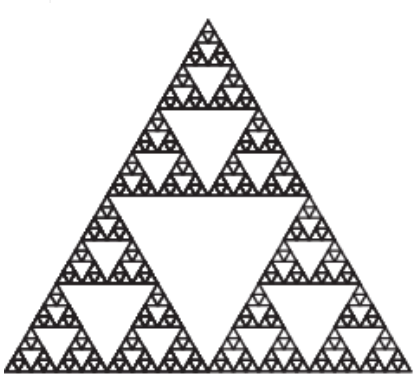
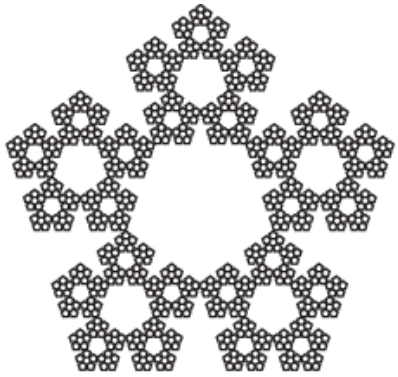
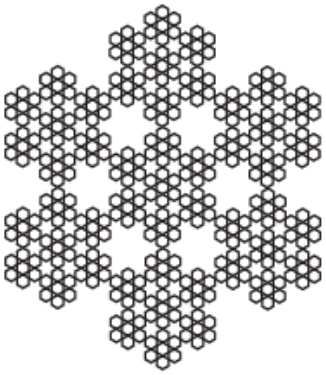
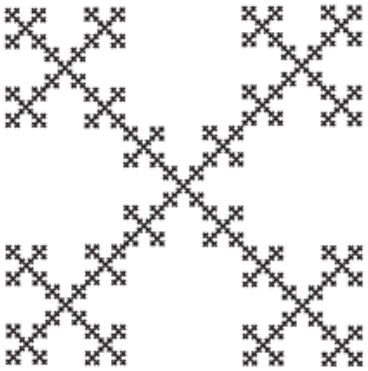
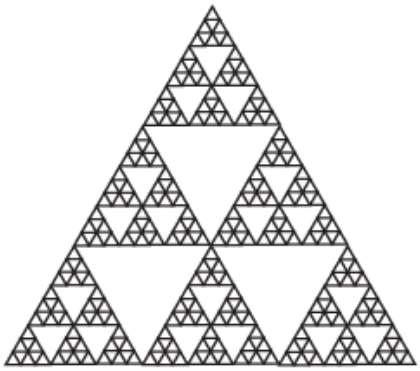
Week 06 – Fractals and Recursion

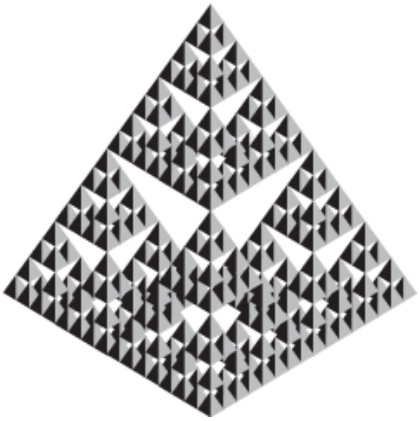


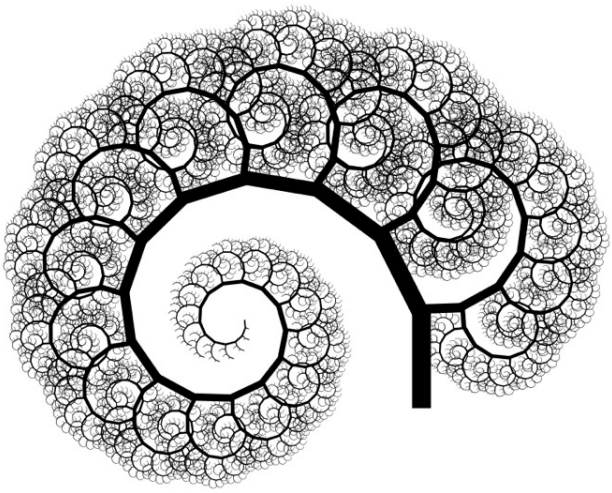


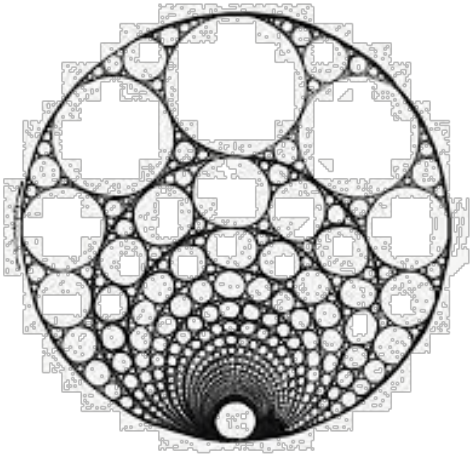












Iteration vs. recursion

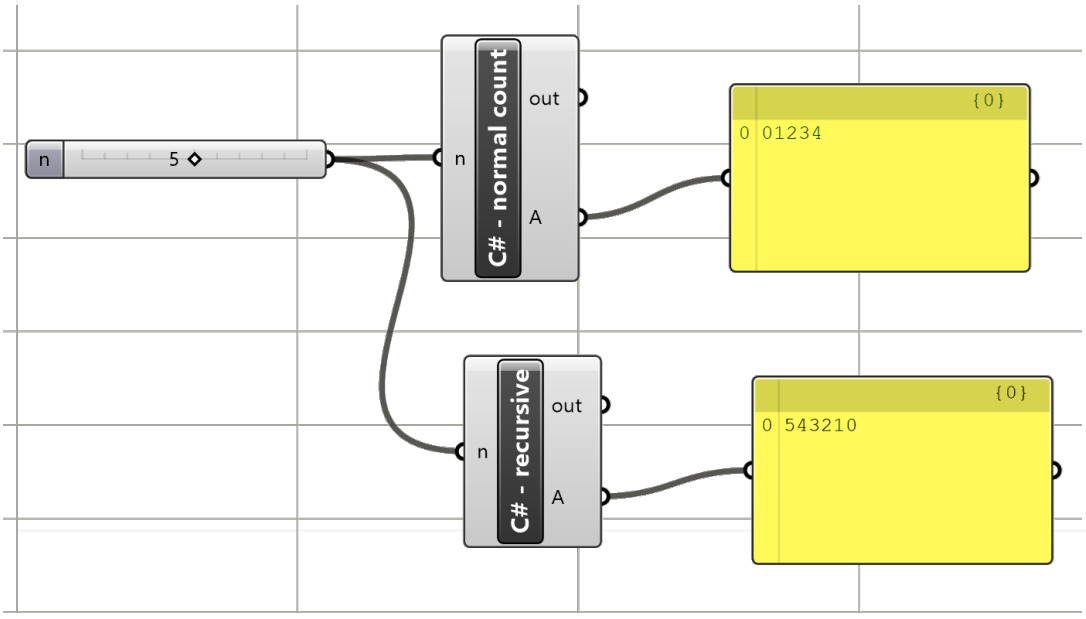
>>

iteration

```
private void RunScript(int n, ref object A)
{
  string output = "";

  for (int i = 0 ; i < n ; i++ ) {
    output += i.ToString();
  }
  A = output;
}
```

“recursion”

```
private void RunScript(int n, ref object A)
{

  string output = "";
  output = iter(n);
  A = output;
}

/**/
string iter(int level) {
  if (level == 0) return 0.ToString();
  else return level.ToString() + iter(level - 1);
}
```



Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 0; output = “0”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

i = 3; output = “”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 1; output = “01”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

i = 3; output = “”

i = 2; output = “”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 2; output = “012”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

level = 3; output = “”

level = 2; output = “”

level = 1; output = “”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 3; output = “0123”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

level = 3; output = “”

level = 2; output = “”

level = 1; output = “”

level = 0; output = “0”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 3; output = “0123”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

level = 3; output = “”

level = 2; output = “”

level = 1; output = “10”

level = 0; output = “0”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 3; output = “0123”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

level = 3; output = “”

level = 2; output = “210”

level = 1; output = “10”

level = 0; output = “0”

Iteration vs. recursion

iteration

```
private void RunScript(int n, ref object A)
{
   string output = "";

   for (int i = 0 ; i < n ; i++ ) {
     output += i.ToString();
   }
   A = output;
}
```

i = 3; output = “0123”

“recursion”

```
private void RunScript(int n, ref object A)
{

   string output = "";
   output = iter(n);
   A = output;
}

/**/
string iter(int level) {
   if (level == 0) return 0.ToString();
   else return level.ToString() + iter(level - 1);
}
```

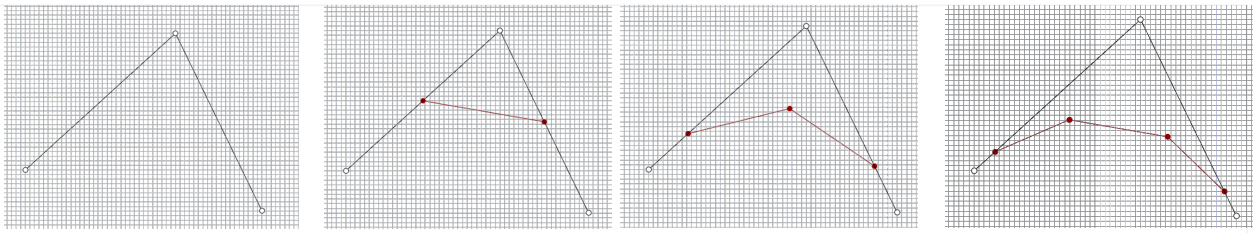
level = 3; output = “3210”

level = 2; output = “210”

level = 1; output = “10”

level = 0; output = “0”

Subdivision curve



iteration

>>

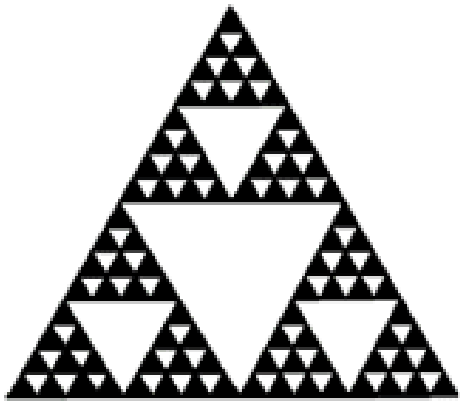
```
  for (int i = 0 ; i < n; i++) {
    points = pointIteration(points);
  }

  A = points;

}

/**/

public List<Point3d> pointIteration(List<Point3d> oldpoints) {
  List<Point3d> newpoints = new List<Point3d>();
  newpoints.Add(oldpoints[0]);
  for (int i = 1; i < oldpoints.Count; i++) {
    Point3d newPoint = (oldpoints[i - 1] + oldpoints[i]) / 2.0;
    newpoints.Add(newPoint);
  }
  newpoints.Add(oldpoints[oldpoints.Count - 1]);
  return newpoints;
}
```

recursion

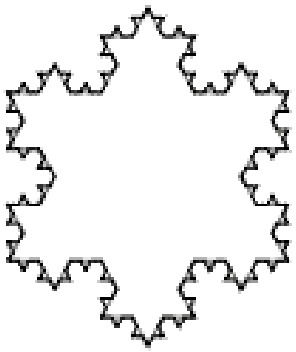
>>

```
    A = pointRecursion(points, n);


  }

  /**/

  public List<Point3d> pointRecursion(List<Point3d> oldpoints, int n) {
    if (n == 0) return oldpoints;

    List<Point3d> newpoints = new List<Point3d>();
    newpoints.Add(oldpoints[0]);
    for (int i = 1; i < oldpoints.Count; i++) {
      Point3d newPoint = (oldpoints[i - 1] + oldpoints[i]) / 2.0;
      newpoints.Add(newPoint);
    }
    newpoints.Add(oldpoints[oldpoints.Count - 1]);

    return pointRecursion(newpoints, n - 1);

  }
```

Classic fractals



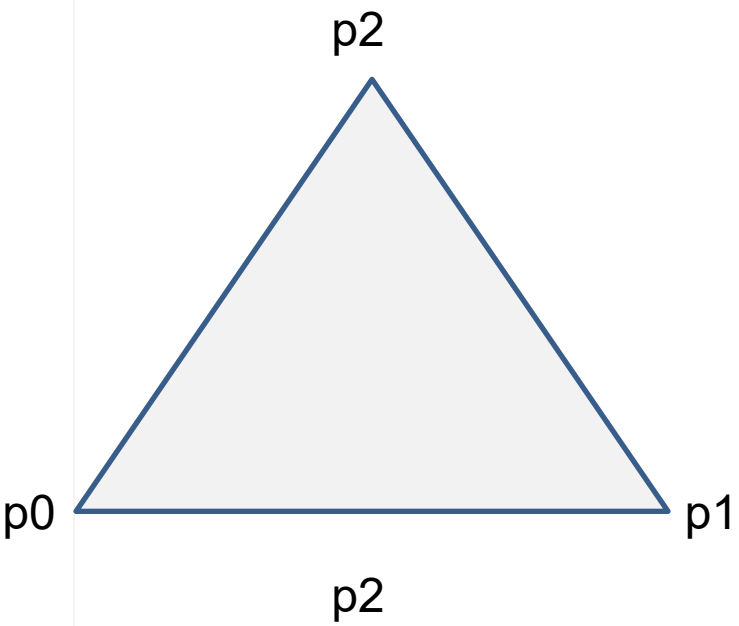
Sierpinski gasket



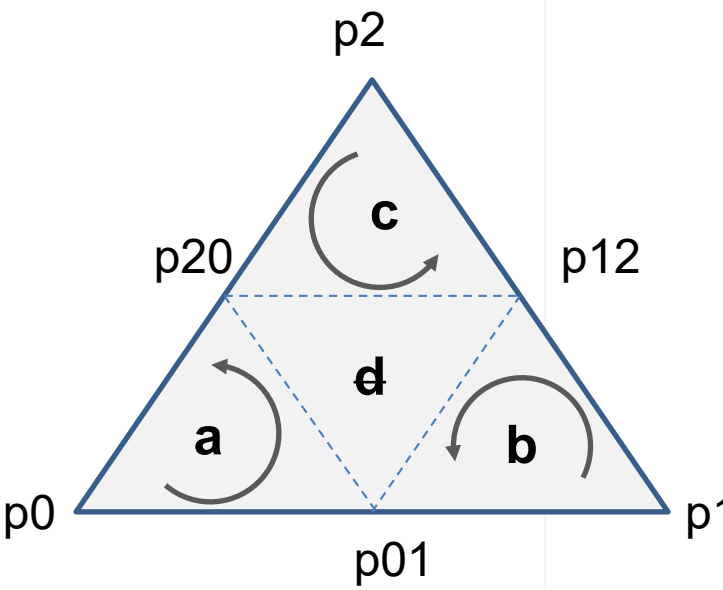
Von Koch snowflake

Sierpinski gasket

>>



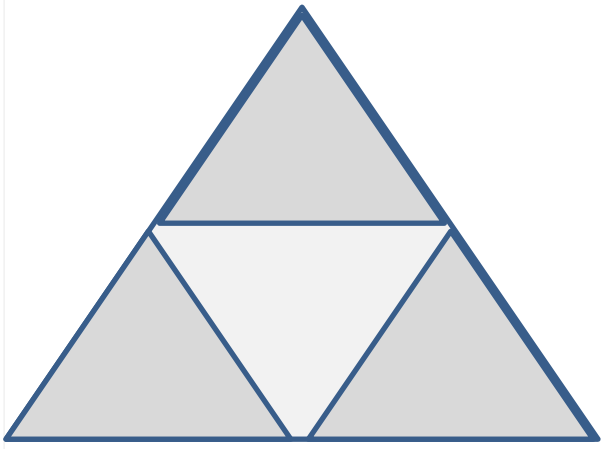
iter(p0, p1, p2)

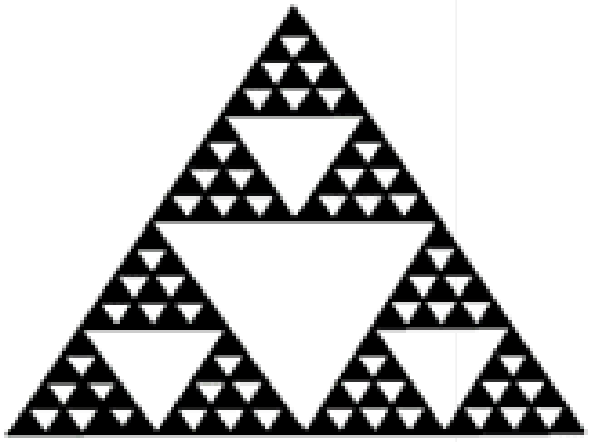


iter(p0, p01, p20)

iter(p1, p12, p01)

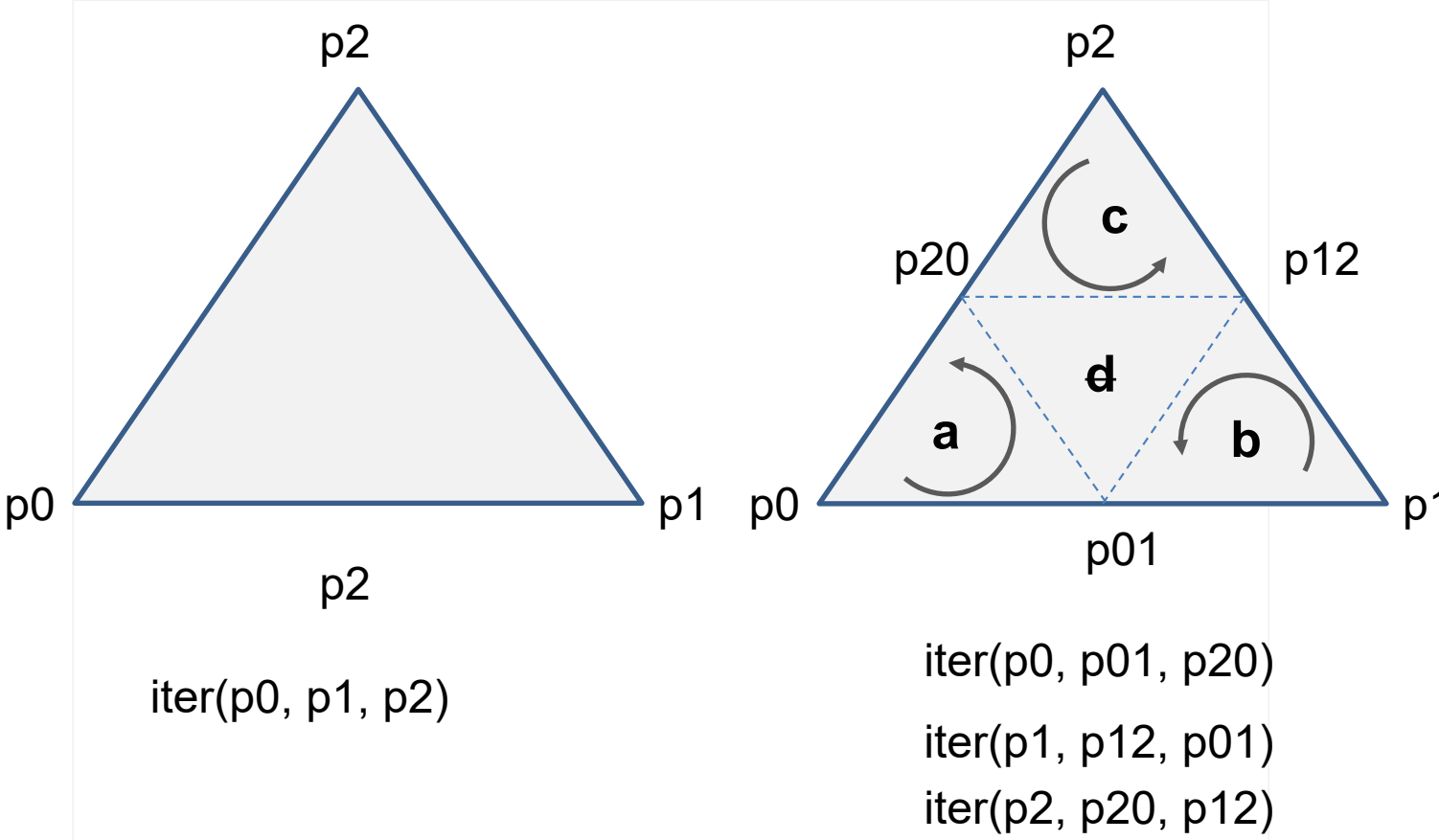
iter(p2, p20, p12)





Sierpinski gasket

>>

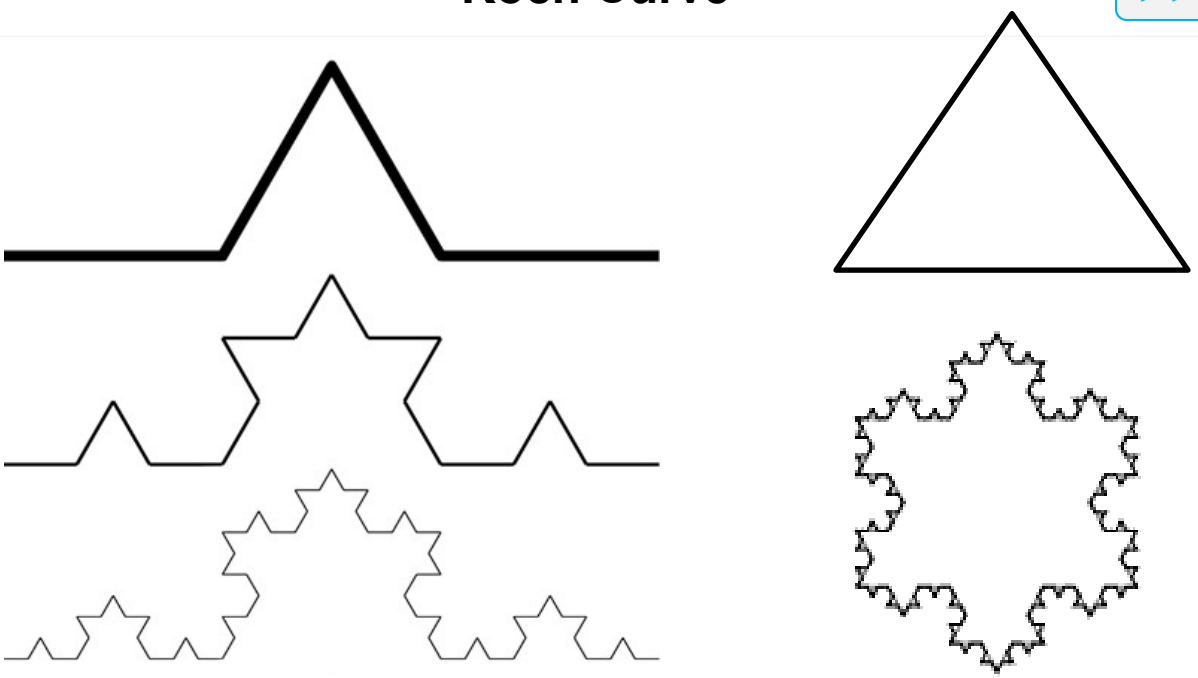



```
{
  A = recursiveTriangle(p0, p1, p2, n);
}
/**/
public List<object> recursiveTriangle(Point3d p0, Point3d p1, Point3d p2, int n) {
  List<object> returnObj = new List<object>();
  if (n == 0) {
    List<Point3d> tPoints = new List<Point3d>();
    tPoints.Add(p0); tPoints.Add(p1);tPoints.Add(p2);tPoints.Add(p0);
    returnObj.Add(new Polyline(tPoints));
    return returnObj;
  }

  Point3d p01 = (p0 + p1) / 2.0;
  Point3d p12 = (p1 + p2) / 2.0;
  Point3d p20 = (p2 + p0) / 2.0;

  List<object> returnObj0 = recursiveTriangle(p0, p01, p20, n - 1);
  List<object> returnObj1 = recursiveTriangle(p1, p12, p01, n - 1);
  List<object> returnObj2 = recursiveTriangle(p2, p20, p12, n - 1);

  foreach(object item in returnObj0) {returnObj.Add(item);};
  foreach(object item in returnObj1) {returnObj.Add(item);};
  foreach(object item in returnObj2) {returnObj.Add(item);};
  return returnObj;
}
```

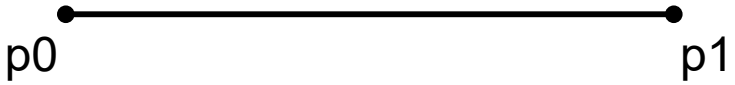
Sierpinski gasket

```
{

  A = recursiveTriangle(p0, p1, p2, n);

}

/**/

public List<object> recursiveTriangle(Point3d p0, Point3d p1, Point3d p2, int n) {

  List<object> returnObj = new List<object>();
  if (n == 0) {
    List<Point3d> tPoints = new List<Point3d>();
    tPoints.Add(p0); tPoints.Add(p1);tPoints.Add(p2);tPoints.Add(p0);
    returnObj.Add(new Polyline(tPoints));
    return returnObj;
  }

  Point3d p01 = (p0 + p1) / 2.0;
  Point3d p12 = (p1 + p2) / 2.0;
  Point3d p20 = (p2 + p0) / 2.0;

  List<object> returnObj0 = recursiveTriangle(p0, p01, p20, n - 1);
  List<object> returnObj1 = recursiveTriangle(p1, p12, p01, n - 1);
  List<object> returnObj2 = recursiveTriangle(p2, p20, p12, n - 1);

  foreach(object item in returnObj0) {returnObj.Add(item);};
  foreach(object item in returnObj1) {returnObj.Add(item);};
  foreach(object item in returnObj2) {returnObj.Add(item);};
  return returnObj;
}



  private void RunScript(Point3d p0, Point3d p1, Point3d p2, double n, ref object A)
  {
    minDist = n;
    A = recursiveTriangle(p0, p1, p2);
  }

  /**/

  double minDist; // Global variable - accessible in RunScript and recursiveTriangle
  public List<object> recursiveTriangle(Point3d p0, Point3d p1, Point3d p2) {

    List<object> returnObj = new List<object>();
    if (p0.DistanceTo(p1) < minDist || p1.DistanceTo(p2) < minDist || p2.DistanceTo(p0) < mir
      List<Point3d> tPoints = new List<Point3d>();
      tPoints.Add(p0); tPoints.Add(p1);tPoints.Add(p2);tPoints.Add(p0);
      returnObj.Add(new Polyline(tPoints));
      return returnObj;
    }

    Point3d p01 = (p0 + p1) / 2.0;
    Point3d p12 = (p1 + p2) / 2.0;
    Point3d p20 = (p2 + p0) / 2.0;

    List<object> returnObj0 = recursiveTriangle(p0, p01, p20);
    List<object> returnObj1 = recursiveTriangle(p1, p12, p01);
    List<object> returnObj2 = recursiveTriangle(p2, p20, p12);

    foreach(object item in returnObj0) {returnObj.Add(item);};
    foreach(object item in returnObj1) {returnObj.Add(item);};
    foreach(object item in returnObj2) {returnObj.Add(item);};
    return returnObj;
  }
```

Koch Curve

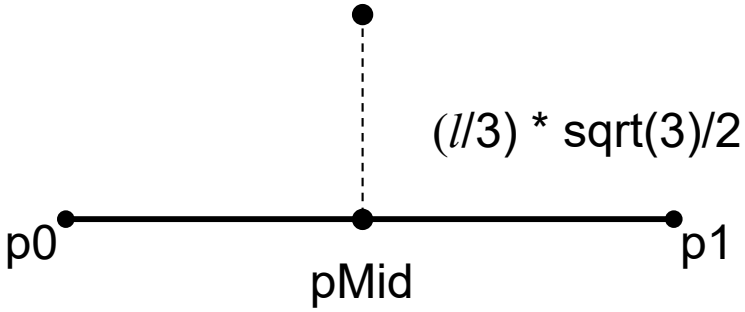


Given a line

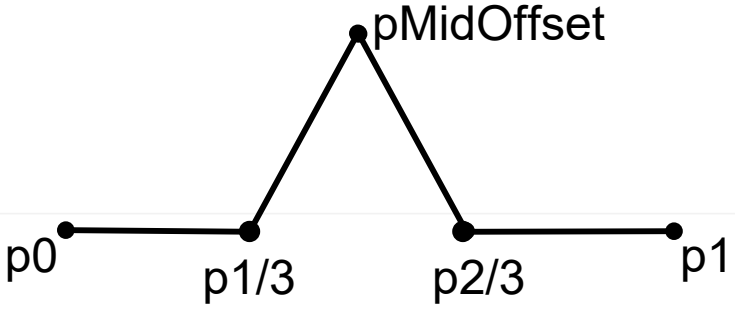


Divide in 3,
plus the mid point

Find the normal
vector and offset



Iterate with the 4
new segments



Koch Curve

```
public List<Line> recursiveVonKoch(Line l, int n) {

  List<Line> returnObj = new List<Line>();
  if (n == 0) {
    returnObj.Add(l);
    return returnObj;
  }

  Point3d p0 = l.From;
  Point3d p1 = l.To;
  Point3d p13 = (p0 * 2.0 + p1) / 3.0;
  Point3d p23 = (p0 + p1 * 2.0) / 3.0;

  Point3d pmid = (p0 + p1) / 2.0;
```

Divide in 3, plus the mid point

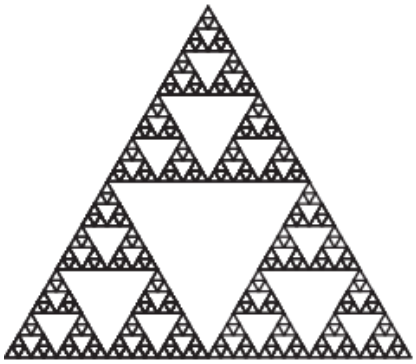
```
  Vector3d dir = p1 - p0;
  double dist3 = dir.Length / 3.0;
  Vector3d norm = Vector3d.CrossProduct(dir, Vector3d.ZAxis);
  norm.Unitize();
  norm *= dist3 / 2.0 * Math.Sqrt(3.0);
  Point3d pmidoff = pmid + norm;
```

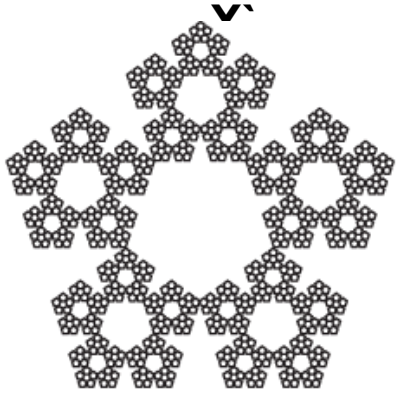
Find the normal vector and offset

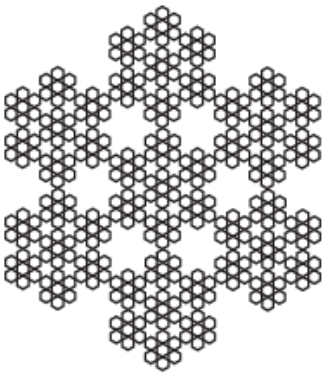
```
  Line l0 = new Line(p0, p13);
  Line l1 = new Line(p13, pmidoff);
  Line l2 = new Line(pmidoff, p23);
  Line l3 = new Line(p23, p1);
```

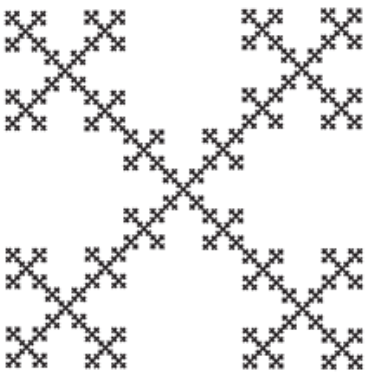
Iterate with the 4 new segments

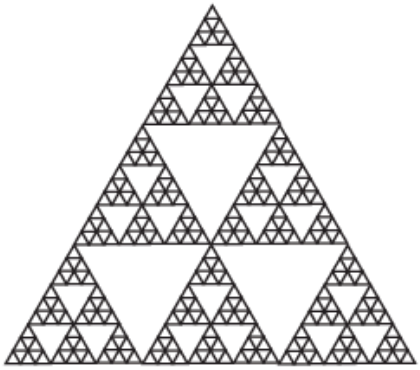
```
  List<Line> returnObj0 = recursiveVonKoch(l0, n - 1);
  List<Line> returnObj1 = recursiveVonKoch(l1, n - 1);
  List<Line> returnObj2 = recursiveVonKoch(l2, n - 1);
  List<Line> returnObj3 = recursiveVonKoch(l3, n - 1);

  foreach(Line item in returnObj0) {returnObj.Add(item);};
  foreach(Line item in returnObj1) {returnObj.Add(item);};
  foreach(Line item in returnObj2) {returnObj.Add(item);};
  foreach(Line item in returnObj3) {returnObj.Add(item);};
  return returnObj;
}
```

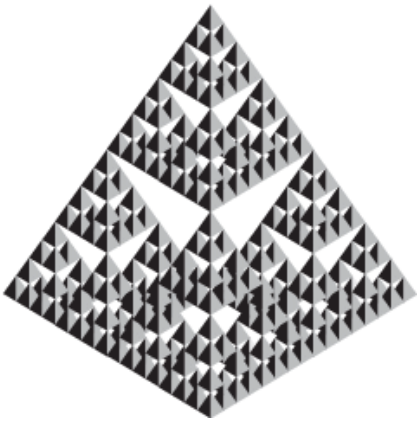


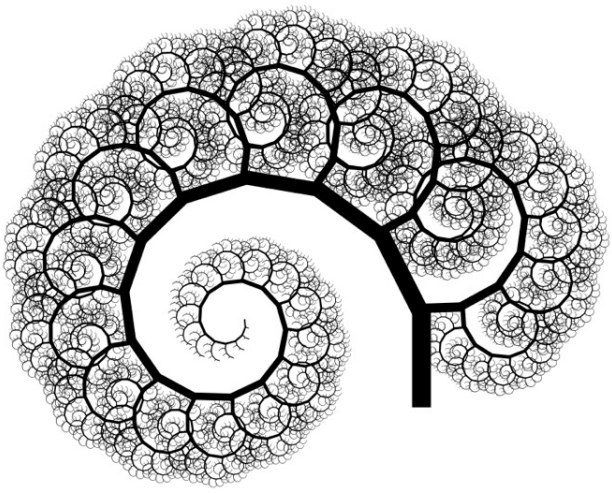


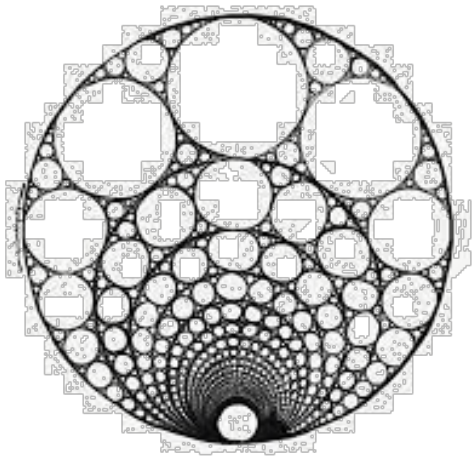








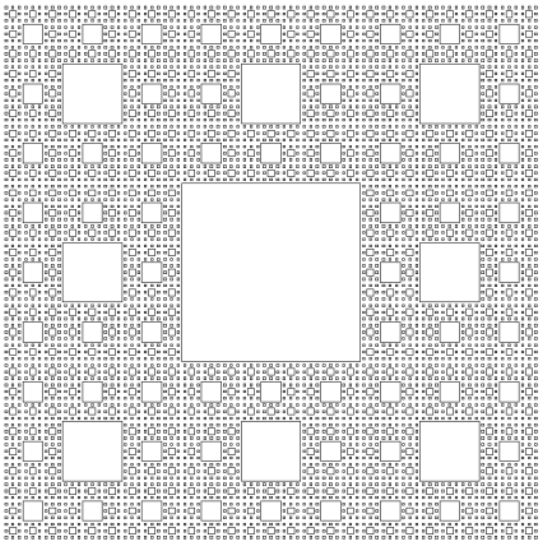


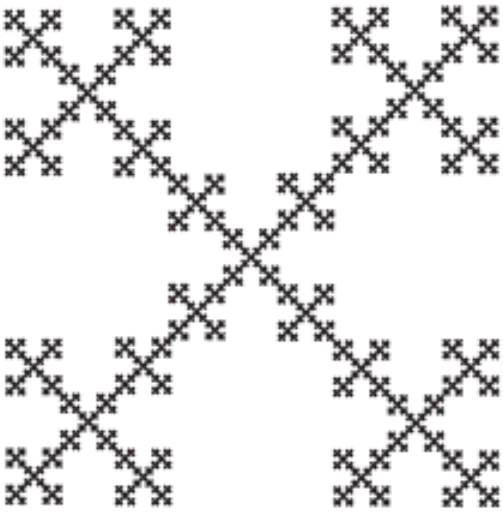


Homework

Modify the examples to produce 2 of the following:

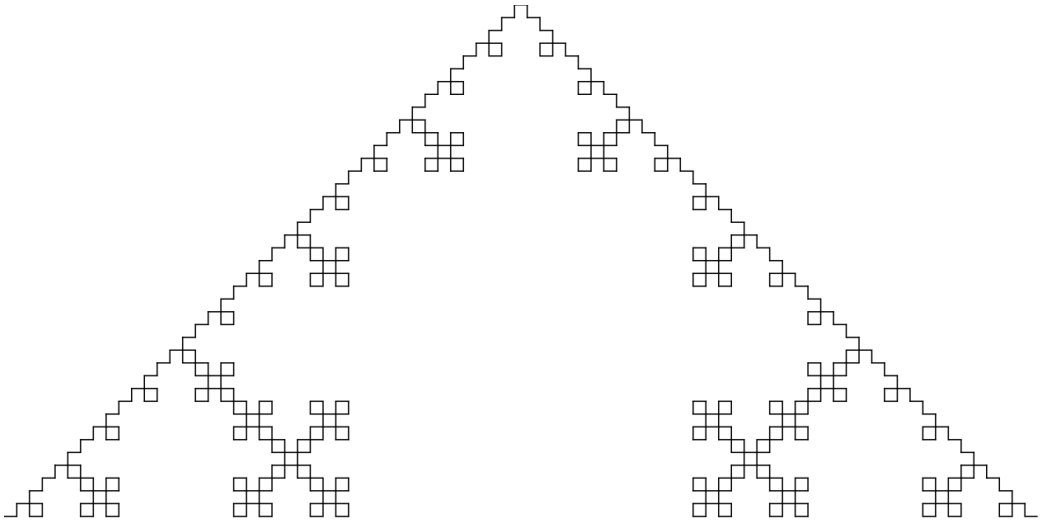
- Sierpinski carpet
- Square Kock Curve
- A fractal of your own chosing

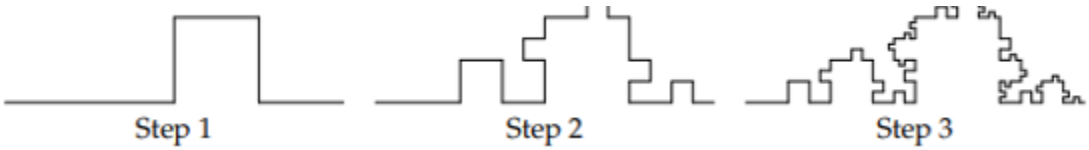




Regular Koch Curve

Square Koch Curve





Step 1

Step 2

Step 3