Quadrics

Implicit surfaces of form

$$\begin{pmatrix} X & Y & Z & 1 \end{pmatrix} \begin{pmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = 0$$

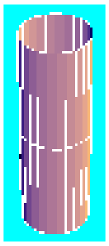
Or, can be expressed as

$$Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2$$
$$+2Fyz + 2Gy + Hz^2 + 2Iz + J = 0$$

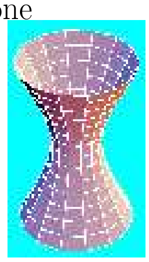
"Normalized" form:

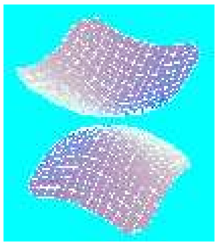
A, E, H, and one of { D, G, I } $\in 0, \pm 1$

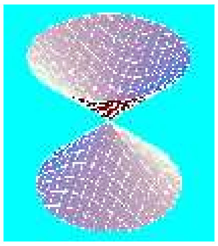
Ellipsoid (Sphere); Cylinder



Hyperboloid of One Sheet
Hyperboloid of Two Sheets
Cone







`http://amath.colorado.edu/appm/staff/fast/java/qg.html`

How to render?

In some cases, can write as $z = f(x, y)$

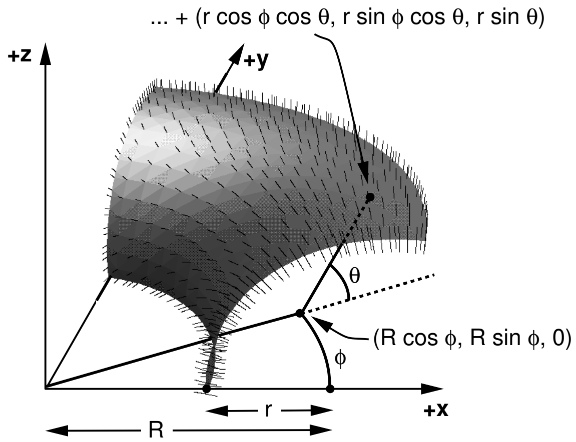
Or, use projection method as with spheres

Base object: cube, plane, plane pair, etc.

Parametrizing More Complex Objects

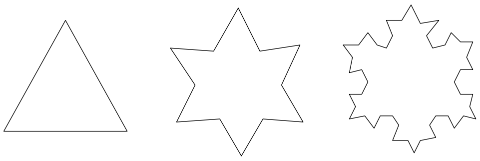
Example: Torus

Use major, minor radii R, r respectively



Fractals: Self-similar objects

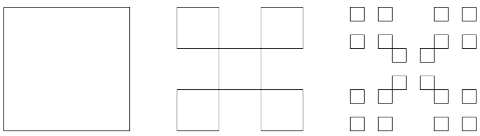
Recursive *construction rules* yield regular fractals



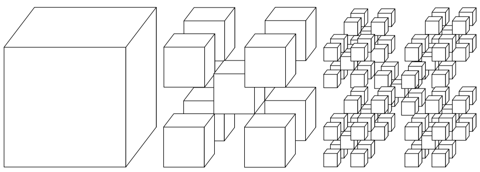
von Koch snowflake

(strictly speaking, convergent or limit curve)

Sub-objects need not be connected:

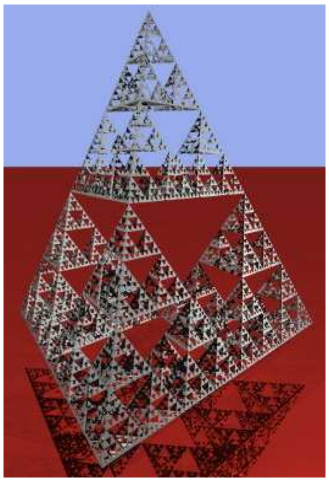


Three-D construction rules abound:



Fractal Objects

Sierpinski's Gadget:

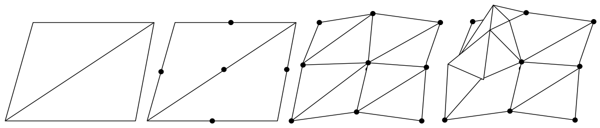


(Credit: Unreal image, rendered on Amiga)

What are its area, volume in limit?
Can make analogous objects from many shapes

Terrain: irregular fractal generation

Usually height fields, i.e., $z = f(x, y)$



Displacements can be *random*:

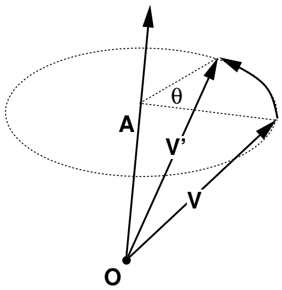
```
// returns pseudo-random numbers in [ 0 .. (2^15)-1 ]
extern int rand ( void );

// return random float in [low..high]
float randr ( float low, float high )
{
    return low + ( rand() / ((1<<15)-1.0) ) * (high-low);
}
```


Rotation about Arbitrary Axis

Rotate vector V about axis A by angle θ:
$\mathbf{V}_{\|}$, parallel to $\mathbf{A}$ is $\mathbf{A}(\mathbf{A} \cdot \mathbf{V})$
$\mathbf{V}_{\perp}$, perpendicular to $\mathbf{A}$ is $\mathbf{V} - \mathbf{V}_{\|}$
To form basis in plane of rotation, need $\mathbf{B} = \mathbf{A} \times \mathbf{V}$

Thus
$$\mathbf{V'} = \mathbf{A}(\mathbf{A}.\mathbf{V})[1 - \cos\theta] + \mathbf{V}\cos\theta + (\mathbf{A} \times \mathbf{V})\sin\theta$$



Matrix form left as exercise.

Rendering Pipeline

What actually happens between modeling & viewing?
Recall input, output of rendering process:

Input:

Geometric model:
Description of all object, surface, and light source geometry and transformations

Lighting model:
Computational description of object and light properties, interaction (reflection)

Synthetic Viewpoint: or *Camera:*
Eye position and viewing frustum

Raster Viewport:
Pixel grid onto which image plane is mapped

Output:

Colors/Intensities suitable for framebuffer display
(For example, 24-bit RGB value at each pixel)