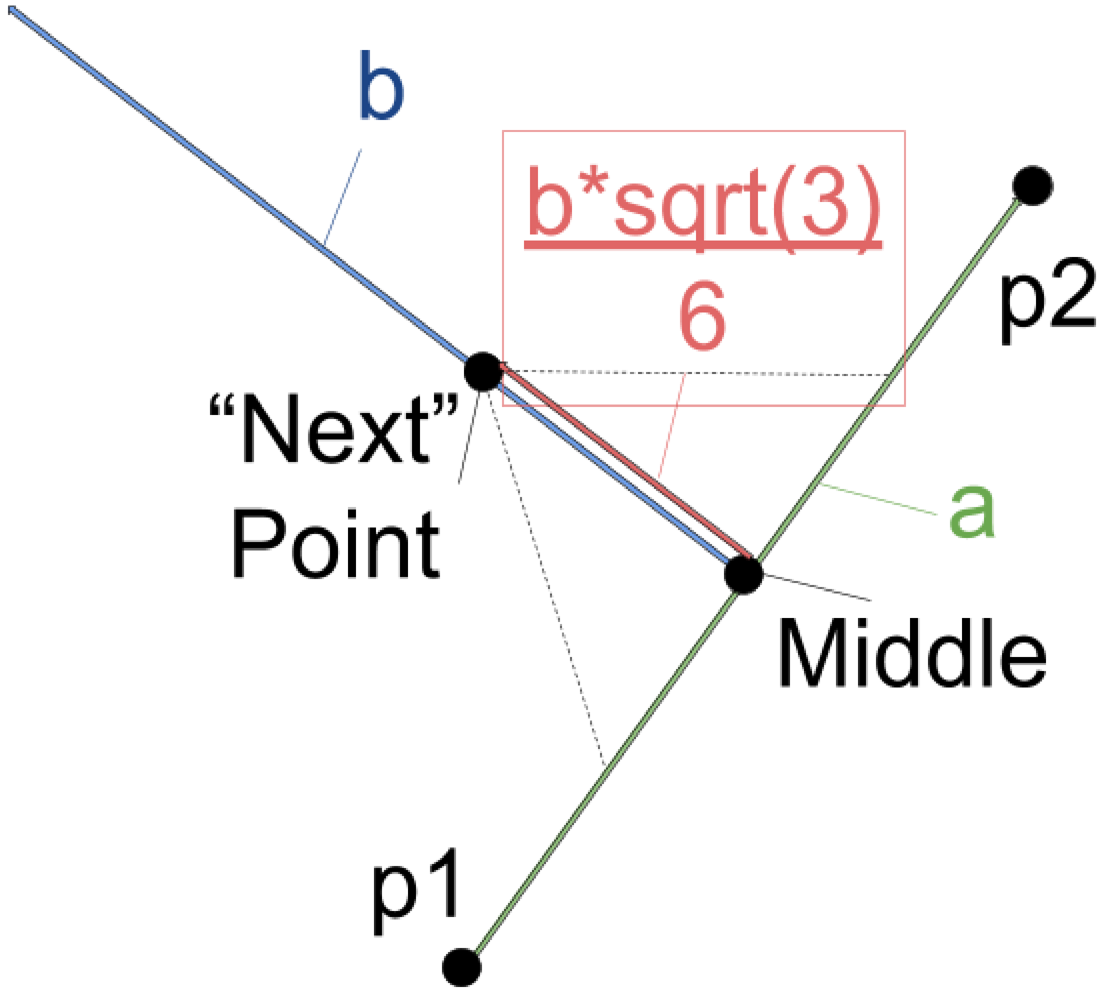
Peter Robe
CS 4613/6813 - Graphics
Assignment 1: Koch Snowflake

Arguments:

segment length, n

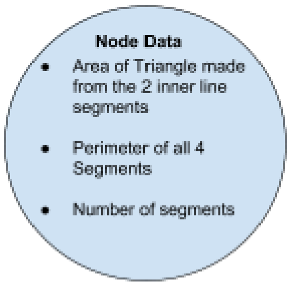
Implementation:

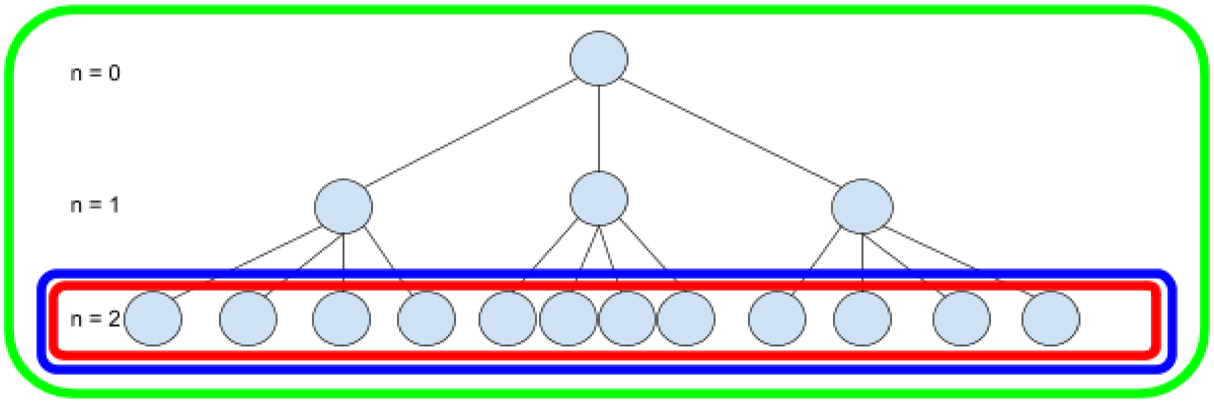
The drawing of the snowflake was pretty straightforward. I didn't find that difficult or time consuming. To find the "next" point (the tip of the new triangle), I subtracted p2 by p1 to get a (green). I then turned it perpendicular and made its origin the middle point to get b (blue). I took this vector and multiplied it by sqrt(3)/6 to get a vector that I add to the middle point to get my "next" point.



Peter Robe
CS 4613/6813 - Graphics
Assignment 1: Koch Snowflake

The hard part was the calculation of the table. I decided I wanted my program capable of extracting any information about the recursion at any level. When iterating through the Koch Snowflake, I created a tree of data (perimeter, number of segments, area) with each node being a call in the recursive function. That allowed me to later recursively search a certain depth of that tree and add up all of the nodes to get the perimeter and the number of segments. To get the area, I had to add up ALL of the values in the tree.


Node Data
• Area of Triangle made from the 2 inner line segments
• Perimeter of all 4 Segments
• Number of segments



Adding all the respective values at each of the nodes circled would get you the total Perimeter, Number of Segments, and Area. The very last recursive call is all that matters for the perimeter and the number of segments, but the area requires knowledge of previous calls.

As for the shaders, I just pass 4 uniform values x1, y1, x2, y2 for each line I draw. I didn't understand Matrix3D and all that at the time of doing this project.

Other Features:

- When the side length is increased/decreased, the image is scaled to fit within the window.
- The window icon is taken from a url if you are connected to the internet.

Reflection:

Upon inspection, my method is in no way the most efficient way of completing the project, but storing the information in a tree not only tested my knowledge of recursion more, but it provides me with all of the available information about each recursive call, not just the information at each level or recursion. This solution might be more helpful if recursive calls of the same level of iteration (n) didn’t produce exactly the same results. However, I don’t think that would ever happen. Although it does allow me to populate a table of values at every level of recursion equal to and below the provided n.

```
 n | Segment Count |   Perimeter   |      Area     |
 0 |             3 |      3.000000 |      0.433013 |
 1 |            12 |      4.000000 |      0.577350 |
 2 |            48 |      5.333333 |      0.641500 |
 3 |           192 |      7.111111 |      0.670011 |
 4 |           768 |      9.481483 |      0.682683 |
 5 |          3072 |     12.641965 |      0.688315 |
 6 |         12288 |     16.855938 |      0.690818 |
 7 |         49152 |     22.474552 |      0.691930 |
 8 |        196608 |     29.965635 |      0.692425 |
 9 |        786432 |     39.953194 |      0.692644 |
10 |       3145728 |     53.262867 |      0.692742 |
```

Peter Robe
CS 4613/6813 - Graphics
Assignment 1: Koch Snowflake