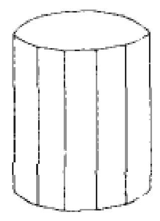
# 3D Object Representation (Polygon Surfaces: Planar)

## 4.3.1 Polygon Surfaces

Graphics scenes can contain trees, flowers , clouds rocks water, rubber, paper , bricks etc.
Polygon and quadratic surfaces provide precise descriptions for simple Euclidean objects such as polyhedrons ellipsoid.

### Polygon Surfaces

3-D graphics object is a set of surface polygons that enclose the object interior
A polygon mesh is a set of connected polygonally bounded planar surfaces
Equation of plane is $Ax + By + Cz + D = 0$
A polygon mesh is a collection of edges, vertices and polygons connected such
that each edge is shared by at most two polygons.

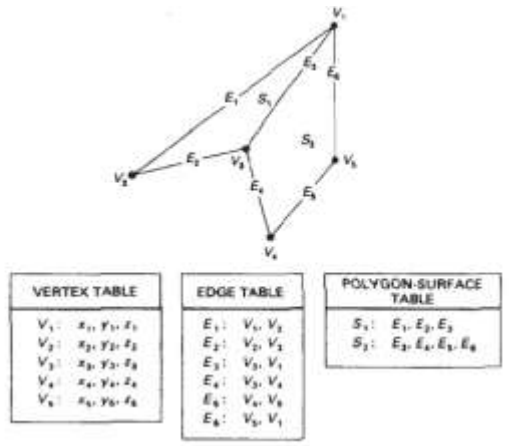


### Polygon Tables

Polygon data tables can be organized into two groups: geometrical and attribute tables
Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of polygon surfaces
Attribute information for an object includes parameters specifying the degree of transparency of object and its surface reflectivity and texture characteristics.

A convenient organization for storing geometric data is to create three lists:

i. a vertex table
ii. an edge table
iii.a polygon table



| VERTEX TABLE | |
|---|---|
| $V_1$: | $x_1, y_1, z_1$ |
| $V_2$: | $x_2, y_2, z_2$ |
| $V_3$: | $x_3, y_3, z_3$ |
| $V_4$: | $x_4, y_4, z_4$ |
| $V_5$: | $x_5, y_5, z_5$ |

| EDGE TABLE | |
|---|---|
| $E_1$: | $V_1, V_2$ |
| $E_2$: | $V_2, V_3$ |
| $E_3$: | $V_3, V_1$ |
| $E_4$: | $V_3, V_4$ |
| $E_5$: | $V_4, V_5$ |
| $E_6$: | $V_5, V_1$ |

| POLYGON-SURFACE TABLE | |
|---|---|
| $S_1$: | $E_1, E_2, E_3$ |
| $S_2$: | $E_3, E_4, E_5, E_6$ |

**Errors in Polygon Table (**Checking data for consistency and completeness)
Error checking is easier when three data tables (vertex, edge, and polygon) are used, since this scheme provides the most information.
Some of the tests that could be performed by a graphics package are that
(1) every vertex is listed as an endpoint for at least two edges,
(2) every edge is part of at least one polygon,
(3) every polygon is closed,
(4) each polygon has at least one shared edge, and
(5) if the edge table contains pointers to polygons, every edge referenced by a polygon pointer has a reciprocal pointer back to the polygon.

**Plane Equations**

The equation for a plane surface can be expressed as

$$Ax + By + Cz + D = 0 \quad \ldots\ldots.(i)$$

where, (x,y,z) is any point on the plane- coefficients ABCD are constants describing the spatial properties of the plane

For solving ABCD consider three successive polygon vertices $(x_1, y_1, z_1)$, $(x_2, y_2, z_2)$, $(x_3, y_3, z_3)$

So equation (i) is modified to,

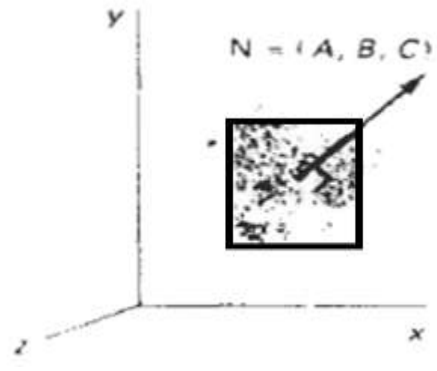
$$\frac{A}{D} x_k + \frac{B}{D} y_k + \frac{C}{D} z_k = -1 \quad \ldots\ldots.(ii) \qquad k = 1,2,3$$

Using Cramers rule,

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \qquad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding Determinants,

$$A = y_1 ( z_2 - z_3 ) + y_2 ( z_3 - z_1 ) + y_3 ( z_1 - z_2 )$$
$$B = z_1 ( x_2 - x_3 ) + z_2 ( x_3 - x_1 ) + z_3 ( x_1 - x_2 )$$
$$C = x_1 ( y_2 - y_3 ) + x_2 ( y_3 - y_1 ) + x_3 ( y_1 - y_2 )$$
$$D = -x_1 (y_2z_3 - y_3z_2 ) - x_2 (y_3z_1 - y_1z_3 ) - x_3 (y_1z_2 - y_2z_1 )$$



The vector N, normal to the surface of a plane described by equation Ax + By + Cz + D =0 has Cartesian Components (A,B,C)

Plane equations are used to identify the position of spatial points relative to the plane surfaces of an object. For any point (x,y,z) not on plane with parameters ABCD we have

$$Ax + By + Cz + D = 0$$

We can identify the point as either inside or outside the plane surface according to the sign( + or -) of

$$Ax + By + Cz + D$$

If $Ax + By + Cz + D < 0$ then the point (x,y,z) is inside the surface

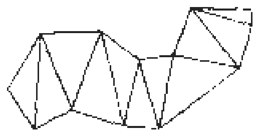
If $Ax + By + Cz + D > 0$ then the point (x,y,z) is outside the surface

**Polygon Meshes**

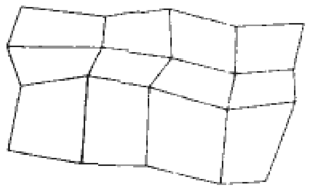
Some graphics packages (PHIGS programmer's Hierarchical Interactive Graphics Standard) provide several polygon functions for modeling objects.

One type of polygon mesh is the triangle strip

It produces n – 2 connected triangles, given the coordinates for n vertices.



Another similar functions the quadrilateral mesh that generates a mesh of (n-1)(m-1) quadrilaterals,
given the coordinates for an n by m array of vertices



When polygons are specified with more than 3 vertices, it is possible that the vertices may not all lie in one plane.

This can be due to numerical error or errors in selecting coordinate positions for the vertices

Remedies:

i. Simply divide the polygons into triangles
ii. Approximate the plane parameters ABC and project in same plane (i.e. approximate A to yz plane, B to xz plane, C to xy plane etc)

High quality graphics systems typically model objects with polygon meshes and set up a database of geometric and attribute information to facilitate processing of the polygon facets.

**Curved Surfaces and Lines**
Display of 3D curved lines and surfaces can be generated from an input set of mathematical functions (spheres, ellipsoid etc) defining the objects or from a set of user specified data points.

Spline representations are examples of generating curves and surfaces. These are commonly used to design new object shapes , to digitize drawings and to describe animation paths.

Curve fitting methods are also used to display graphs of data values by fitting specified curve functions to the discrete data set, using regression techniques such as the least square methods.

### 4.3.2 Cubic Spline and Beizer Curve

**Spline** : A spline is a flexible strip that passes thru a designated control points.

**Bezier Curve**

A spline approximation method proposed by Pierre Bezier

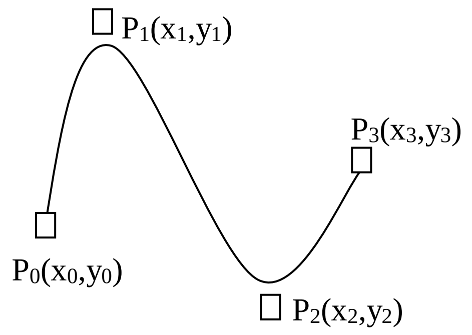
Used for the design of Renault automobiles bodies

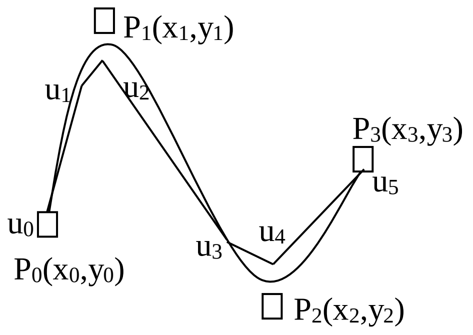
The number of control points to be approximated and their relative position determine the degree of the Bezier polynomial. So, the degree of the curve = number of control points - 1

If the number of control points used is two we have a linear curve (Straight line)

If the number of control points used is three we have a quadratic curve (Parabolic)

If the number of control points used is four we have a cubic curve





A smooth curve is approximated with a large number of very small line segments. The above figure shows an approximation of the curve with five line segments only.

The approach below is used to draw a curve for any number of control points

Suppose $P_0, P_1, P_2, P_3$ are four control points

Number of line segments used to approximate the curve is nSeg

Let the variable 'i' be incremented linearly from 0 to nSeg i.e. i = 0 to nSeg

Let $u = i / nSeg$ [then its values varies from 0 to 1, $0 <= u <= 1$]

Or $u_0 = 0/nSeg = 0$, $u_1 = 1/nSeg$, $u_2 = 2/nSeg$, $u_3 = 3/nSeg$, $u_4 = 4/nSeg$ …….. $u_{nSeg} = nSeg/nSeg = 1$

i.e. if five line segments are used to approximate a curve

$u_0 = 0/5 = 0$, $u_1 = 1/5 = 0.2$, $u_2 = 2/5=0.4$, $u_3 = 3/5=0.6$, $u_4 = 4/5 = 0.8$ , $u_5 = 5/5 = 1$

The individual curve coordinates of the position vectors that describes the path of an approximating Bezier Polynomial function between $P_0$ and $P_n$ can be expressed as

$x(u) = \sum_{j=0}^{n} x_j \, BEZ_{j,n}(u)$ n : number of control points

$x(u) = x_0 BEZ_{0,3}(u) + x_1 BEZ_{1,3}(u) + x_2 BEZ_{2,3}(u) + x_3 BEZ_{3,3}(u)$

similarly

$y(u) = \sum_{j=0}^{n} y_j \, BEZ_{j,n}(u)$ n : number of control points

$y(u) = y_0 BEZ_{0,3}(u) + y_1 BEZ_{1,3}(u) + y_2 BEZ_{2,3}(u) + y_3 BEZ_{3,3}(u)$

The Bezier blending function BEZj,n (u) is defined as,

$$BEZ_{j,n}(u) = \frac{n!}{j!\,(n-j)!} u^j (1-u)^{n-j}$$

$$BEZ_{j,n}(u) = C_{(n,j)} \, u^j (1-u)^{n-j}$$

Where $C_{(n,j)}$ is the Binomial Coefficient

$C_{(n,j)} = \dfrac{n!}{\phantom{x}}$

$j!\,(n-j)!$

For each 'u' the coordinates x and y are computed and desired curve is produced when the adjacent coordinates (x,y) are connected with a straight line segment

Now the position vector P(u) can be expressed as

$P(u) = P_0\, BEZ_{0,3}(u) + P_1\, BEZ_{1,3}(u) + P_2\, BEZ_{2,3}(u) + P_3\, BEZ_{3,3}(u)$

Four blending functions must be found based on Bernstein Polynomials

$BEZ_{0,3}(u) = \dfrac{3!}{0!\ 3!}\, u^0 (1-u)^3 = (1-u)^3 \qquad BEZ_{1,3}(u) = \dfrac{3!}{1!\ 2!}\, u^1 (1-u)^2 = 3u\,(1-u)^2$

$BEZ_{2,3}(u) = \dfrac{3!}{2!\ 1!}\, u^2 (1-u) = 3u^2 (1-u) \qquad BEZ_{3,3}(u) = \dfrac{3!}{3!\ 0!}\, u^3 (1-u)^0 = u^3$

Normalizing properties apply to blending function s that means thy all add up to one

Substituting these functions in above equation

$P(u) = (1-u)^3 P_0 + 3u\,(1-u)^2 P_1 + 3u^2 (1-u) P_2 + u^3 P_3$

When u = 0 then $P(u) = P_0$ and when u =1 then $P(u) = P_3$

in Matrix Form

$$P(u) = \left[(1-u)^3 \quad 3u\,(1-u)^2 \quad 3u^2 (1-u) \quad u^3\right] \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

or

$$P(u) = \left[(1-3u+3u^2-u^3) \quad (3u-6u^2+3u^3) \quad (3u^2-3u^3) \quad u^3\right] \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

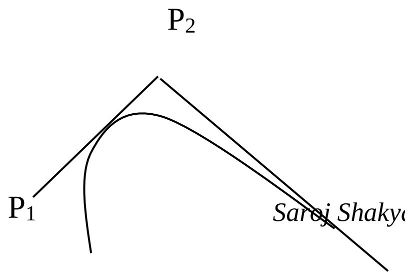
or

$$P(u) = \left[u^3 \ \ u^2 \ \ u^1 \ \ 1\right] \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

**Properties of a Bezier Curve**

1. Bezier curve lies within the convex hull of the control points which ensure that the curve smoothly follows the

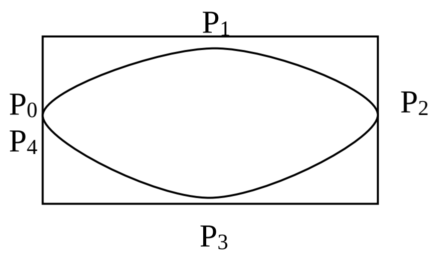
control Points



2. Four Bezier polynomials are used in the construction of curve to fit four control points

3. It always passes thru the first and the last control points

4. Closed curves can be generated by specifying the first and last control points at the same position



5. Specifying multiple control points at a single position gives more weight to that position

6. Complicated curves are formed by piecing several sections of lower degrees together

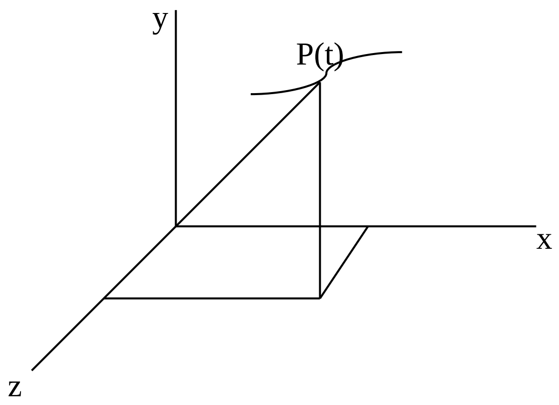
7. The tangent to the curve at an end point is along the line joining the end point to the adjacent control point

8. Two pieces of Bezier curve segments can be joined if the first control point of the second curve matches the last control point of the first curve

**Parametric Cubic Curve**

A parametric cubic curve is defined as $P(t) = \sum_{i=0}^{3} a_i t^i \qquad 0 <= t <= 1$ ------ (i)

Where, $P(t)$ is a point on the curve



Expanding equation (i) yields

$P(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ ------------------------------(ii)

This equation is separated into three components of P(t)

$x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x}$

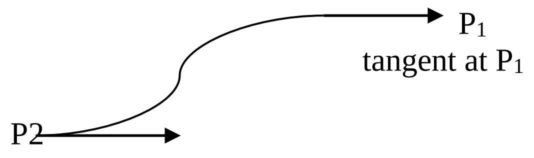
$y(t) = a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_{0y}$

$z(t) = a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z}$ -----------------------------(iii)

To be able to solve (iii) the twelve unknown coefficients $a_{ij}$ (algebraic coefficients) must be specified

From the known end point coordinates of each segment, six of the twelve needed equations are obtained. The other six are found by using tangent vectors at the two ends of each segment

The direction of the tangent vectors establishes the slopes(direction cosines) of the curve at the end points



This procedure for defining a cubic curve using end points and tangent vector is one form of *hermite* interpolation

Each cubic curve segment is parameterized from 0 to 1 so that known end points correspond to the limit values of the parametric variable *t*, that is P(0) and P(1)

Substituting *t* = 0 and *t* = 1 the relation ship between two end point vectors and the algebraic coefficients are found

$P(0) = a_0$ $\qquad$ $P(1) = a_3 + a_2 + a_1 + a_0$

To find the tangent vectors equation ii must be differentiated with respect to t

$P'(t) = 3a_3t^2 + 2a_2t + a_1$

The tangent vectors at the two end points are found by substituting t = 0 and t = 1 in this equation

$P'(0) = a_1$ $\qquad$ $P'(1) = 3a_3 + 2a_2 + a_1$

The algebraic coefficients '$a_i$' in equation (ii) can now be written explicitly in terms of boundary conditions – endpoints and tangent vectors are

$a_0 = P(0)$ $\qquad$ $a_1 = P'(0)$

$a_2 = -3P(0) + 3P(1) - 2P'(0) - P'(1)$ $\qquad$ $a3 = 2P(0) - 2P(1) + P'(0) + P'(1)$

substituting these values of '$a_i$' in equation (ii) and rearranging the terms yields

$P(t) = (2t^3 - 3t^2 + 1) P(0) + (-2t^3 + 3t^2) P(1) + (t^3 - 2t^2 + t) P'(0) + (t^3 - t^2) P'(1)$

The values of P(0), P(1), P'(0), P'(1) are called *geometric coefficients* and represent the known vector quantities in the above equation

The polynomial coefficients of these vector quantities are commonly known as *blending functions*

By varying parameter t in these blending function from 0 to 1 several points on curve segments can be found

## 4.3.3 Non-Planer Surface: Bezier Surface

**Bezier Surfaces**

Two sets of orthogonal Bezier curves can be used to design an object surface be specifying by an input mesh of control points. The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions:

Bezier surfaces are defined by simple generalization of the curve formulation. Here, tensor product approach is used with two directions of parameterization 'u' and 'v'.

Any point on the surface can be located to given values of parametric pair by

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} P_{j,k} BEZ_{j,m}(u) BEZ_{k,n}(v) \qquad 0 \le u,v \le 1$$

As in the case of Bezier curves the $P_{j,k}$ define the control vertices and the $BEZ_{j,m}(u)$ and $B_{k,n}$`(v) are the Bernstein blending functions in the u and v directions

The Bézier functions specify the weighting of a particular knot. They are the Bernstein coefficients. The definition of the Bézier functions is

$BEZ_{j,m}(u) = C(m,j)\, u^j (1-u)^{m-j}$

$BEZ_{k,n}(v) = C(n,k)\, v^k (1-v)^{n-k}$

where C(m,j) represents the binomial coefficients.

$$C(m,j) = \frac{m!}{j!\,(m-j)!} \qquad C(n,k) = \frac{n!}{k!\,(n-k)!}$$

where C(n,k) represents the binomial coefficients.

When t = 0, the function is one for j = 0 and zero for all other points.

When we combine two orthogonal parameters, we find a Bézier curve along each edge of the surface, as defined by the points along that edge.

Bézier surfaces are useful for interactive design and were first applied to car body design.

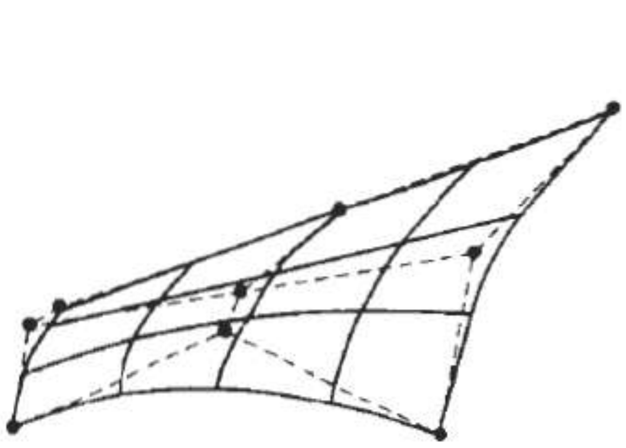
The degree of blending functions does not have to be the same in two parametric directions it could be cubic in 'u' and quadratic in 'v'

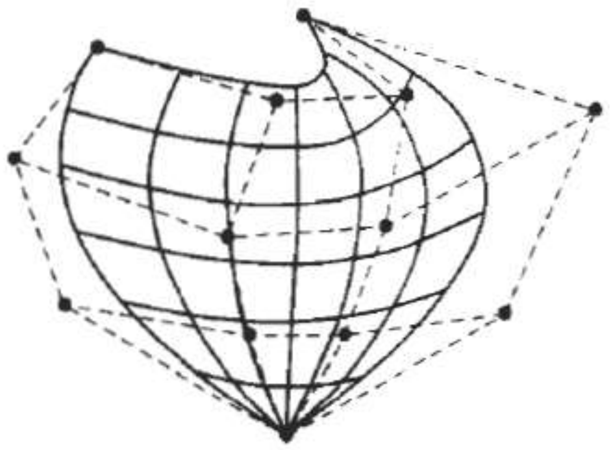
The properties of Bezier surfaces are controlled by the blending functions

- The surface takes the general shape of the control points
- The surface is contained within the convex hull of the control points
- The corners of the surface and the corner control vertices are coincident

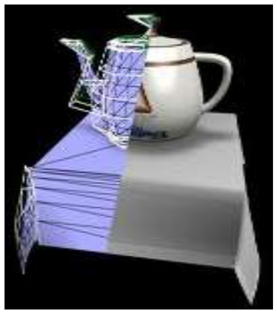
Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points.

The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions with $P_{j,k}$ specifying the location of the ( m + 1) by ( n + 1) control points.





Bezier surfaces constructed for m = 3 n = 3 and m = 4 n = 4
Dashed lines connect the control points



### 4.3.4 Fractal Geometry Method

Natural objects, such as mountains and clouds, don't have smooth surface or regular shapes instead they have fragmented features, and Euclidean methods do not realistically model these objects.

Natural objects can be realistically described with fractal-geometry methods, where procedures rather than equations are used to model objects.

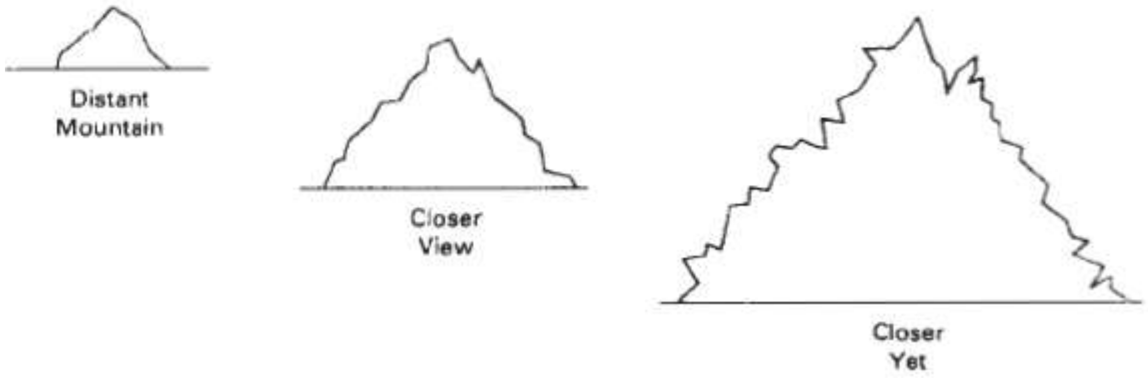
In computer graphics, fractal methods are used to generate displays of natural objects and visualizations of various mathematical and physical systems.

A fractal object has two basic characteristics:

i. **Infinite detail** at every point
ii. Certain **self-similarity** between the object parts and the overall features of the object

A fractal object is described with a procedure that specifies a repeated operation for producing the detail in the object subparts. Natural objects are represented with procedures that theoretically repeat an infinite number of times. Graphics displays of natural objects are, of course, generated with a finite number of steps.

If we zoom in on a continuous Euclidean shape, no matter how complicated, we can eventually get the zoomed-in view to smooth out.
But if we zoom in on a fractal object, we continue to see as much detail in the magnification as we did in the original view.



Zooming in on a graphics display of a fractal object is obtained by selecting a smaller window and **repeating the fractal procedures** to generate the detail in the new window.
A consequence of the infinite detail of a fractal object is that it has no definite size. As we consider more and more detail, the size of an object tends to infinity, but the coordinate extents of the object remain bound within a finite region of space.
The amount of variation in the object detail with a number called the ***fractal dimension.***
In graphics applications, fractal representations are used to model terrain, clouds, water, trees and other plants, feathers, fur, and various surface textures, and just to make pretty patterns.

**Fractal-Generation Procedures**
A fractal object is generated by **repeatedly applying a specified transformation function** to points within a region of space.
If $\mathbf{P_0}$ = (x0, y0, z0) is a selected initial point, each iteration of a transformation function F generates successive levels of detail with the calculations
In general, the transformation function can be applied to a specified point set, or we could apply the transformation function to an initial **set** of primitives, such as straight lines, curves, color areas, surfaces, and solid objects.
Also, we can use either deterministic or random generation procedures at each iteration. The transformation function may be defined in terms of geometric transformations (scaling, translation, rotation), or it can be set up with nonlinear coordinate transformations and decision parameters.
Although fractal objects, by definition, contain infinite detail, we apply the transformation function a finite number of times. Therefore, the objects we display actually have finite dimensions.
A procedural representation approaches a "true" fractal as the number of transformations is increased to produce more and more detail.
The **amount of detail** included in the final graphical display of an object **depends on the number of iterations performed** and the resolution of the display system
We cannot display detail variations that are smaller than the size of a pixel.
To **see more** of the object detail, we **zoom in on selected sections** and **repeat the transformation function** iterations.
**Classification of Fractals**
**i. Self-similar fractals**
Self-similar fractals have parts that are scaled-down versions of the entire object.
Starting with an initial shape, the object subparts are constructed by apply a scaling parameter ‘s’ to the overall shape.
The same scaling factors can be used for all subparts, or different scaling factors can be used for different scaled-down parts of the object.

If random variations are applied to the scaled-down subparts, the fractal is said to be statistically self-similar. The parts then have the same statistical properties.
Statistically self-similar fractals are commonly used to model trees, shrubs, and other plants.

**ii. Self-affine fractals**

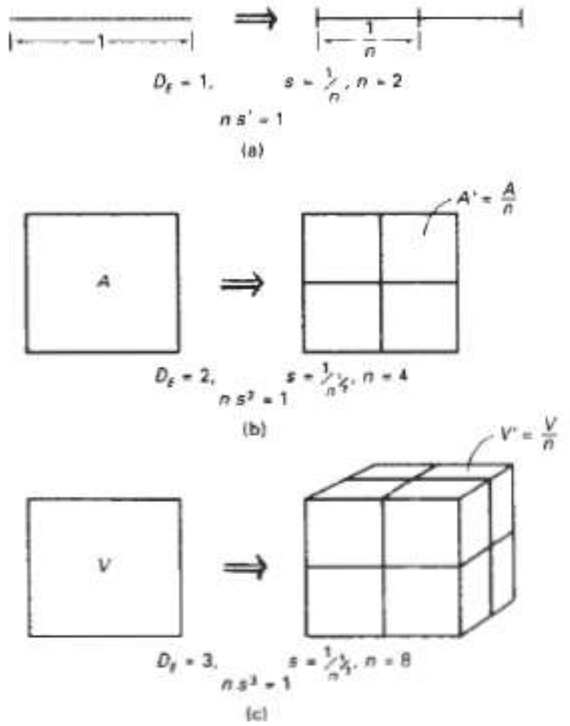
Self-affine fractals have parts that are formed with different scaling parameters, $\mathbf{s_x, s_y, s_z}$ in different coordinate directions.
Random variations can also be used to obtain statistically self-affine fractals.
Terrain, water, and clouds are typically modeled with statistically self-affine fractal construction methods.

**iii. Invariant fractal**

Invariant fractal sets are formed with nonlinear transformations.
This class of fractals includes self-squaring fractals, such as the Mandelbrot set, which are formed with squaring functions in complex space; and self-inverse fractals, formed with inversion procedures.

**Fractal Dimension**

The detail variation in a fractal object can be described with a number D, called the **fractal dimension**, which is a measure of the roughness, or fragmentation, of the object.
More jagged-looking objects have larger fractal dimensions.
Iterative procedures can be set up to generate fractal objects using a given value for the fractal dimension **D.**
An expression for the fractal dimension of a self-similar fractal, constructed with a single scalar factor **s**, is obtained by analogy with the subdivision of a Euclidean object.
Suppose an object is composed of clay or elastic. If it is deformed into a line then its topological dimension $D_t$ is 1, if it is deformed into a plane or a disk then the topological Dimension is 2 and if it is deformed into a ball or a cube then its topological dimension is 3



The relationships between the scaling factor s; and the number of subparts n for subdivision of a unit straight-line segment, A square, and a cube can be shown
If take a line segment having length L and divide it into n pieces each piece having length '*l*'
The scaling factor s = 1/n
If it is broken into two pieces, $s^1 = 1/2$, the unit line segment is divided into two equal-length subparts.
Similarly, the square is divided into four equal-area subparts, $s^2 = 1/4$
The cube is divided into eight equal-volume subparts $s^3 = 1/8$
For each of these objects, the relationship between the number of subparts and the scaling factor is $n \cdot s^D = 1$. In analogy with Euclidean objects, the fractal dimension D for self-similar objects can be obtained from

$$n \cdot s^D = 1.$$

Solving this expression for D, the fractal similarity dimension, we have

$$D = (\log n) / (\log (1/s))$$

The fractal dimension gives the measure of the roughness or fragmentation of objects and is always greater than the corresponding topological dimension

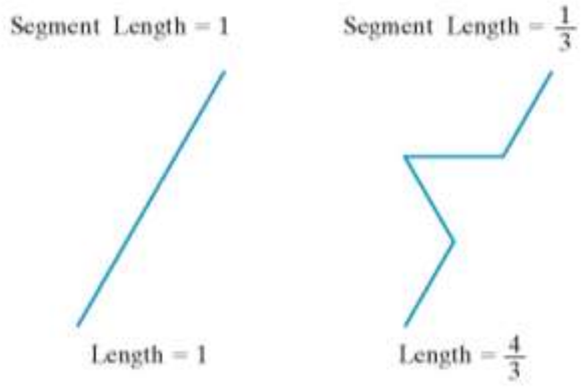
For a self-similar fractal constructed with different scaling factors for the different parts, the fractal similarity dimension is obtained from the implicit relationship where $\mathbf{s_k}$ is the scaling factor for subpart number k.

**Geometric Construction of Deterministic Self-Similar Fractals**

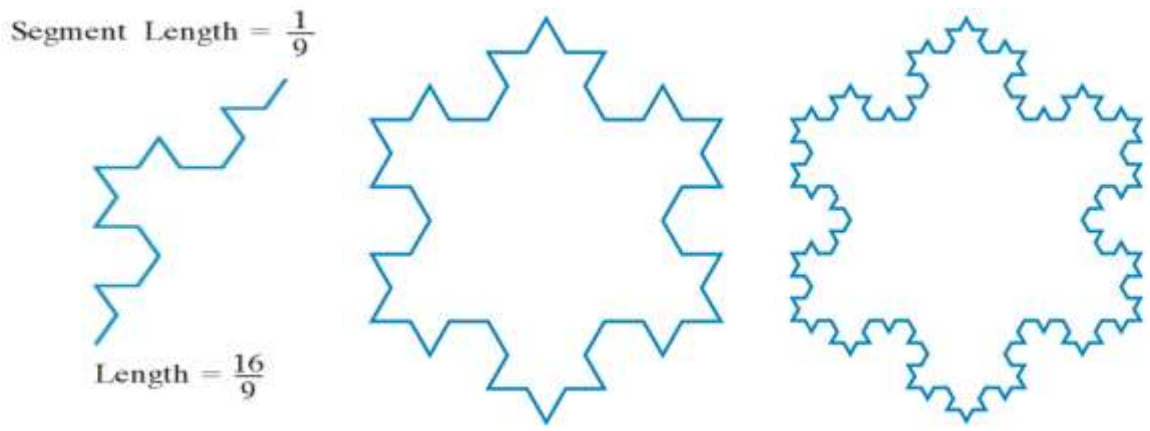
To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the ***initiator.*** Subparts of the initiator are then replaced with a pattern, called the generator.

**Koch Curve**

1. Begin with a line segment
2. Divide it into thirds i.e. scaling factor = 1/3 and replace the center third by the two adjacent sides of an equilateral triangle
3. There are now 4 equal length segments each 1/3 the original length , so the new curve has 4/3 length of the original length



5. Repeat the process for each of the four segments
6. The cure has gained more wiggles and its length now is 16/9 times the original



7. Repeat this indefinitely and the length every time increases by 4/3 factor. The curve will be infinite but is folded in lots of tiny wiggles
8. Its topological dimension is 1 and it's Fractal dimension can be calculated as follows

We have to assemble 4 such curves to make the original curve so N = 4 and Scaling factor S = 3 as each segment has 1/3 the original segment length

So the fractal dimension is $D = (\log 4) / (\log (3)) = 1.2618$

**Peano Curve**

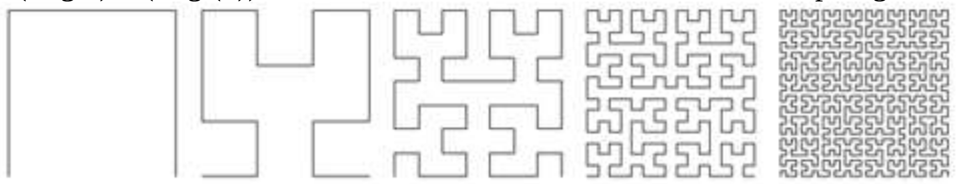
It is also called space filling curve and is used for filling two dimensional object e.g. a square

**Steps to generate a Peano curve**

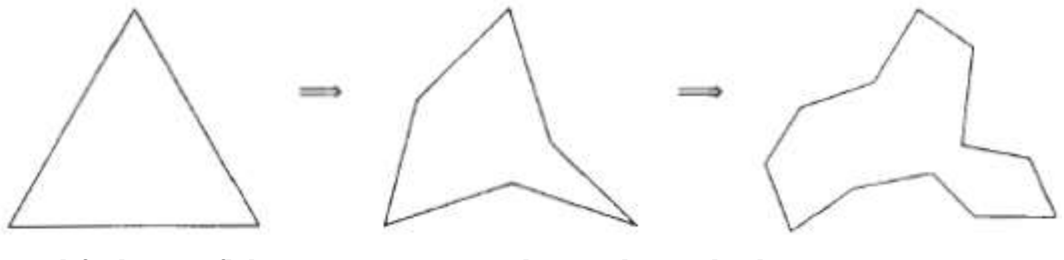
1. Sub-divide a square into 4 quadrants and draw the curve which connects the center points of each
2. Further subdivide each of the quadrants and connect the centers of each of these finer divisions before moving to the next major quadrant
3. The third approximation subdivides again it again connects the centers of the finest level before stepping to the next level of detail
The above process is indefinitely continued depending upon the degree of roughness of the curve generated

- The curve never crosses itself
- There is no limit to the subdivision
- The curve fills the square
- With each subdivision the length increases by a factor of 4 and since there is no limit to subdivision there is no limit to the length
- The curve constructed is topologically equivalent to the line $D_t = 1$ but it is so twisted and folded that it exactly fills up a square
- The Fractal dimension of the curve:

  At each subdivision the scale changes by 2 but length changes by 4
  For the square it takes 4 curves of half scale to build the full sized object so the Dimension is given by
  $D = (\log 4) / (\log (2))$ So, the Fractal Dimension is 2 and the Topological Dimension is 1



**Geometric Construction of Statistically Self-Similar Fractals**
One way to introduce some randomness into the geometric construction of a self-similar fractal is to choose a generator randomly at each step from a set of predefined shapes. Another way to generate random self-similar objects is to compute coordinate displacements randomly.
A random snowflake pattern can be created by selecting a random, midpoint displacement distance at each step.



***A*** modified "snowflake" pattern using random midpoint displacement.

**Affine Fractal-Construction Methods**
Highly realistic representations for terrain and other natural objects can be obtained using affine fractal methods that model object features as ***fractional Brownian motion.***
This is an extension of standard Brownian motion, a form of "random walk**",** that **describes** the erratic, zigzag movement of particles in a gas or otter fluid.

Starting from a given position, we generate a straight-line segment in a random direction and with a random length. We then move to the endpoint of the first line segment