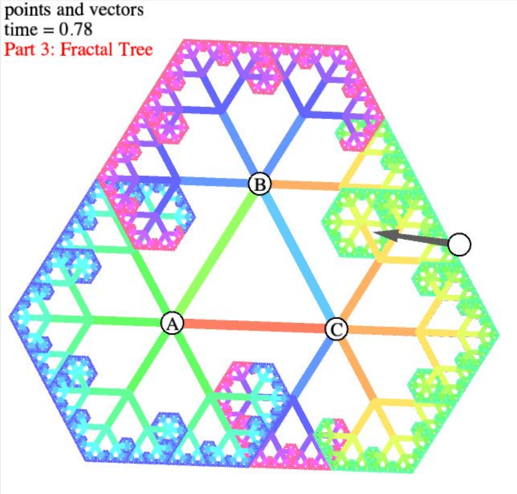
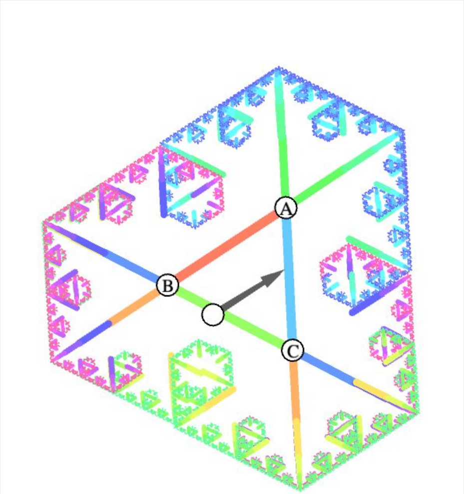
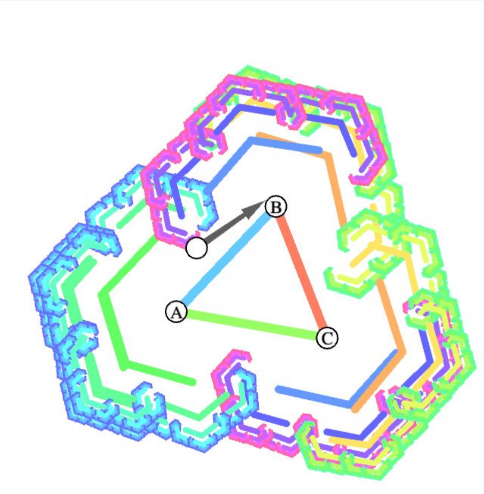
# Playing With Points and Vectors Part 3



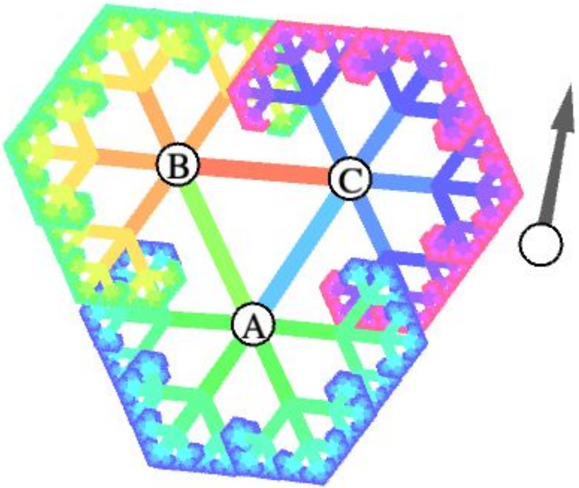

By: Kaushalya Chandraratna, Alexander Goebel, Philip Huynh

# PHSE 3: Problem Statement

- Given a set of 3 points {A, B, C}, create a visualization of a fractal tree based on branching triangles that simulate a Sierpinski-esque pattern.
- Assumptions:
  - Starting shape is a triangle
  - Each vertices branch out recursively into smaller triangles
  - A,B,C are not collinear

# PHSE 3: Solution outline

- By recursively calling our triangleTree function, we can create a series of branching triangles that emerge from each of the three vertices and creates 3 branches that decrease in size with every iteration.
- In order to keep the triangle oriented correctly, we used the determinant of the initial triangle two sides  to ensure that outward triangles didn’t overlap.
- 3 styles of the fractal tree can be outlined depending on which edge is shown



# PHSE 3: Solution math

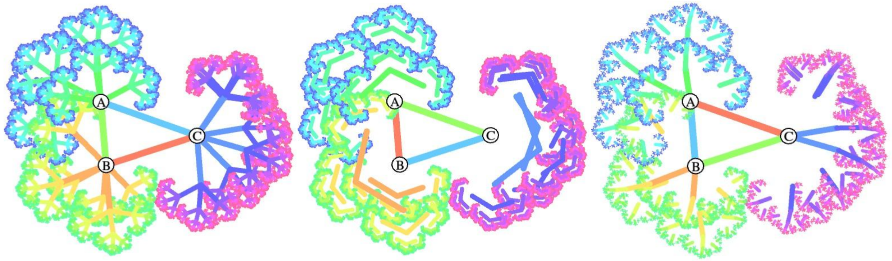
- The angle of rotation used at each branch is determined by using the angle between the two adjacent edges. When the points are in a CW orientation this means that the left angle can be calculated as angle(AB,AC) and the right angle can be calculated as angle(BC, BA). When the points are in a CCW orientation this means that the left angle can be calculated as angle(AC,AB) and the right angle can be calculated as angle(CB, CA)

JUSTIFICATION:

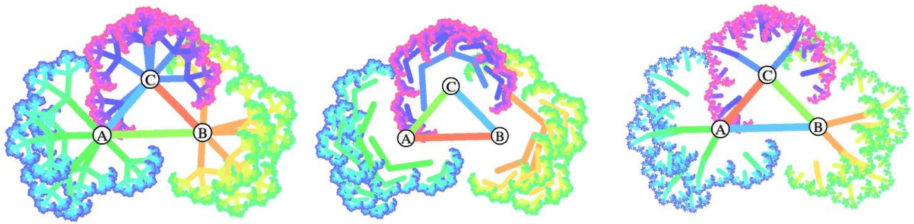
- When starting with a single line in a plane and continuously adding a rotated shorter line segment to the original line segment, a continuous curve is established through each iteration.

# PHSE 3: Solution examples and limits

- Works for CW, CCW

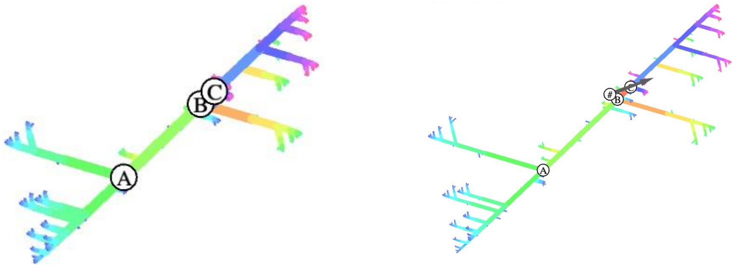



CW




CCW

- Limitations: when points {A,B,C} are collinear



# PHSE 3: Code

main GUI MyProject arrow circles ducks grid interpolation pt

```
//================================================================ PART 3

int branches = 8;
boolean showAB=true, showBC=false, showAC = false;
void showPart3(PNT A, PNT B, PNT C, PNT D) //
  {
  PartTitle[3] = "Fractal Tree";
  float leftAngle = angle(V(A,B),V(A,C));
  float rightAngle = angle(V(B,C),V(B,A));
  float leftAngle2 = angle(V(A,C), V(A,B));
  float rightAngle2 = angle(V(C,B), V(C,A));
  triangleTree(A,B,C,leftAngle, rightAngle, leftAngle2, rightAngle2,10,branches);
  triangleTree(B,C,A,leftAngle, rightAngle, leftAngle2, rightAngle2,200,branches);
  triangleTree(C,A,B,leftAngle, rightAngle,leftAngle2, rightAngle2,100,branches);
  triangleTree(A,C,B,leftAngle, rightAngle, leftAngle2, rightAngle2,200,branches);
  triangleTree(C,B,A,leftAngle, rightAngle, leftAngle2, rightAngle2,10,branches);
  triangleTree(B,A,C,leftAngle, rightAngle,leftAngle2, rightAngle2,100,branches);
  guide="j/k/l:show/hide solutions, 'a' to start/stop animation ";
  A.circledLabel("A"); B.circledLabel("B"); C.circledLabel("C");
}
```

# PHSE 3: Code

```
void triangleTree(PNT A, PNT B, PNT C, float leftAngle, float rightAngle, float leftAngle2, float rightAngle2,int c,  int branches)
  {
  if (branches==0)
    {
      return;
    }
    PNT leftBranch, rightBranch, centerBranch;
    if(det(V(A,B),V(A,C)) > 0)
      {
      leftBranch = P(B,V(0.5,V(A,B)).rotateBy(-leftAngle));
      rightBranch = P(B,V(0.5,V(A,B)).rotateBy(rightAngle));
      centerBranch = P(B, V(0.5,V(A,B)).rotateBy((-leftAngle + rightAngle)/3));
      } else
      {
      leftBranch = P(B,V(0.5,V(A,B)).rotateBy(leftAngle2));
      rightBranch = P(B,V(0.5,V(A,B)).rotateBy(-rightAngle2));
      centerBranch = P(B, V(0.5,V(A,B)).rotateBy((leftAngle2 + (-rightAngle2))/3));
      }

    color c1 =c;
    stroke(c1,60,150);
    strokeWeight((branches*5)*0.3); //alter strokeweight of each branch
    if (showAB){
      show(A,B);
     if (showBC){
       show(B,C);
     }
     if (showAC){
       show(A,C);
     }
     triangleTree(B,leftBranch,centerBranch, leftAngle, rightAngle, leftAngle2, rightAngle2, c + 20, branches-1); //left branch
     triangleTree(B,rightBranch,centerBranch, leftAngle, rightAngle, leftAngle2, rightAngle2, c + 20, branches-1); //right branch
     triangleTree(B,centerBranch,centerBranch, leftAngle, rightAngle, leftAngle2, rightAngle2,c + 20, branches-1); //left branch
```

# PHSE 3: Sources

- The solution was based on nature of fractal trees and solving for proper angle rotations.
- Design was based on Wikipedia's definition of a fractal tree and Sierpiński_triangle
- https://en.wikipedia.org/wiki/Pythagoras_tree_(fractal)
- https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle