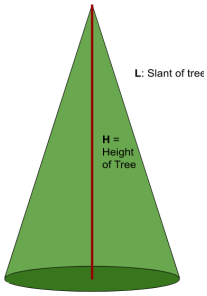
MSCS Mess

Department of Mathematics, Statistics, and Computer Science
St. Olaf College, Northfield, MN 55057
December 16, 2022 | Volume 51, No. 11

Happy Holidays from the MSCS Mess!

We here at the MSCS Department wish you luck with your finals and a restful winter break!



Christmas Trees

So I know I said a restful winter break, but have you ever had an intrusive puzzle that just needed to be solved? Well this year I'm trying to dazzle the infamously judgemental Santa Claus with the *perfect* Christmas tree. But how should I go about this? Math of course!

Really this is just an exercise in describing a problem and potential avenues to solve it, so don't expect super complex math or a perfect proof!

To start, I define a Christmas tree to be a Cone with four attributes: LEDs, garland, branches and ornaments. So I will begin by describing how to properly distribute these items on our tree.

Let us imagine we have our tree:

Beautiful I know. Now how can we describe wrapping a cord around our Christmas tree? How can we describe the distance between arcs, or loops, on our tree, and the length of cord needed?

Here's an idea, notice how any two points on the edge the cone, if you give them equal horizontal distance, will have equal slant distance! Thus, we only really have to worry about a two dimensional way of explaining the distance between each arc of our cord. I found the easiest way of drawing a spiral is in polar coordinates.

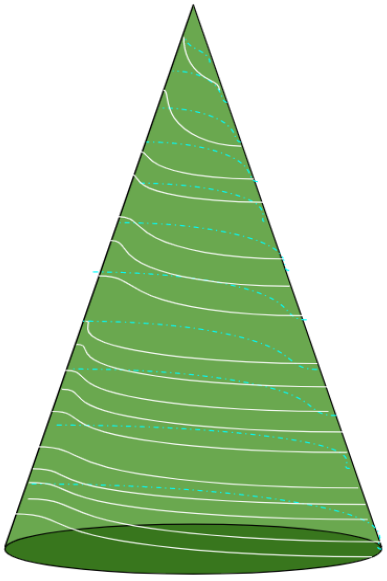
Polar coordinates, if we remember from Pre-Calculus, coordinate system (r, θ), where r is the distance from the origin, and θ is the angle from the positive x-axis. We can convert between π and degrees by using the ratio $\frac{180}{\pi}$.

Alright, back to spirals. Describing spirals in polar coordinates is super easy, basically we just us the formula $r = a + b * \theta$, where a and

b are real numbers. Describing certain types of spirals involves changing the equation a bit, but for simplicity's sake, this will do.

Now how should we distribute LEDs and garland on our tree? I think a perfect tree has an even slant length between arcs of the LED cord, while the garland has uneven slant lengths between its arcs. This is because the garland serves to draw the eye away from the LED cord within the tree.

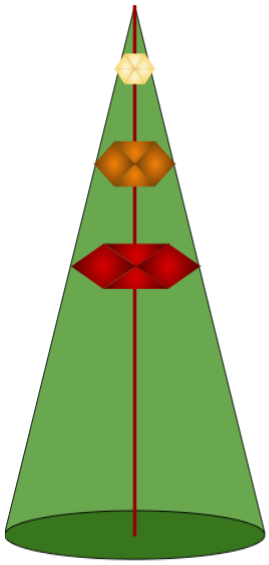
In other words, the LED spiral should have an equal distance of let's say 1, so $b = 1, a = 0$. This particular spiral is called an **Archimedean Spiral**. The garland however will use a sequence, lets say the Fibonacci sequence, to determine how many arcs we should have between any two units. For example between 0 and 1 units on the tree, we expect $F(1) = 1$ arc of garland at the top, similarly between unit 1 and 2 we expect $F(2) = 1$ arc. However between units 2 and 3 we now expect $F(3) = 2$ arcs. For simplicity's sake we are considering b to change suddenly at the end of a given slice, rather than formulate a **Fermat Spiral** which describes a more gradual increase in garland arcs. Below is the result:

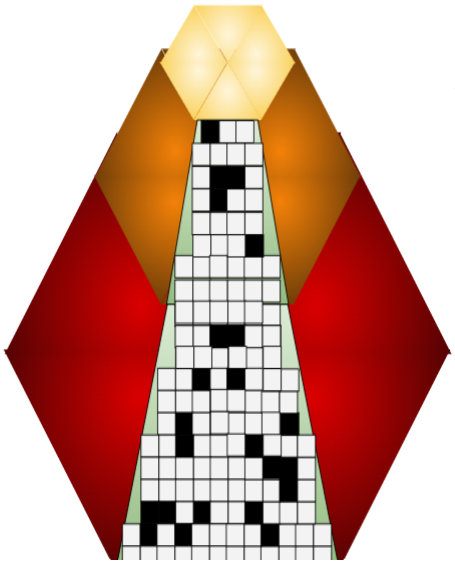


Additionally, let us consider if we wanted our LEDs to equally light the tree. Well, if we simplify the shape of the LEDs to be spherical, and diffuse light equally along its surface, we can calculate the desired brightness through its Candela, or the amount of light in a particular direction, as a 'bubble'. Then we ensure that this bubble just touches each other LED on the cord, and that the slant lengths between arcs either just touch or overlap. I couldn't quite find the Candela or Lumens of Christmas LEDs, so this is just another application of these spirals in geometric terms.

Another thing we might want to consider for determining how many loops we want to make is the length of the cord itself! Luckily, the formula for doing so is super simple. To calculate the length of the cord of LEDs, we simply let $f(\theta) = 0 + b * \theta$, where b is an unknown for now, and calculate the integral $\int_0^{TreeRadius} \sqrt{f(\theta) + f'(\theta)}\, d\theta$, and if my math is correct, $f'(\theta) = 0$. Thus you can set the upper bound to be the radius of your tree's base, and then select a b such that you get close to the length of your LED cord. Similarly you can do a summation of integrals between garland unit chunks with the same formula, calculating the integral piece-wise until you've run out of garland.

Now for the more difficult description: Branches and ornaments. Now, us CS Majors are allergic to the outdoors, so I'm going to build my own tree. Luckily some biologist with too much time figured out that there is a particular angle, the **Golden Angle** at which trees grow their limbs, roughly 137.5°. To visually think of this, imagine that every 'slice' of the tree with the height equal to the diameter of the branch will have about 3 branches. Next let us state that our branches split into two every horizontal unit, until the branch reaches the boundary of the cone. The angle between branches is random, however let the furthest two branches that share a root split up to 137.5° by its final branch. This makes our tree more 'dense' with branches at the bottom, but sparser towards the top. However, as the diameter gets smaller, the branches themselves need fewer splits to cover the area of the slice.





Now that we have a way of describing our branches determined by the diameter of said branch, let us determine a space on the tree for ornaments. Let us draw two regular hexagons inside the cone called hexagon upper [H_1], and hexagon lower [H_2], with edges connected by 6 trapezoids. Let's say we're sticking to the the Fibonacci sequence, and are determined to draw these regular hexagons every $F(i)$ down the slant, until we reach the base. We can estimate the number of branches by the lower bound [LB] of that unit slice [SL=Lower bound-Upper bound radius] with the formula, and the diameter of the branches [DB], and the rough estimate of the number of branches per unit vertically traveled 3, which results in the number of branches in a particular slice to equal

$$3 * \frac{(LB * SL)}{DB},$$

i.e. the number of branches we expect per slice multiplied by the number of slices we have multiplied by the amount of branches we can physically fit on those slices. For example, if we let our $DB = .25$ units, from units 0 to 1, the top of the tree can expect $3 * (1 * 1)/.25 = 12$ branches, the next slice from 1 to 2 can expect $3 * \frac{(2*1)}{.25} = 32$ branches, and so on. The area we obtain from each of the resultant trapezoids is equal to

$$A = (a + b)/2 * h,$$

where $h = F(i)$, $a = \frac{1}{2}H_{1,diameterflat}$, $b = \frac{1}{2}H_{2,diameterflat}$, with a total amount of branches equal to $\sum_{n=0}^{h} 3 * \frac{H*SL}{DB}$.

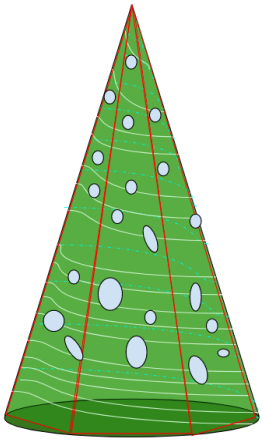
Now that we have our branches, we can determine the maximum number of ornaments per slice, as well as determining the likelihood of an ornament occupying a particular portion of the tree.

Let us investigate these trapezoids that arise from connecting the hexagons. We get six 'panes' of the tree with equal area $A = (a + b)/2 * h$. Note that this handles the triangle panes obtained at the top, if we consider an infinitely small regular hexagon. Also note that there might be a Real Analysis method of generalizing the following argument, but hexagons suffice.

Now how do I distribute the ornaments on these panes such that it is semi random, but not too random? Well first we determine an amount of area we want to have obscured by ornaments, and fill it in with some irregular shapes. Then imagine if we rotate a hexagon, we want it so that the area covered by these irregular ornaments is not equal to our original orientation. Note that we can place ornaments anywhere inside original panes, but only with the constraint that each pane has a proportionally equal amount of area covered by ornaments. The ornaments themselves can be distributed through some random static and the DBScan algorithm. The idea is that DBScan will select a certain amount of clusters (Random) and determine the best center point between a collection of points for a number of clusters. Say we end up with 3 clusters, or we might end up with 8 clusters. This method gives us a constant area to cover, but an irregular amount of ornaments to use per slice. Also as a consequence of the trapezoidal shape of our panes, we end up with less static and thus have a smaller proportion of clusters (assuming the ratio of pixels are a ratio of area difference).

So we end up with the probability of less ornaments towards the top, with less area to cover, and generally a higher change for more ornaments to cover the bottom, or large ornaments to cover the bottom. The exact numbers to use with DBScan might be based on how your noise is generated chaotically, but should remain consistent for every pane. When selecting the branch to use to hang the ornament, we simply choose the closest available branch on that slice. Thus larger slices are more likely to space clusters further out and apart than smaller slices with fewer branches to choose from.

To visualize what DBScan is doing, in the above image we might have one ornament in the top most trapezoid where the three black cells are, with DBScan treating the other two as noise. Meanwhile in the lower trapezoid we will have two ornaments, again where we have 3 or more black cells touching. Then we extrapolate the most appropriate point from those clusters, which might be the corner of each of the three points that make a cluster, and then find the closest branch. Notice however that the area covered by the two ornaments in the lower trapezoid and the single ornament in the upper trapezoid must cover an equal ratio of the trapezoid. My final Tree therefore looks something like this:



Happy Holidays!

Volunteer/Experience Opportunities

REUs: Summer Research in MSCS

If you are interested in being paid to collaborate on a research project with students from around the country off campus this summer, keep reading! To look through the programs available for Research Experiences for Undergraduates (REU's), check out this link! Most of them are done over the course of 8 – 10 weeks during the summer and include stipends around $4,000. Applications will open in November and most will be due between late January and early March.

Read the eligibility for each because many are restricted to certain years in school, certain majors, or US citizenship. The website has a variety of tabs at the top to help you find programs that apply to you! In particular, there are lots available for international as well as domestic students!

Most applications require a personal statement about why you would like to participate in the REU as well as letters of recommendation, so start looking into these sooner rather than later! And keep an eye on your email for an invitation to an REU application workshop coming later in November!

Make sure to reach out to us (mercur1@stolaf.edu and mainel1@stolaf.edu) if you have any questions!

CS Undergraduate Research

St. Olaf CS invites applicants for undergraduate research student work during Fall '22 and possibly Spring '23, for work on projects involving cloud computing or Raspberry Pi units, as part of the CSinParallel research group. Specifically, the work relates to (1) "Runestone Backend", an automated containerized parallel/cluster computations on Google Cloud using Kubernetes, and (2) the Self-Organizing

Cluster system for the Raspberry Pi, including system image development. Qualifications depend on the particular project, as described in the application details. Both are ongoing projects with flexible expectations for hours per week, and strong applicants who may have partial qualifications are encouraged to apply for one or both projects.

Please apply here, applications will be accepted and considered until further notice.

To submit an article, event, or anything else for publication in the Mess, email hilst1@stolaf.edu; to receive the Mess digitally each Friday, email habero1@stolaf.edu; visit http://wp.stolaf.edu/mscs/mscs-mess/ for a digital archive of previous MSCS Mess issues.

Jacob Hilst, Editor
Daniel Stoertz, Faculty Adviser
Ellen Haberoth, Mess Czar