lecture 9

Object hierarchies

- call trees and GL_MODELVIEW stack
- fractals
- L systems

Last lecture:

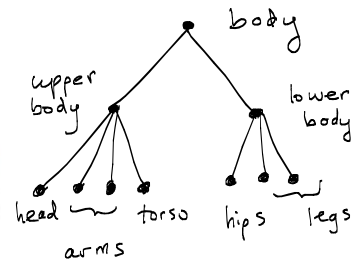
- hierarchy of bounding volumes of objects and scenes
- spatial partition represented as a tree (BSP trees, octrees)

Today:

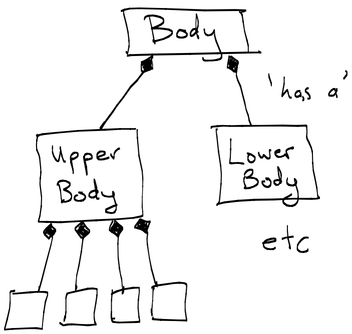
- how to model and draw object hierarchies ?

Example: human body

The object has a hierarchy of parts.



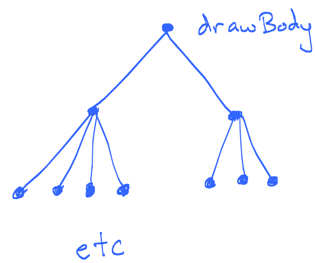
<u>ASIDE:</u> In an object oriented design, you might define a class hierarchy :



We will ***not*** discuss OOD approaches today, however.

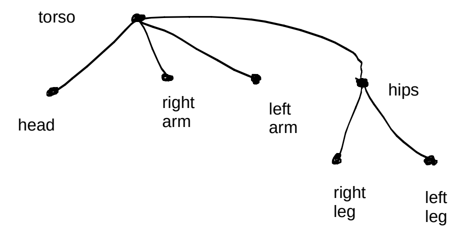
How to draw a person ? (call tree)

```
drawBody( ){
   drawUpperBody()
   drawLowerBody()
}

drawUpperBody(){
    drawHead()
    drawTorso()
    drawRightArm()
    drawLeftArm()
}

drawLowerBody(){
    drawHips()
    drawRightLeg()
    drawLeftLeg()
}
```



Hierarchy of coordinate systems

Consider a tree whose nodes are the object parts and whose edges represent coordinate transformations between parts.



In OpenGL, you use the GL_MODELVIEW matrix stack to traverse the tree.

Of course, neither of these two types of trees exists (as data structures).

The call tree does not necessarily correspond to the coordinate system tree.

OpenGL programs may involve both of these trees. Let's sketch some examples.

```
drawUpperBody(){
   glPushMatrix()
     drawTorso()        //  Head and arm coordinate systems
                        //   are relative to torso.
     glPushMatrix()
        glTranslate
        glRotate()       // Allow head rotation.
        drawHead()
     glPopMatrix()

     glPushMatrix()
        glTranslate()   // Allow shoulder joint motion.
        glRotate()
        drawLeftArm()
     glPopMatrix()

        :                 //    right arm too

   glPopMatrix()
}
```

```
drawLowerBody(){
   glPushMatrix()
     drawHips()              //  Leg coordinate systems
                             //   are relative to hips.
     glPushMatrix()
        glTranslate
        glRotate()           // Allow hip joint rotation.
        drawLeftLeg()
     glPopMatrix()

     glPushMatrix()
        glTranslate()
        glRotate()
        drawRightLeg()
     glPopMatrix()

   glPopMatrix()
}
```

```
drawLeftArm(   ...   ){
    glPushMatrix()
        glRotate( )
        drawLeftUpperArm()
        glTranslate( )
        glRotate( )
        drawLeftForeArm()
        glTranslate()
        glRotate()
        drawHand()              //   etc.  draw palm, fingers
    glPopMatrix()
}
```

Notation (used later in lecture)

$D_{upperbody}$ = [D_{torso} [T R D_{head}] [T R $D_{leftarm}$] ....]

$D_{leftarm}$ = [R $D_{leftupperarm}$ T R $D_{leftforearm}$ T R D_{hand}]

D_{hand} = ...

- D is draw (including lines, triangles, etc)
- [is glPushMatrix() and] is glPopMatrix()
- T and R are glTranslate() and glRotate()

lecture 9

Object hierarchies

- call trees and GL_MODELVIEW stack
- fractals
- L systems

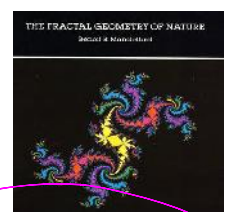
Many natural objects have complicated geometry.

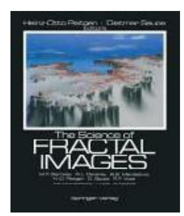
http://paulbourke.net/fractals/googleearth/

"How long is the coastline of Britain? ....

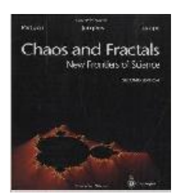
Statistical Self-Similarity and Fractional Dimension", B. Mandelbrot, *Science*, 1967

Fractals entered computer graphics in 1980's....

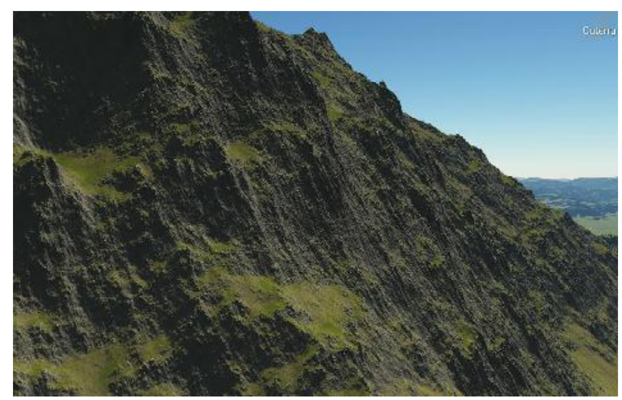




See video link for Mandelbrot set
http://kottke.org/10/10/benoit-mandelbrot-rip

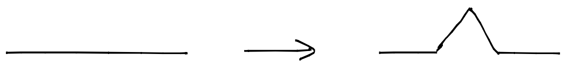


Here is what computer graphics can easily do now,...



But let's go back to the beginning,... the first fractals.

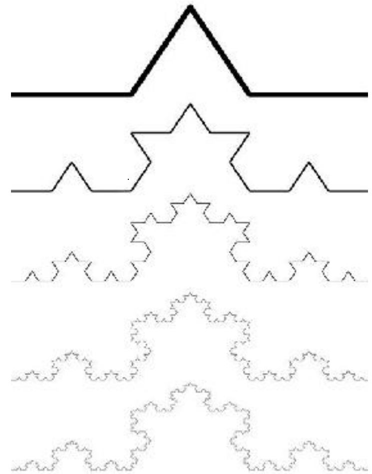
Koch Curve (1903)



Start with a line segment.

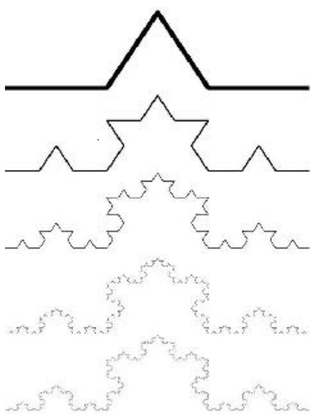
Replace the line segment with 4 line segments, each of length 1/3 the original.

Repeat (recursively)....



As n goes to infinity, the Koch Curve ...

- remains continuous
- has infinite length
- has no tangent *anywhere*
- is ***self-similar*** (a key property of fractal geometry)



length

$\frac{4}{3}$

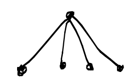
$\left(\frac{4}{3}\right)^2$

$\left(\frac{4}{3}\right)^3$

$\left(\frac{4}{3}\right)^4$

$\left(\frac{4}{3}\right)^5$

```
def koch(i):    //   0 < i < infinity
   if i == 0
      drawline()
   else if i > 0
      glPushMatrix()
         glScalef(1/3, 1/3,1/3 )
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(60, 0.0, 0.0, 1.0)
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(-60, 0.0, 0.0, 1.0)
         glRotatef(-60, 0.0, 0.0, 1.0)
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(60, 0.0, 0.0, 1.0)
         koch(i-1)
      glPopMatrix()
```



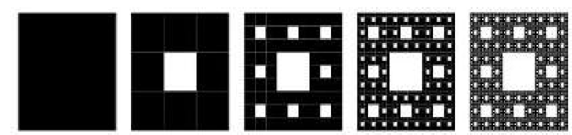
In this example, the call tree corresponds to coordinate system hierarchy tree.

The branching factor is 4.

The draw commands occur at the leaves which are all at the same depth.

Sierpinski Carpet

Start with square, partition into 9 squares of width 1/3, and delete the central square. Repeat recursively.

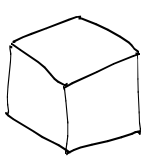


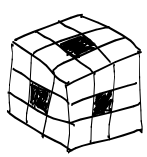
area $\quad 1 \quad \frac{8}{9} \quad \left(\frac{8}{9}\right)^2 \quad \left(\frac{8}{9}\right)^3 \quad \left(\frac{8}{9}\right)^4$

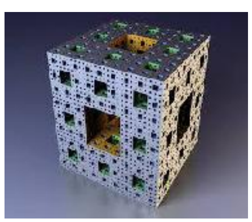
Area goes to 0 as n goes to infinity.

Sierpinski Cube

Start with cube, partition into 27 subcubes of width 1/3, and delete the 7 cubes containing the central xyz axes. Repeat recursively.







volume

$1 \quad \frac{20}{27} \quad \ldots \quad \left(\frac{20}{27}\right)^4$

Fractal dimension

Calculus deals with objects that have integer dimension.

dim(line segment) = 1

dim(square) = 2

dim(cube) = 3

Fractals have a non-integer dimension.

Fractal dimension

Definition is based on "self-similarity across scale".

Assume our set (object) is in R^n and has the following property:

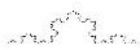
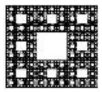
We can scale it by some S > 1 in each of the n dimensions, such that the scaled object consists of C translated and/or rotated copies of the original one.

Then the set has fractal dimension D where:

$$C = S^D$$

or equivalently

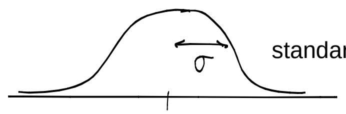
$$D = \log(C) / \log(S)$$

	S	C	D = log(C) / log(S)	
line segment	2	2	1	
square	2	4	2	
cube	2	8	3	
Koch curve	3	4	~1.26	
Sierpinkski carpet	3	8	~1.89	
Sierpinkski cube	3	20	~2.73	

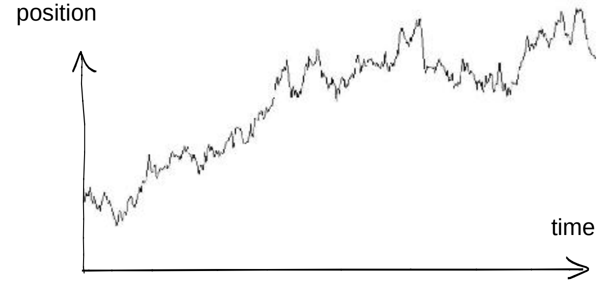
Fractals in nature are typically random.

To generate models of random fractals, we use random variables.

e.g. normal distribution ("Bell curve")



Example: Random Walks ("drunken sailor")



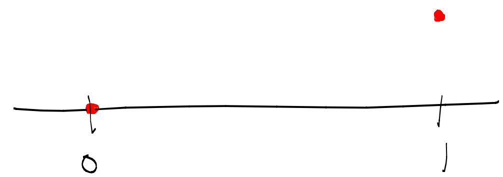
Step size at time t has a normal distribution.

Random Fractals

How could we compute fractal "random walks" ?

These are random walk curves that continue to appear rough as we "zoom in".

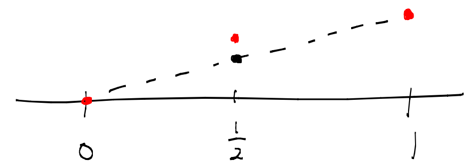
Midpoint displacement method [Fournier et al 1982]



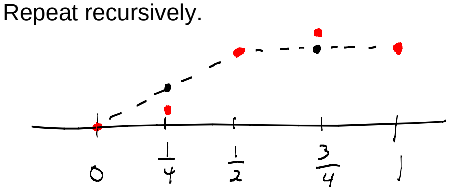
First, initialize the curve to 0 everywhere.
Then, choose endpoints of the curve (somehow).

How to interpolate ?

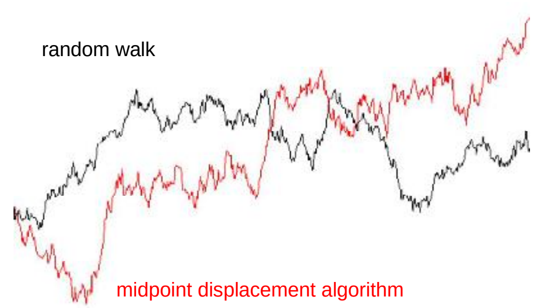
Consider midpoint. Set midpoint value to be average of values at endpoints plus a random displacement.



Repeat recursively.




```
def midpointDisplacement( a, std, roughness) {
   //  a is an array
   //  roughness is a scale factor between 0 and 1.
   //  roughness = 1/ sqrt(2) is called Brownian motion

   newStd = roughness* std

   size = len( a)
   if (size <= 2)
      return a
   else{                              // *Python syntax.
      middle = size / 2
      a[middle]    = (a[0] + a[size-1] ) /2 + rand.normal( newStd )
      a[0:middle+1] = midpointDisplacement(
                           a[ 0:middle+1 ], newStd, roughness )
      a[middle:size] = midpointDisplacement(
                           a[ middle:size ], newStd, roughness )
      return a
   }
}  // Subtle note:   midpointDisplacement() only
     changes the value of the midpoint.   Thus, a[middle] does not get
     changed in the second and third recursive calls.
```



What should be the probability distribution of random displacements at each level of the recursion ?

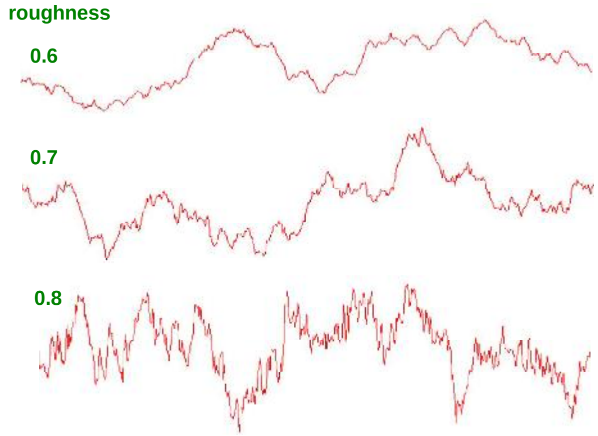
The math is very advanced and subtle (and not our concern in COMP 557).

The algorithm is simple and flexible, for example, just scale the standard deviation of the displacement.

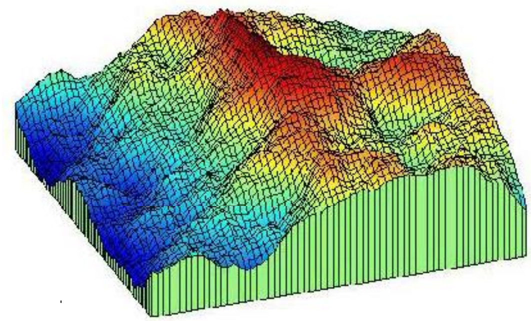
[Think of *drunken caffeinated sailor* taken faster steps, each of smaller size.]

```
def  midpointDisplacement( a, std, roughness) {

    //   roughness is a scale factor between 0 and 1.
    //   roughness =  1/ sqrt(2)  is called Brownian motion

    newStd =  std * roughness

    size = len( array)
    if (size <= 2)
        return array
    else{

        middle = size / 2
        a[middle]      =  (a[0] +  a[size-1] ) /2 + rand.normal( newStd )
        a[0:middle+1]  =  midpointDisplacement(
                                  a[ 0:middle+1 ],  newStd, roughness )
        a[middle:size] = midpointDisplacement(
                                  a[ middle:size ],  newStd, roughness )
        return a
     }
}
```

Examples



Straightforward extension to 2D doesn't work so well.
But more clever methods work very well.



Anytime you see something like this in a Hollywood movie, ... it isn't real. It was made with such fractal-based algorithms.

lecture 9

Object hierarchies

- call trees and GL_MODELVIEW stack
- fractals
- L systems

Recall notation from earlier ...

$D_{upperbody}$ = [D_{torso} [T R D_{head}] [T R $D_{leftarm}$] ...]

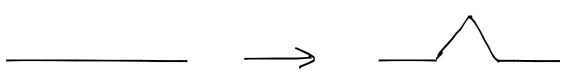
```
drawUpperBody(){
   glPushMatrix()
     drawTorso()

     glPushMatrix()
         glTranslate
         glRotate()
         drawHead()
     glPopMatrix()

     glPushMatrix()
         glTranslate()
         glRotate()
         drawLeftArm()
     glPopMatrix()
          :
   glPopMatrix()
}
```

K ⟶ [S K T R K T R' R' K T R K]

```
def koch(i):
   if i == 0
      drawline()
   else if i > 0
      glPushMatrix()
         glScalef(1.0/3,1.0/3,1.0/3)
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(60, 0.0, 0.0, 1.0)
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(-60, 0.0, 0.0, 1.0)
         glRotatef(-60, 0.0, 0.0, 1.0)
         koch(i-1)
         glTranslatef(1.0, 0.0, 0.0)
         glRotatef(60, 0.0, 0.0, 1.0)
         koch(i-1)
      glPopMatrix()
```

(L systems) Notation: "production"

K ⟶ [S K T R K T R' R' K T R K]

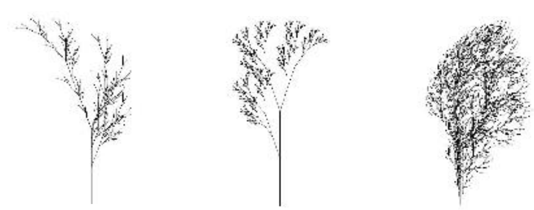


We replace symbols with strings (of symbols).

Most of you are familiar with this concept from formal grammars and language theory e.g. compilers.

L systems

Introduced by theoretical biologist Astrid Lindemayer (1960's) to describe structure and growth of biological systems, especially plants.

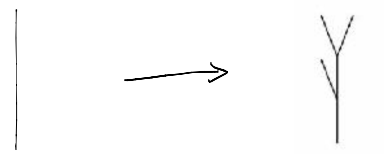


Later adopted by computer graphics, for drawing objects by recursive substitutions (like fractals, *but finite*).

@ U.Calgary

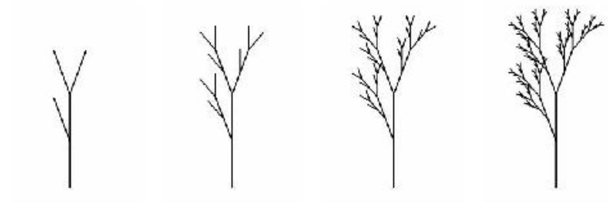
http://algorithmicbotany.org/papers/abop/abop.pdf

Example



L → [S D T [R L] D T [R' L] R L]

D is 'draw a line'
T, R, S, are translate (by 1), rotate (30), scale (by 1/2)



ASIDE: Formal grammar

We have an "alphabet" of symbols.
Think of OpenGL library, plus drawX().

We start with an "axiom" string or starting symbol.

We then replace symbols, using productions
Think of a function calling another function (or recursion).

"Parametric L systems"

- the symbols can have parameters
  e.g. R(30) rotate CCW by 30 deg.

- can keep track of the level of recursion,
  specify a base case e.g.

L(n) → [S D T [R L(n-1)] D T [R' L(n-1)] R L(n-1]

L(0) → D T

As computer scientists, you should find nothing conceptually new here.

"Probabilistic L systems"

Different productions can occur with different probabilities.

Again, nothing new here. Essentially this means....

```
def   draw() {
    ....
    if  (rand()  >   p0)       //   rand() returns value in [0,1]
        draw1()
    else
        draw2()
     ....
}
```

"Open L systems"

Use global variables (instead of random numbers) to determine which productions get called or what the parameters passed are.

e.g. Only grow a leaf if it receives direct sunlight rather than being in shadow
(we'll talk about lighting in a few weeks).

Model soil based on terrain shape and allow
plants to compete for light and water...

book (2005)

http://www.graphics.stanford.edu/papers/ecosys/
(paper from 1998)

Announcements

next few weeks (revised schedule)

9. object hierarchies
   OpenGL transformation stack, fractals, L-systems
10. smooth curves and surfaces
   cubics: Hermite curves & Catmull-Rom splines, bicubic surfaces
11. meshes
   level of detail, edge collapse

Rendering: light, material, texture, transparency

12. lighting, material
   ambient/point/spot, diffuse/mirror/glossy

13. review + exercises
14. midterm exam (Thursday Feb. 19)