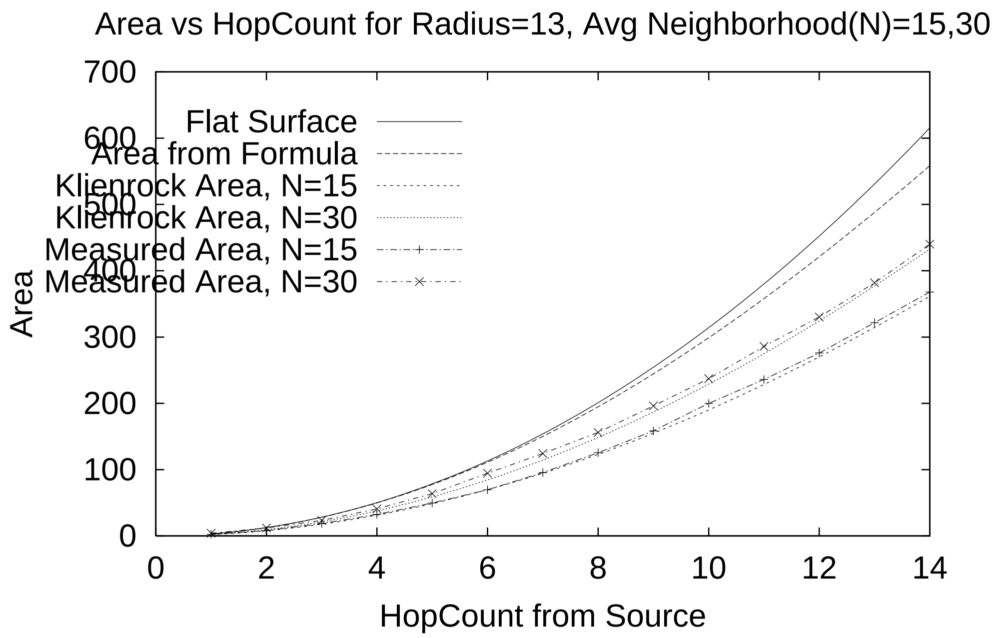


Figure 2: graph2

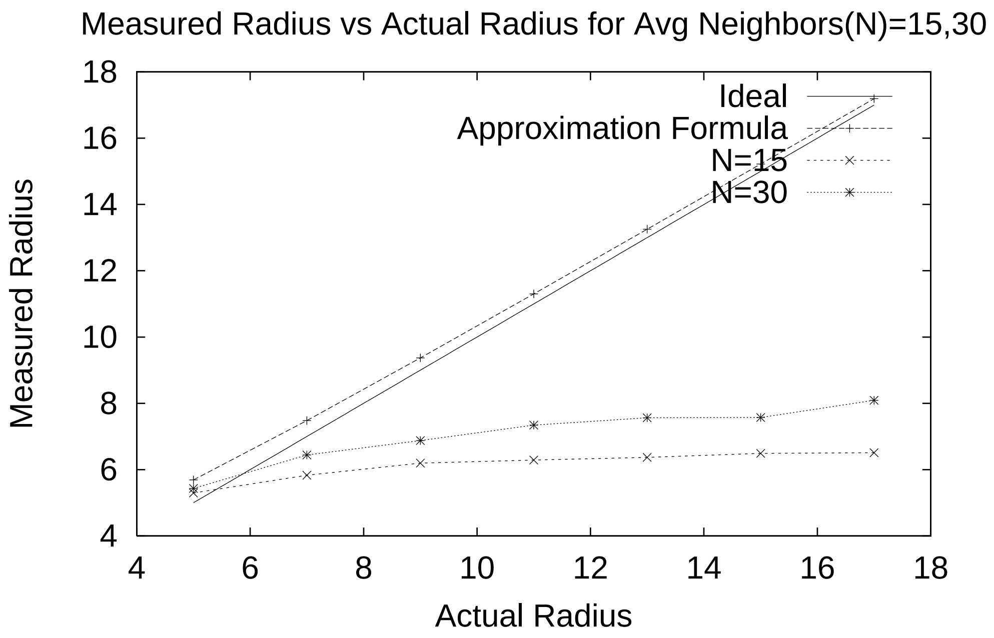


Figure 3: graph3

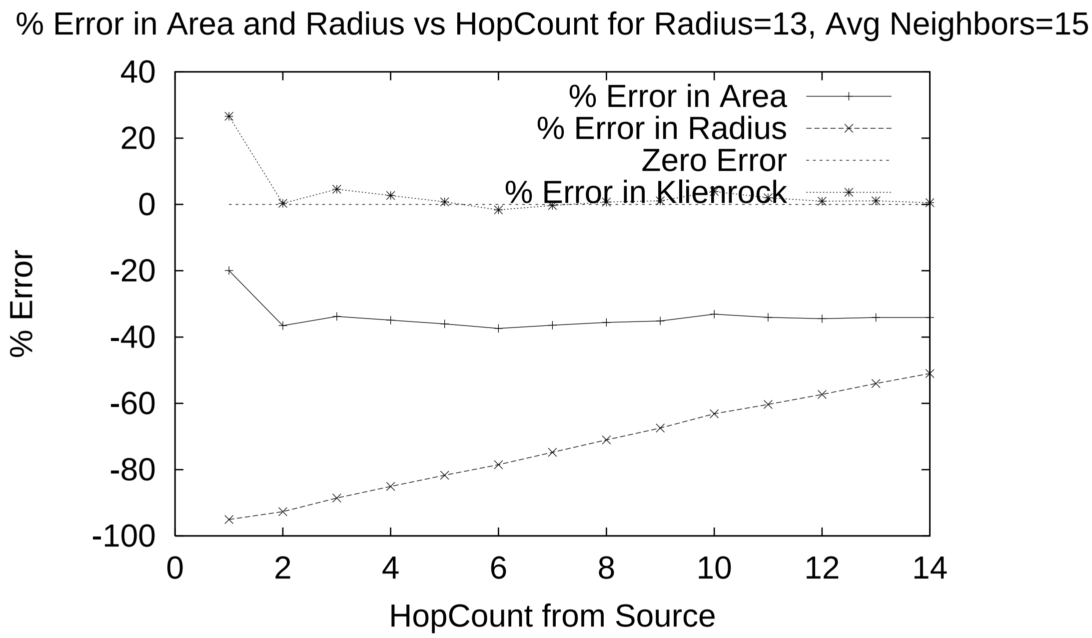
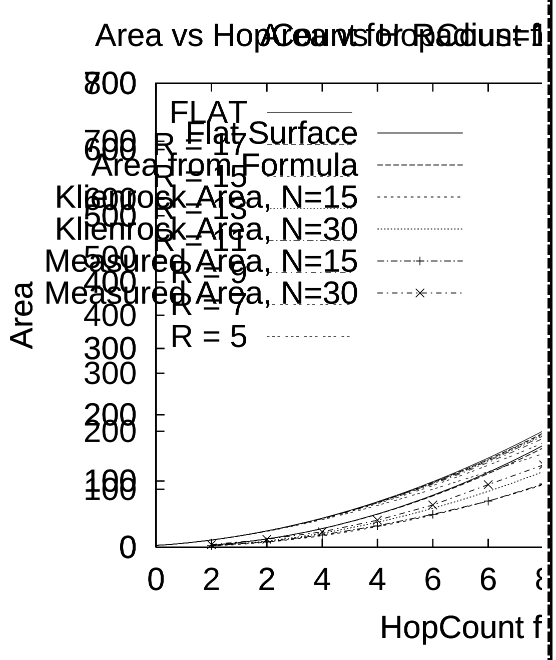


Figure 4: graph4



Fig

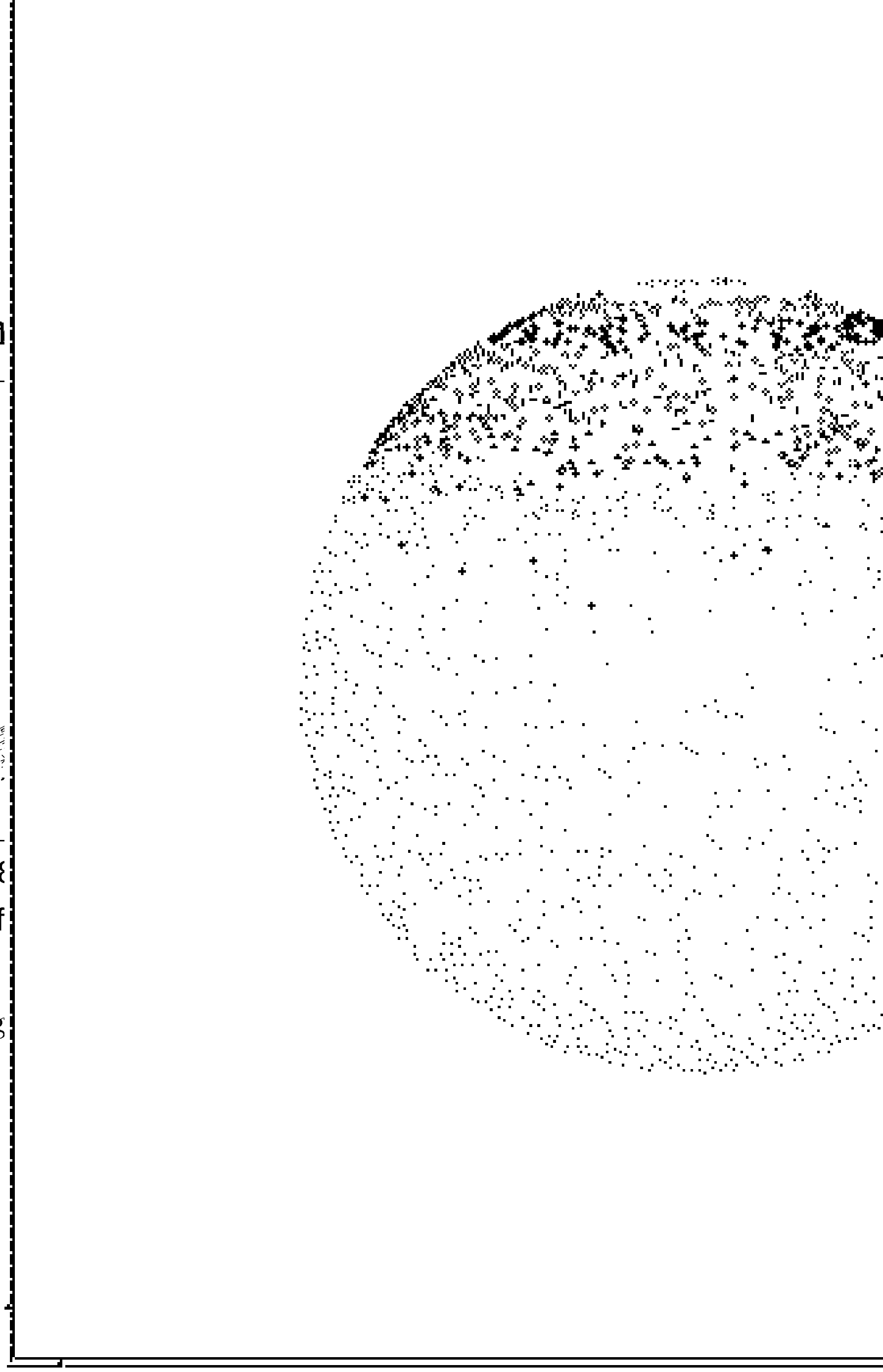


Figure 6: Picture