Adding Visual Realism

Road to Point Reyes (see figure)

Texture Mapping

Fractals

Particle Systems

Texture Mapping

Avoid boring, flat, smooth surfaces

Avoid actually modeling each surface detail

Have patch of texture defined - Brodatz textures

Map it all over the object

Two parts:

1) Texture mapping
eg image distortion

2) Texture filtering
eg antialiasing

Texture patch application:

1) Tiling
eg light and dark marble tiles
   repetitive pattern on bark

2) Nontiling
eg billboard, painting
Texture-Surface Transformation  

Can do texture mapping in either direction:

1) texture scanning
   texture space to object space to image space

2) pixel-order scanning
   image space to object space to texture space

Texture to object space

parametric linear functions

\[ u = a_u s + b_u t + c_u \]

\[ v = a_v s + b_v t + c_u \]

Object to image space

regular viewing and projection transformations concatenated

Problem with mapping texture space to image space

the texture patch often doesn't match up with the pixel boundaries
Mapping from image space to texture space

Disadvantage:

Must compute inverse to the viewing and projection transformations

Advantage:

Don't need to subdivide pixels

Can easily incorporate image processing - e.g. filtering

Example:

Map texture onto cylindrical surface defined by:

\[ u = \theta; \quad v = z \]

\[ 0 \leq \theta \leq \frac{\pi}{2}; \quad 0 \leq z \leq 1 \]

In \(x,y,z\):

\[ x = r \cos \theta; \quad y = r \sin \theta; \quad z = v \]

Texture:
Map texture array to the surface

pattern origin to lower left corner

\[ u = s^{1/2}; \quad v = t \]

Select viewing position and apply inverse transformation

Map image coordinates to object space

\[ u = \tan^{-1}(y/x); \quad v = z; \]

Map object space to texture space

\[ s = 2u^{1/1}; \quad t = v; \]

Antialiasing:

Why may need to do it?

Interference of sampling rate due to pixel spacing and texture pattern

How do it?

One simple way: pyramid weighted filtering

Project larger pixel area from image space

Use pyramid weighting
Fractals

Man-made Objects

Flat surfaces

Smooth curved surfaces

Objects in Nature

Rough jagged edges

eg lightening bolt

nice to be able to specify just the ends points and not each little jag

Get roughness and jagginess using fractals

Fractals are self-similar

Start with Artificial Fractals

Recursively defined curves
Koch Curves - (1904 - Sweden - Helge von Koch)

Infinite length within a finite area

Formation Process for $K_i$ from $K_{i-1}$

For each straight line at $K_{i-1}$ (of length $L$)
- Break line into thirds (of length $L/3$)
- Replace middle third with two lines of length $L/3$
  with 60 degree angle between first new line and the original line, and second new line joining up two unattached ends

Start with a triangle, get Koch Snowflake
Consider the length of $K_0$ to be 1

Then $|K_0| = ?$

$|K_1| = ?$

$|K_i| = (4/3)^i$

As $i$ goes to infinity, $|K_i|$ goes to infinity

But the whole thing is still within a finite area

Look at objects of this nature as having a dimension
greater than a line (1)
but less than a plane (2)

That is a fractional dimensional object

fractal

This new dimension is called the Hausdorff-Besikovich dimension, $D$

$$D = \log (N) / \log (1/S)$$

where $N$ is the number of line segments going from one stage to the next

and $S$ is the length of each segment, relative to the length of segments in the previous level

For Koch Curve:

$N = 4; \quad S = 1/3; \quad D = \log 4 / \log 3 = 1.2619$
Other artificial fractals:

C-Curve

\[ N = 2, \quad S = \frac{1}{\sqrt{2}} \quad D = \frac{\log 2}{\log \sqrt{2}} = 2 \]

(see figure)

Dragon Curve

One elbow up, the next down

\[ N = ?, \quad S = ?, \quad D = ? \]

(see figure)
Hilbert Curves

Space filling curves

As order goes to infinity,
every point in the area is passed through

Four basic primitives:

Connected by extra lines as follows:
Hilbert's Curve Rules:

\[ A_{i+1} = B_i \text{ up } A_i \text{ right } A_i \text{ down } C_i \]
\[ B_{i+1} = A_i \text{ right } B_i \text{ up } B_i \text{ left } D_i \]
\[ C_{i+1} = D_i \text{ left } C_i \text{ down } C_i \text{ right } A_i \]
\[ D_{i+1} = C_i \text{ down } D_i \text{ left } D_i \text{ up } B_i \]
Curve never crosses itself
Curve is arbitrarily close to any point in square
Length of curve is infinite
$L_{i+1}$ versus $L_i$?
Strictly self-similar curves:

- Koch
- Peano, etc

Statistically self similar

- Fern
- Coastline

Mapmaker puts in bays, peninsulas, fiords

Accurate? No! eg Cape Cod

If look at coastline at different scales
get similar pattern of wiggles and bays

What is the length of the coastline?
I step it off
My cat steps it off
An ant steps it off

eg measure it with dividers set at different lengths
length depends on scale at which it is measured

Coastline fractal dimensions about 1.15 to 1.25

Other examples

- clusters of stars
- shapes of snowflakes
How to program statistically self-similar fractals

Random Midpoint Displacement Algorithm

For each line segment in object,

Replace it by an "elbow" with random offset

![](image)

Line is ab
L is its bisector
m is its midpoint

choose t randomly

\[
c = (m_x - (by - ay) t, \ m_y + (b_x - a_x) t )
\]

For each segment choose a new random value of t

Use normal (Gaussian) distribution with mean of 0

Standard deviation of \( s \)

Recursively apply such an algorithm
At each recursive level can change $s$ by multiplying it by a factor $f$

$f = 1$ gives Brownian motion

$f < 1$ gives "smooth" curve

$f > 1$ gives very rough surface

Could easily model a coastline using midpoint displacement

Similar algorithm for fractal surfaces

Use to build artificial landscapes, mountains, etc

Represent surfaces by a triangular mesh
At each level of recursive algorithm

Find midpoint of each side of triangle (a,b and c)
Add random value to each midpoint (get a', b' and c')
Form four new triangles: Ca'c', Ba'c', Ab'c', a'b'c'