UNIT 1 CURVES AND SURFACES

1.1 INTRODUCTION

In CS-60, Block 2 and 4 we have studied the technique of drawing curves in different coordinate systems. Also we got the idea that it is the revolution of a curve about some axis that gives rise to an object enclosing some volume and area. For better understanding, just think how a potter works to create vessels of different shapes. He just put a lump of wet soil on the disc which is revolving about its axis at high speed, then his/her fingers and palm works as a curve, in contact with the wet soil. Thus, it is the curve which revolves the some axis to produce vessels of the shape s/he desires. In this unit let us study some of the practical implementations of the concepts studied in CS-60 to computer graphics. This will help a lot because in nature God has created everything with a level of fineness, that if human beings try to achieve that level in their created art (whether computer generated or not) such that it is quite close to reality, then one has to excessively make use of curves and surfaces in a balanced format, and the balance is provided by mathematics. So with the edge of mathematics computer graphics we can achieve realism. In this unit, we will study the polygon representation methods – these methods are quite important because it's the polygon that constitutes every closed object like tree, clouds, ball, car, etc., to be represented through graphics. Further, each polygon has its own mathematical equation which works as the generating function for that polygon. Under this topic we will discuss polygon tables, polygon meshes and equation of plane. A study of these topics will provide you a computer oriented approach to understand the implementation of mathematical concepts. We are going to discuss one more important topic in this unit, which is Bezier Curves and their properties. It's the Bezier curves which have revolutionised the field of computer graphics and opened a new arena, i.e., automobile sector, for analysis and designing of automobiles. Inspired with this achievement, scientists have worked hard and now there is no area which is complete without computer graphics and animation. In this unit, we will deal with fitting curves to the digitized data. Two techniques are available for obtaining such curves *cubic spline and parabolicaly blended curves.* These are based on curve fitting techniques. That is, they are not an approximate method but fit the curve point exactly. We will also discuss curve fairing techniques like Bezier and B spline curves, used to approximate the curve when you are not having a proper knowledge of the shape of the curve. Last but not the least, we will discuss the concept of surface of revolution. It is an important topic because the products which are designed are in fact the surfaces enclosed by the revolution of the curve about some axis. So let us begin our journey.

1.2 OBJECTIVES

After going through the unit, you should be able to:

- implement the methods used to represent a polygon;
- deduce equation of plane and explain the need of planes in graphics;
- discuss various curves and surface representation schemes;
- discuss the piecewise cubic polynomial equation and their need in object representation;
- describe Bezier curve/surface and their properties, and
- discuss the concept of surface of revolution.

1.3 POLYGON REPRESENTATION METHODS

Any scene to be created through computer graphics may contain a variety of objects, some of them natural and some manmade. Thus, to achieve realism in our scene we are not having any specific single method which handles the realistic representational complexities of all components (whether natural or manmade) in a scene.

So let us have an idea of the basic representational schemes available:

- Polygon and quadric surfaces provide precise description for simple Euclidean objects like polyhedrons, ellipsoids.
- Spline surfaces and construction techniques are useful for designing aircraft wings, gears, and other engineering structures with curved surfaces
- Procedural methods such as fractals constructions and particle systems give us accurate representations for the natural objects like clouds, trees etc.
- Physically based modeling methods using systems for interacting forces can be used to describe the non-rigid behaviors of piece of jelly, or a piece of cloth.
- Octrees encoding are used to represent internal features of the objects, such as those obtained from medical CT images, volume renderings and other visualization techniques.

The representational schemes of solid objects are divided into two broad categories:

- **Boundary representations:** Here the 3D object is represented as a set of surfaces that separate the object interior from the environment. Examples are polygonal facets and spline patches. For better understanding consider *Figure 1*.

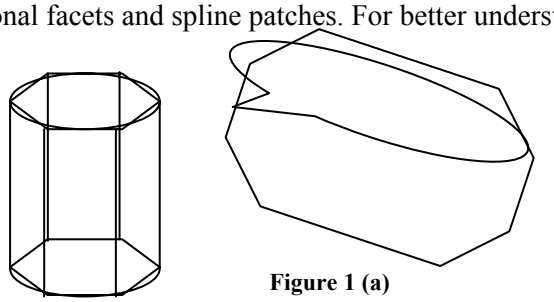


Figure 1 (a)

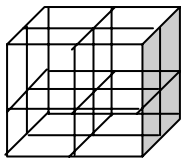


Figure 1 (b)

- **Space partitioning representations:** These are used to describe the interior properties, by partitioning the spatial regions containing an object into a set of small non-overlapping contiguous solids (usually cubes). Example: Octree (which is the space partitioning description of 3D object). For a better understanding consider *Figure 2.*

.

Out of the various representational techniques mentioned above, the most commonly used boundary representation mechanism for representing 3D objects is,

using a set of polygon surfaces to enclose the object interior. Let us discuss this mechanism in our next section.

1.3.1 Polygon Surfaces

From *Figure 1* and *Figure 2* it is quite clear that it is possible to store object's description as a set of surface polygons and the same is actually done by many graphic systems, actually this way of object description fastens the rendering and display of object surfaces. The approach is beneficial because all the surfaces can now be described with linear equations and hence polygonal description of the surfaces is referred as "standard graphic objects". Very many times it is the polygonal representation which is available to describe the object but there are other schemes like spline surfaces which are converted to polygonal representation for processing.

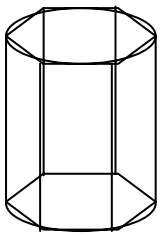


Figure 2

Consider *Figure 3* where the cylindrical surface is represented as a mesh of polygons. The representation is known as wire frame representation which can quickly describe the surface structure. On the basis of the structure a realistic rendering can be performed by interpolating the shading patterns across the polygon surfaces. Thus polygon mesh representation of the curved surface is actually dividing a curved surface into polygon facets. This improves and simplifies the process of rendering (transforming a 3D scene to 2D scene with least loss of information like height, depth etc). Now the objects are composed of standard graphic objects (polygon surfaces). Each graphic object needs some method for its description which could be a polygon table or equation etc. So, let us study the procedures to represent a polygon surface.

1.3.2 Polygon Tables

Every polygon is analogous to a graph G(V,E). We have studied graphs and their theory in detail in MCS-033. Keeping in mind the analogy we can say that a polygon surface can be specified with as a set of vertex coordinates and associated attribute parameters (the attributes may be colour, contrast, shading, etc). Most systems use tables to store the information entered for each polygon, and it's the data in these tables which is then used for subsequent processing, display and manipulation of the objects in the scene. As the data required to be stored for an object in the scene involves both the geometric information and attributes associated with the object, so polygon data tables can be organised into two groups:

- **Attribute tables:** This table holds object information like transparency, surface reflexivity, texture characteristics, etc., of an object in the scene.

- **Geometric tables:** This table stores the information of vertex coordinates and parameters like slope for each edge, etc. To identify the spatial orientation of polygon surface.

In order to store geometric information of a polygon properly, this table is further bifurcated into three more tables:

a) **Vertex table:** Holds coordinate values of vertices in the object.

b) **Edge table:** Holds pointers back in to the vertex table for identification of the vertices related to each polygon edge.

c) **Polygon table or polygon surface table:** Holds pointers back into the edge table for identification of the edges related to the polygon surface under construction.

This tabular representation of a polygon surface is shown in *Figure 4*. Such representations helps one to quickly refer to the data related to a polygon surface. Also, when the data is put for processing then the processing can be quite efficient, leading to efficient display of the object under consideration.

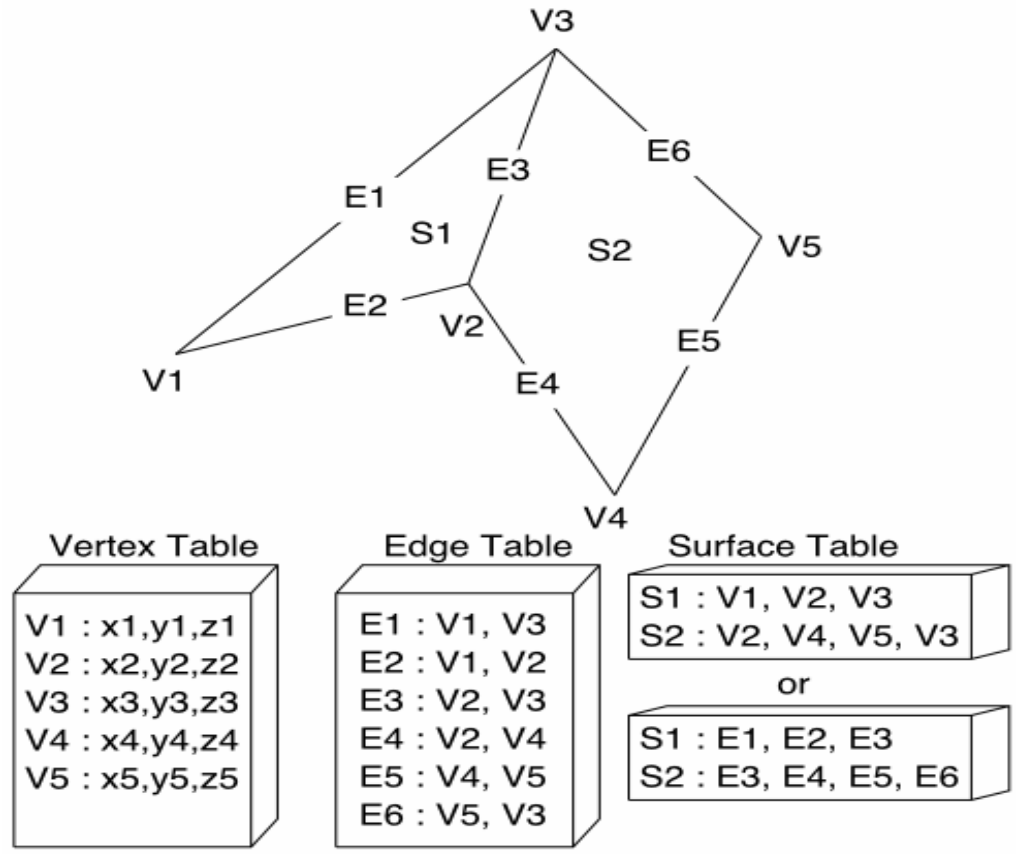


Figure 3

Some basic tests that should be performed before producing a polygon surface by any graphic package:

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,
5) if the edge table contains pointer to polygons, every edge referenced by a polygon pointer to polygon, every edge referenced by a polygon pointer has a reciprocal pointer back to polygon.

Example 1: Set up a geometric data table for an 3d rectangle.

Solution:

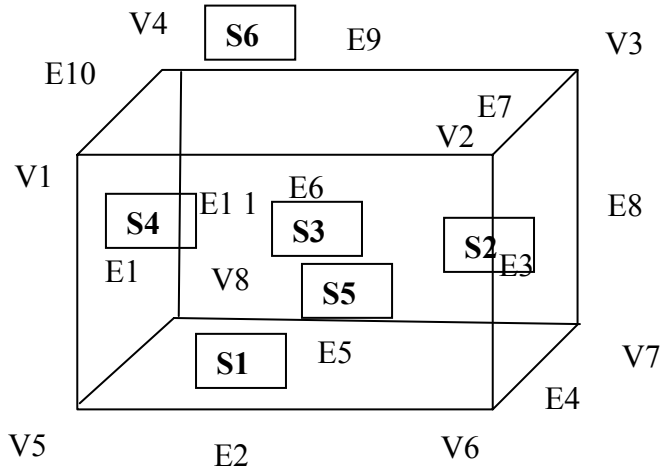


Figure 4

Vertex Table		**Edge Table**		**Polygon Surface Table**	
V1	X1,y1,z1	E1	V1,v5	S1	V1,v2,v6,v5
V2	X2, y2, z2	E2	V5,v6	S2	V2,v6,v7,v3
V3	X3, y3, z3	E3	V6,v2	S3	V8,v7,v3,v4
V4	X4, y4, z4	E4	V6,v7	S4	V1,v4,v5,v8
V5	X5, y5, z5	E5	V7,v8	S5	V8,v5,v6,v7
V6	X6, y6, z6	E6	V1,v2	S6	V4,v1,v2,v3
V7	X7,y7,z7	E7	V2,v3		
V8	X8,y8,z8	E8	V7,v3		
		E9	V3,v4		
		E10	V4,v1		
		E11	V1,v2		

☞Check Your Progress 1

1) What do you think about the utility of polygon surface table? Can't we do the implementation of a polygon surface with just a vertex table and an edge table?

……

……

……

……

2) What happens if we expand the edge table such that it also stores the forward pointers into the polygon table?

……

……

……

3) Can we extend the vertex table? Give reasons in support of your answer.

……

……

……

4) What do you think expanding the tables will make error detection easy or difficult?

……

……

……

5) Set up a geometric data table for a 3d rectangle using only vertex and polygon tables.

……

……

……

1.3.3 Plane Equation

Plane is a polygonal surface, which bisects its environment into two halves. One is referred to as forward and the other as backward half of any plane. Now the question is, which half is forward and which backward, because both are relative terms. So to remove this dilemma, we use the mathematical representation of planes, i.e., concepts like equations of planes, normal to a plane, etc., which we have already studied in CS-60. Now we have understood that both forward and backward halves are relative terms but w.r.t what? Yes, it's the plane itself in respect of which we can say any point in the surrounding environment is in front or back of the plane. So we consider any point on the plane should satisfy the equation of a plane to be zero, i.e., $Ax + By + Cz + D=0$. This equation means any point (x,y,z) will only lie on the plane if it satisfies the equation to be zero any point (x,y,z) will lie on the front of the plane if it satisfies the equation to be greater than zero, and any point (x,y,z) will lie on the back of the plane if it satisfies the equation to be less than zero.

Where (x, y, z) is any point on the plane, and the coefficients A, B, C and D are constants describing the spatial properties of the plane? This concept of space partitioning is used frequently in method of BSP (Binary Space Partitioning) trees generation a polygon representation scheme, quite similar to Octrees.

The importance of plane equations is that they help in producing display of any 3 D object, But for that we need to process the input data representation for the object through several procedures, which may include the following steps of processing.

- To transform the modeling and world-coordinate descriptions to viewing coordinates,
- To devise coordinates,
- To identify visible surfaces,
- To apply surface-rendering procedures.

For some of these processes, we need information about the spatial orientation of the individual surface components of the object. This information is obtained from the vertex coordinates values and the equations that describe the polygon planes. Let us

study how we can determine the equation of any plane. The equation of a plane, say ax + by + cz + d = 0, can be determined (generally) in 2 ways:

(1) Say we are given a point P_0 (x_0, y_0, z_0) which lies on the plane and say $\overline{N_i}$ be the . normal to that plane, but we are not given the equation of plane. Then the straight forward procedure to find the equation of plane is:

- choose any other point P (x, y, z) on plane
- determine the line joining point P and P_0 i.e., $\overline{P_0P} = (x - x_0, y - y_0, z - z_0)$
- take dot product of the line $\overline{P_0P}$ and normal to the plane i.e., $\overline{N}$

As Normal is perpendicular to any line on the plane so the result of the dot product should be zero. Therefore,

$$\begin{aligned}\overline{P_0P} \cdot \overline{N} &= (x - x_0, y - y_0, z - z_0).\ (n_1, n_2, n_3) = 0 \\ &= (x - x_0)\, n_1 + (y - y_0)\, n_2 + (z - z_0)\, n_3 = 0 \\ &= n_1x + n_2y + n_3z = (n_1x_0 + n_2y_0 + n_3z_0)\end{aligned}$$

$\therefore$ equation of plane is: $n_1x + n_2y + n_3z - (n_1x_0 + n_2y_0 + n_3z_0) = 0.$

(2) In the equation $Ax + By + Cz + D = 0$ for a plane surface ,where (x, y, z) is any point on the plane, and the coefficients A, B, C and D are constants describing the spatial properties of the plane. If the values of A,B,C, and D are not given then we can obtain the values of A, B, C and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane, which are say(x_1,y_1,z_1) , (x_2,y_2,z_2) , (x_3,y_3,z_3)

For this purpose, we can select the following set of simultaneous linear plane equations for the ratios A/D, B/D, and C/D:

Then

$$\left.\begin{aligned} &Ax_1 + By_1 + Cz_1 + D = 0 \\ &Ax_2 + By_2 + Cz_2 + D = 0 \\ &Ax_3 + By_3 + Cz_3 + D = 0 \\ &(A/D)x_k + (B/D)y_k + (C/D)z_k = -1, \qquad k = 1, 2, 3 \end{aligned}\right\} \quad (1)$$

The solution for this set of equations can be obtained in determinant form, using Cramer's rule, as

$$\left.\begin{aligned} A &= \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \\ C &= \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = -1 \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \end{aligned}\right\} \quad (2)$$

Expanding the determinants, we can write the calculations for the plane coefficients in the form

$$\left.\begin{aligned} A &= y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2) \\ B &= z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2) \\ C &= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \\ D &= -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1) \end{aligned}\right\} \quad (3)$$

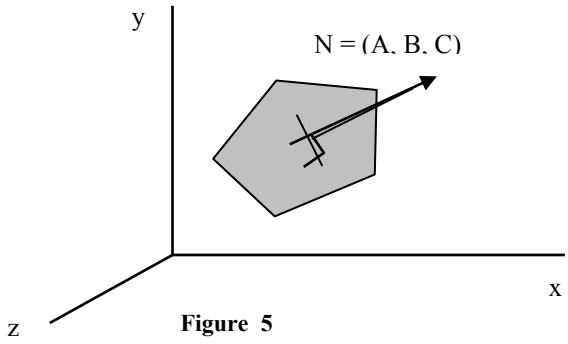


Figure 5

Note: Orientation of a plane surface in space can be described with the normal vector to the plane, as shown in the *Figure 5*. Further the Cartesian component of vector N, normal to the surface of the plane described by equation $Ax + By + Cz + D = 0$, is given by (A,B,C) where parameters A, B, and C are the plane coefficients calculated in equations above.

While dealing with polygon surfaces we have understood that polygon tables play a vital role in storing the information about the polygon. So, the vertex values and other information are entered into the polygon data structure and hence the values for A, B, C and D are computed for each polygon and stored with the other polygon data.

Since, we are usually dealing with polygon surfaces that enclose an object interior, we need to distinguish between the two sides of the surface. The side of or outward side is the "outside" face. If polygon vertices are specified in a counterclockwise direction when viewing the outer side of the plane in a right-handed coordinate system, the direction of the normal vector will be from inside to outside. This is demonstrated for one plane of a unit cube shown in the *Figure 6*.

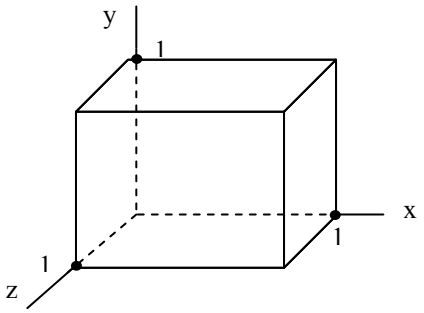


Figure 6

To determine the components of the normal vector for the shaded surface shown in the *Figure 6* of a cube., we select three of the four vertices along the boundary of the polygon. These points are selected in a counterclockwise direction as we view from outside the cube toward the origin. Coordinates for these vertices, in the order selected, can be used in Equations. (3) to obtain the plane coefficients:
$A = 1, B = 0, C = 0, D = -1$. Thus, the normal vector for this plane is in the direction of the positive *x*- axis.

The elements of the plane normal can also be obtained using a vector cross product calculations. We again select three vertex positions, V_1, V_2, and V_3, taken in counterclockwise order when viewing the surface from outside to inside in a right-handed Cartesian system. Forming two vectors, one from V_1 *to* V_2 and the other from V_1 to V_3, we calculate N as the vector cross product:

$$N = (V_2 - V_1) \, X \, (V_3 - V_1)$$

This generates values for the plane parameters A, B and C. We can then obtain the value for parameter D by substituting these values and the coordinates for one of the polygon vertices in plane equation and solving for D. The plane equation can be expressed in vector form using the normal N and the position P of any point in the plane as:

$$N \,.\, P = -D$$

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

$$Ax + By + Cz + D \neq 0$$

We can identify the point as either inside or outside the plane surface according to the sign (negative or positive) of $Ax + By + Cz +$ D:

if $Ax + By + Cz + D < 0$, the point (x, y, z) is inside the surface
if $Ax + By + Cz + D > 0$, the point (x, y, z) is outside the surface

These inequality tests are valid in a right-handed Cartesian system, provided the plane parameter A, B, C and D were calculated using vertices selected in a counterclockwise order when viewing the surface in an outside-to-inside direction. For example, in the above figure of the cube any point outside the shaded plane satisfies the inequality $x - 1 > 0$, while any point inside the plane has an x-coordinates value less than 1.

Example 2: Find equation of plane which passes through point P (0, 0, 0) and say the normal to the plane is given by $\overline{\text{N}}(1, 0, -1)$?

Solution: Let us use method 1 discussed above to determine the equation of the plane

Given $\overline{\text{N}}$ (1, 0, – 1) and P (0, 0, 0) ; equation of plane = ?

say P'(x, y, z) be another point on the plane then line $\overrightarrow{\text{PP'}} = (x\text{-}0, y\text{-}0, z\text{-}0) = x\hat{\text{i}} + \hat{y}\,\text{j} + \text{z}\,\text{k}$

now determine the dot product of PP' and normal N

$$\overline{\text{PP'}}.\ \overline{\text{N}} = 0 \ \Rightarrow n_1x, + n_2y + n_3z - (x_0n_1 + y_0n_2 + z_0n_3) = 0$$
$$1.x + 0.\ y + (-1).z - (0 + 0 + 0) = 0$$
$$\text{x} - \text{z} = 0 \ \rightarrow \text{plane equation}$$

$\Rightarrow x = z$ is the required plane shown in *Figure 7* below

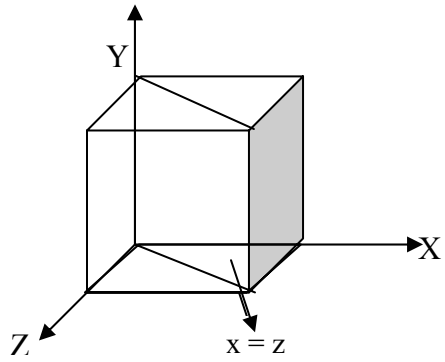


Figure 7

1.3.4 Polygon Meshes

A polygonal surface to be drawn may not be simple and may have enormous curls and curves. Example, a crushed piece of paper, or crushed piece of aluminum foil, etc. In such cases each section of a polygonal surface can be generated (in computer graphics or can be simply drawn) with the help of various standard graphic objects like rectangles, triangles, circles (semicircles), spheres (hemispheres) etc., drawn in a manner that their pattern combination matches with the polygonal surface under construction. This cumulative combination of all standard graphic objects is in fact the mesh or polygonal mesh used to approximate the actual geometry of any complicated object under construction, with the help of the standard graphic objects.

After studying the section 1.3.2 polygon tables, we came to the conclusion that a polygonal surface can be represented with the set of vertices, set of edges and set of surfaces ; which are the general terminologies of nothing but graphs. So we will use this concept here too because, the polygons we need to represent can be arbitrarily large. Thus, it is generally convenient and more appropriate to use a polygon mesh rather than a single mammoth polygon (i.e., single standard graphic object). For example, you can simplify the process of rendering polygons by breaking all polygons into triangles. Triangle renderers can also be implemented in hardware, making it advantageous to break the world down into triangles. Consider below *Figure 8:*

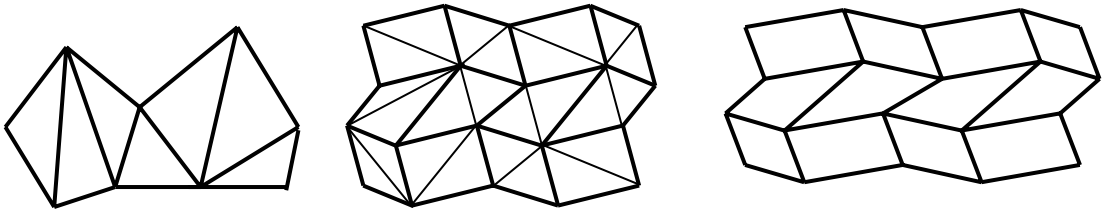


Figure 8 (a) **Figure 8 (b)** **Figure 8 (c)**

Another example where smaller polygons are better is the Inventor lighting model. Inventor computes lighting at vertices and interpolates the values in the interiors of the polygons. By breaking larger surfaces into meshes of smaller polygons, the lighting approximation is improved. From the shown *Figure 8* two important observations are:

- Triangle mesh produces n-2 triangles from a polygon of n vertices.
- Quadrilateral mesh produces (*n*-1) by (*m*-1) quadrilaterals from an *n x m* array of vertices.

It is important to note that specifying polygons with more than three vertices could result in sets of points, which are not co-planar, the reason behind may be the numerical errors or error in selecting the coordinate position of the vertices. Handling non-coplanar vertices is quite difficult, so two ways to handle such situation are:

- Break the polygon into triangles, and deal.
- Approximate *A, B,* and *C* in the plane equation. This can be done either by averaging or by projecting the polygon onto the coordinate planes. *A* should be proportional to the projection in the *yz*-plane, *B* proportional to *xz*, and *C* proportional to xy. High quality graphics system typically model objects with polygon meshes and set up a database of geometric and attribute information to facilitate processing of the polygon facets. Fast hardware implemented polygon renderers are incorporated into such systems with the capability for displaying hundreds of thousands to one millon or more shaded polygon per second including the application of surface texture.

☞Check Your Progress 2

1) Find equation of plane, which passes through point P (1, 1, 1) and say the normal to the plane is given by $\overline{N}$ (-1, 0, -1).

……

……

……

2) Calculate the equation of plane for the quadrilateral planar polygon described by the four vertices v1 (1,0,1), v2 (1,1,0), v3(0,1,1) and v4 (1,1,1).

……

……

……

1.4 BEZIER CURVES AND SURFACES

We had discussed in the previous section of this unit that we can create complex geometries with the help of polygon meshes which are further constituted of standard polygonal objects like triangle, rectangle, square, etc., but apart from this technique to draw complex geometries, we are having some more advanced techniques to do the same job like we can use mathematical equations (parametric equations and polynomial equations), splines, fractals, Bezier curves etc. In this section we will discuss some of these techniques, but to have the flavor of the detailed analysis of these techniques you can refer to books given in suggested readings. Before going on the tour of Bezier curves let us have a brief discussion on other techniques, the technique of using mathematical equations (parametric equations and polynomial equations) to realize the complex natural scenes is not always successful because it requires an enormous number of calculations which consumes an immense amount of processing time. The better technique to generate complex natural scenes is to use fractals. Fractals are geometry methods which use procedures and not mathematical equations to model objects like mountains, waves in sea, etc. There are various kinds of fractals like self-similar, self-affined, etc. This topic is quite interesting but is out of the scope of this unit. Now, let us discuss Bezier curves, which is a Spline approximation method developed by the French engineer Pierre Bezier for use in the design of Renault automobile bodies. Bezier splines have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bezier splines are widely available in various CAD systems, in general graphics packages (such as GL on Silicon Graphics systems), and in assorted drawing and painting packages (such as Aldus Super Paint and Cricket Draw).

Before going into other details of the Bezier curves, we should learn something about the concept of Spline and their representation. Actually Spline is a flexible strip used to produce a smooth curve through a designated set of points known as Control points (it is the style of fitting of the curve between these two points which gives rise to Interpolation and Approximation Splines).We can mathematically describe such a curve with a piecewise cubic Polynomial function whose first and second derivatives are continuous across the various curve sections. In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying specified continuity conditions (Parametric continuity and Geometric continuity conditions) at the boundary of the pieces, without fulfilling these conditions no two curves can be joined smoothly. A spline surface can be described with two sets of

orthogonal spline curves. There are several different kinds of spline specifications that are used in graphics applications. Each individual specification simply refers to a particular type of polynomial with certain specified boundary conditions. Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or the camera in a scene. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

We have mentioned above that it is the style of fitting of the curve between two control points which gives rise to Interpolation and Approximation Splines, i.e., we can specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of the curve. These control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways. When polynomial sections are fitted so that the curve passes through each control point, the resulting curve is said to interpolate the set of control points. On the other hand, when the polynomials are fitted to the general control-point without necessarily passing through any control point, the resulting curve is said to approximate the set of control points. Interpolation curves are commonly used to digitize drawings or to specify animation paths. Approximation curves are primarily used as design tools to structure object surfaces.

A spline curve is defined, modified, and manipulated with operations on the control points. By interactively selecting spatial positions for the control points, a designer can set up an initial curve. After the polynomial fit is displayed for a given set of control points, the designer can then reposition some or all of the control points to restructure the shape of the curve. In addition, the curve can be translated, rotated, or scaled with transformations applied to the control points. CAD packages can also insert extra control points to aid a designer in adjusting the curve shapes.

1.4.1 Bezier Curves

Bezier curves are used in computer graphics to produce curves which appear reasonably smooth at all scales. This spline **approximation method** was developed by French engineer Pierre Bezier for automobile body design. Bezier spline was designed in such a manner that they are very useful and convenient for curve and surface design, and are easy to implement Curves are trajectories of moving points. We will specify them as functions assigning a location of that moving point (in 2D or 3D) to a parameter t, i.e., parametric curves.

Curves are useful in geometric modeling and they should have a shape which has a clear and intuitive relation to the path of the sequence of control points. One family of curves satisfying this requirement are Bezier curve.

The Bezier curve require only two end points and other points that control the endpoint tangent vector.

Bezier curve is defined by a sequence of N + 1 control points, $P_0, P_1, \ldots, P_n$. We defined the Bezier curve using the algorithm (invented by ***DeCasteljeau***), based on recursive splitting of the intervals joining the consecutive control points.

A purely geometric construction for Bezier splines which does not rely on any polynomial formulation, and is extremely easy to understand. The DeCasteljeau method is an algorithm which performs repeated bi-linear interpolation to compute splines of any order.

De Casteljeau algorithm: The control points P0, P1, P2 and P3are joined with line segments called 'control polygon', even though they are not really a polygon but rather a polygonal curve.

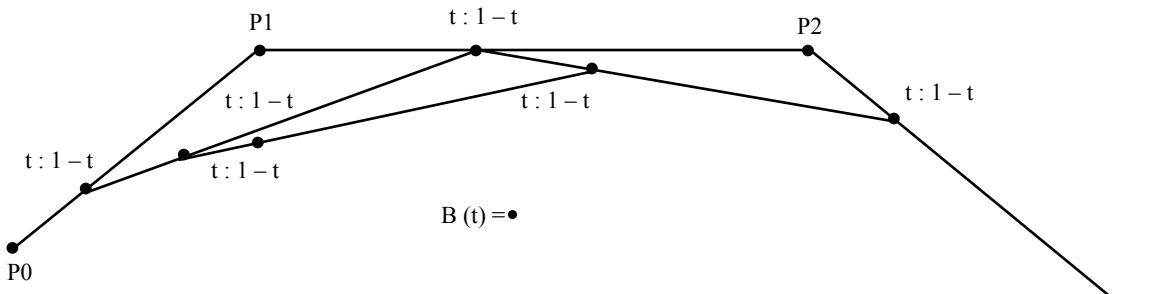


Figure 10

Each of them is then divided in the same ratio t : 1- t, giving rise to the another points. Again, each consecutive two are joined with line segments, which are subdivided and so on, until only one point is left. This is the location of our moving point at time t. The trajectory of that point for times between 0 and 1 is the Bezier curve.

A simple method for constructing a smooth curve that followed a control polygon p with *m-1* vertices for small value of m, the Bezier techniques work well. However, as *m* grows large (*m*>20) Bezier curves exhibit some undesirable properties.

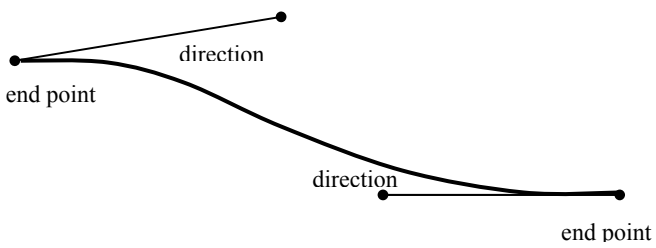


Figure 11 (a) Beizer curve defined by its endpoint vector

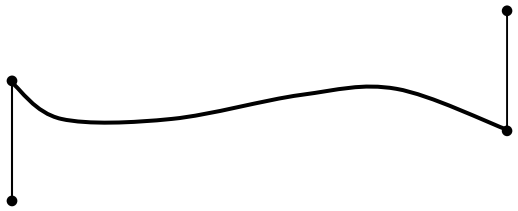


Figure 11 (b): All sorts of curves can be specified with different direction vectors at the end points

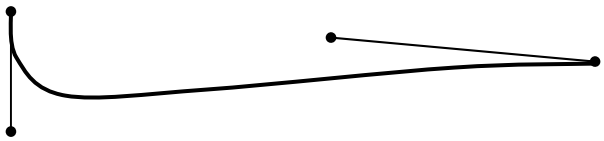


Figure 11 (c): Reflex curves appear when you set the vectors in different directions

In general, a Bezier curve section can be fitted to any number of control points. The number of control points to be approximated and their relative positions determine the degree of the Bezier polynomial. As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending function. For general Bezier curves, the blending-function specification is the most convenient.

Suppose we are given n + 1 control-point positions: $p_k = (x_k, y_k, z_k)$, with *k* varying from 0 to n. These coordinate points can be blended to produce the following position vector *P(u)*, which describes the path of an approximating Bezier polynomial function between p_0 and p_n.

$$P(u) = \sum_{k=0}^{n} p_k B_{k,n}(u), \quad 0 \le u \le 1 \qquad (1)$$

The Bezier blending functions $B_{k,n}(u)$ are the Bernstein polynomials.

$$B_{k,n}(u) = C(n,k)u^k(1-u)^{n-k} \qquad (2)$$

Where the C(n, k) are the binomial coefficients:

$$C(n,k) = \text{nC}_k = \frac{n!}{k!(n-k)!} \qquad (3)$$

equivalently, we can define Bezier blending functions with the recursive calculation

$$B_{k,n}(u) = (1-u)B_{k,n-1}(u) + uB_{k-1,n-1}(u), \qquad n > k \ge 1 \qquad (4)$$

with $\text{BEZ}_{k,k} = u^k$, and $B_{0,k} = (1-u)^k$. Vector equation (1) represents a set of three parametric equations for the individual curve coordinates:

$$\begin{aligned} x(u) &\sum_{k=0}^{n} x_k B_{k,n}(u) \\ y(u) &\sum_{k=0}^{n} y_k B_{k,n}(u) \\ z(u) &\sum_{k=0}^{n} z_k B_{k,n}(u) \end{aligned} \qquad (5)$$

As a rule, a Bezier curve is a polynomial of degree one less than the number of control points used: Three points generate a parabola, four points a cubic curve, and so forth. *Figure 12* below demonstrates the appearance of some Bezier curves for various selections of control points in the xy plane (z = 0). With certain control-point placements, however, we obtain degenerate Bezier polynomials. For example, a Bezier curve generated with three collinear control points is a straight-line segment. And a set of control points that are all at the same coordinate position produces a Bezier "curve" that is a single point.

Bezier curves are commonly found in painting and drawing packages, as well as CAD system, since they are easy to implement and they are reasonably powerful in curve design. Efficient methods for determining coordinate position along a Bezier curve can be set up using recursive calculations. For example, successive binomial coefficients can be calculated as shown below through examples of two-dimensional Bezier curves generated from three, four, and five control points. Dashed lines connect the control-point positions.

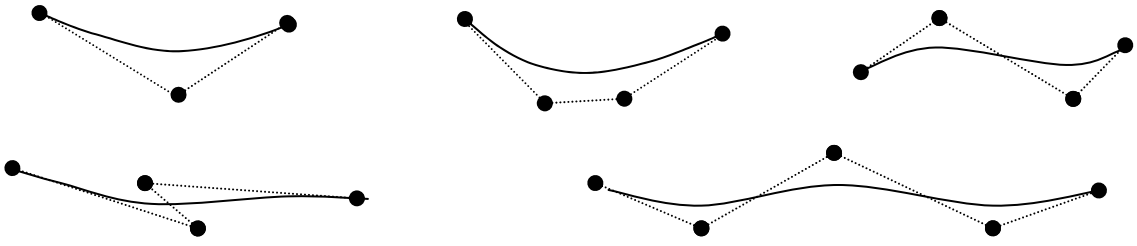


Figure 12

Note:

1) Bezier Curve: $P(u) = \sum_{i=0}^{n} p_i B_{n,i}(u)$ ……. (1)

Where $B_{n,i}(u) = {}_nC_i\, u^i (1-u)^{n-i}$ …….. (2)

$$C(n,i) = {}_nC_i = \frac{n!}{i!(n-i)!} \quad 0 \le u \le 1$$

2) Cubic Bezier curve has n = 3:

$\therefore P(u) = \sum_{i=0}^{3} p_i B_{3,i}(u)$ ……..(3)

$= p_0 B_{3,0}(u) + p_1 B_{3,1}(u) + p_2 B_{3,2}(u) + p_3 B_{3,3}(u)$

Now, lets find $B_{3,0}(u)$, $B_{3,1}(u)$, $B_{3,2}(u)$, $B_{3,3}(u)$ using above equation

$B_{n,i}(u) = {}_nC_i\, u^i (1-u)^{n-i}$

a) $B_{3,0}(u) = {}^3C_0\, u^0 (1-u)^{3-0}$

$= \frac{3!}{0!(3-0)!} . 1. (1-u)^3 = (1-u)^3$

b) $B_{3,1}(u) = {}^3C_1\, u^1 (1-u)^{3-1} = \frac{3!}{1!(3-1)!} u (1-u)^2$

$= 3u(1-u)^2$

c) $B_{3,2}(u) = {}^3C_2\, u^2 (1-u)^{3-2} = \frac{3!}{2!(3-2)!} u^2 (1-u)$

$= 3u^2 (1-u)$

d) $B_{3,3}(u) = {}^3C_3\, u^3 (1-u)^{3-3} = \frac{3!}{3!(3-3)!} u^3$

Using (a), (b) , (c) & (d) in (5) we get

$P(u) = p_0 (1-u)^3 + 3p_1 u (1-u)^2 3p_2 u^2 (1-u) + p_3 u^3$

Example 4: 1 Given p_0 (1, 1): p_1 (2, 3); p_2 (4, 3); p_3 (3, 1) as vertices of Bezier curve determine 3 points on Bezier curve?

Solution: We know Cubic Bezier curve is

$P(u) = \sum_{i=0}^{3} pi\, B_{3,i}(u)$

$\Rightarrow P(u) = p_0 (1-u)^3 + 3p_1 u (1-u)^2 + 3p_2 u^2 (1-u) + p_3 u^3$

$P(u) = (1, 1)(1-u)^3 + 3(2, 3)u(1-u)^2 + 3(4, 3)u^2(1-u) + (3, 1)u^3$.

we choose different values of u from 0 to 1.

u = 0: $P(0) = (1, 1)(1-0)^3 + 0 + 0 + 0 = (1, 1)$

u = 0.5: $P(0.5) = (1, 1)(1 - 0.5)^3 + 3(2, 3)(0.5)(1 - 0.5)^2 + 3(4, 3)(0.5)^2(1 - 0.5) + (3,1)(0.5)^3$

$= (1, 1)(0.5)^3 + (2, 3)(0.375) + (0.375)(4, 3) + (3, 1)(0.125)$

$= (0.125, 0.125) + (0.75, 1.125) + (1.5, 1.125) + (1.125, 0.125)$

$P(0.5) = (3.5, 2.5)$

u = 1 : $P(1) = 0 + 0 + 0 + (3, 1). 1^3$

$= (3, 1)$

Three points on Bezier curve are, P (0) = (1, 1); P (0.5) = (3.5, 2.5) and P (1) = (3, 1).

1.4.2 Properties of Bezier Curves

A very useful property of a Bezier curve is that it always passes through the first and last control points. That is, the boundary conditions at the two ends of the curve are

$$P(0) = p_0$$
$$P(1) = p_n$$

Values of the parametric first derivatives of a Bezier curve at the end points can be calculated from control-point coordinates as

$$P'(0) = -np_0 + np_1$$
$$P'(1) = -np_{n-1} + np_n$$

Thus, the slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoint. Similarly, the parametric second derivatives of a Bezier curve at the endpoints are calculated as

$$p''(0) = n)n-1)[(p_2 - p_1) - (p_1 - p_0)]$$
$$p''(1) = n)n-1)[(p_{n-2} - p_{n-1}) - (p_{n-1} - p_n)]$$

Another important property of any Bezier curve is that it lies within the convex hull (convex polygon boundary) of the control points. This follows from the properties of Bezier blending functions: They are all positive and their sum is always 1,

$$\sum_{k=0}^{n} B_{k,n}(u) = 1$$

so that any curve position is simply the weighted sum of the control-point positions. The convex-hull property for a Bezier curve ensures that the polynomial will not have erratic oscillations.

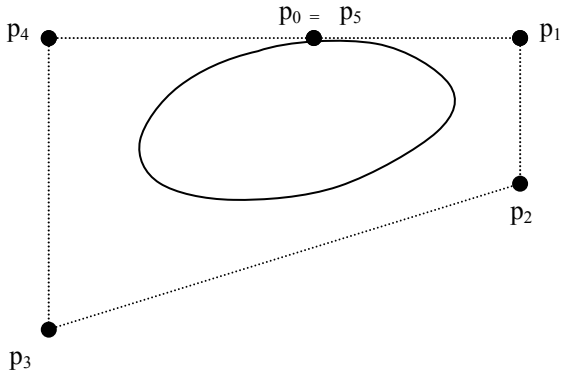


Figure 13 (a): Shows closed Bezier curve generated by specifying the first and last control points at the same location

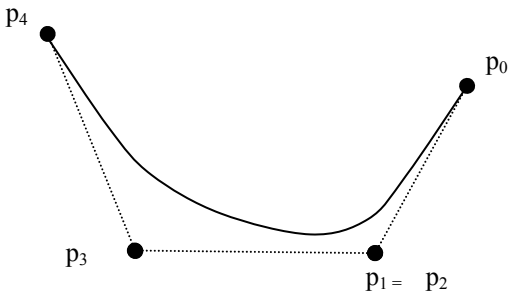


Figure 13 (b): Shows that a Bezier curve can be made to pass closer to a given coordinate position by assigning multiple control points to that position.

Note:

1) Generalising the idea of Bezier curve of degree at n based on n+1 control point p0,…..pn
   P(0)= P_0
   P(1) =p_n

Values of parametric first derivates of Bezier curve at the end points can be calculated from control point

Coordinates as

$P'(0) = -nP_0 + n P_1$

$P'(1) = -nP_{n-1} + nP_n$

Thus, the slope at the beginning of the curve is along the line joining two control points, and the slope at the end of the curve is along the line joining the last two endpoints.

2) **Convex hull**: For $0 \leq t \leq 1$, the Bezier curve lies entirely in the convex hull of its control points. The convex hull property for a Bezier curve ensures that polynomial will not have erratic oscillation.

3) Bezier curves are invariant under affine transformations, but they are not invariant under projective transformations.

4) The vector tangent to the Bezier curve at the start (stop) is parallel to the line connecting the first two (last two) control points.

5) Bezier curves exhibit a *symmetry* property: The same Bezier curve shape is obtained if the control points are specified in the opposite order. The only difference will be the parametric direction of the curve. The direction of increasing parameter reverses when the control points are specified in the reverse order.

6) Adjusting the position of a control point changes the shape of the curve in a "predictable manner". Intuitively, the curve "follows" the control point.

There is *no local control* of this shape modification. Every point on the curve (with the exception of the first and last) move whenever any interior control point is moved.

Following examples prove the discussed properties of the Bezier curves

Example 5: To prove: $P(u = 0) = p_0$

Solution: $\because P(u) = \sum_{i=0}^{n} pi\ Bn, i\ (u)$

$= p_0 B_{n,0}(u) + p_1 B_{n,1}(u) + \ldots\ldots + p_n B_{n,n}(u) \ldots\ldots\ldots\ldots\ldots\ldots (1)$

$B_{n,i}(u) = n_{c_i} u^i (1-u)^{n-i}$

$B_{n,0}(u) = n_{c_0} u^0 (1-u)^{n-0} = \frac{n!}{0!(n-0)!} . 1. (1-u)^n = (1-u)^n$

$B_{n,1}(u) = n_{c_1} u' (1-u)^{n-1} = \frac{n!}{1!(n-1)!} . u. (1-u)^{n-1}$

We observe that all terms expect $B_{n,0}(u)$ have multiple of u^i (i = 0 to n) using these terms with u = 0 in (1) we get,

$P(u = 0) = p_0 (1-0)^n + p_1 . 0. (1-0)^{n-1} . n + 0 + 0 + \ldots\ldots + 0$

$P(u = 0) = p_0$ Proved

Example 6: To prove $\bar{P}(1) = p_n$

Solution: As in the above case we find each term except $B_{n,n}(u)$ will have multiple of $(1-u)^i$ (i = 0 to n) so using u = 1 will lead to result = 0 of all terms except of $B_{n,n}(u)$.

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

$$P(u-1) = p_0 \cdot 0 + p_1 \cdot 0 + \ldots\ldots + p_n \cdot 1^n$$
$$= p_n$$

Example 7: Prove: $\sum_{i=0}^{n} B_{n,\,i} = 1$

Solution: By simple arithmetic we know,

$[(1-u)+u]^n = 1^n = 1$ ……..(1)

expending LHS of (1) binomially we find

$[(1-u)+u]^n = n_{c_0}\,(1-u)^n + n_{c_1}\,u\,(1-u)^{n-1} + n_{c_2}\,u^2\,(1-u)^{n-2} + \ldots.. + n_{c_n}\,u^n$

$$= \sum_{i=0}^{n} n_{c_i}\,u^i\,(1-u)^{n-i}$$

$$[(1-u)+u]^n = \sum_{i=0}^{n} B_{n,\,i}(u) \ldots\ldots\ldots\ldots\ldots\ldots(2)$$

by (1) & (2) we get

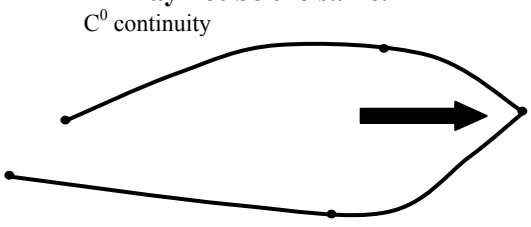
$$\boxed{\sum_{i=0}^{n} B_{n,\,i}(u) = 1}$$

Note: Proof of following properties of Bezier curves is left as an exercise for the students

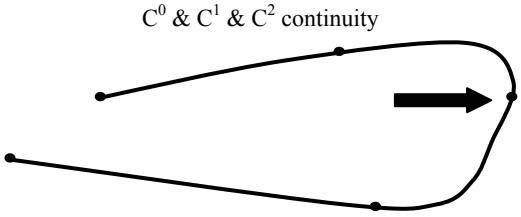
$P'(0) = n(p_1 - p_0)$
$P'(1) = n(p_n - p_{n-1}) = n(p_n - p_{n-1})$
$P''(0) = n(n-1)(p_0 - 2p_1 + p_2)$
$P''(1) = n(n-1)(p_n - 2\,p_{n-1} + p_{n-2})$

To ensure the smooth transition from one section of a piecewise parametric curve or any Bezier curve to the next we can impose various continuity conditions at the connection point for ***parametric continuity*** we match parametric derivatives of adjoining curve section at their common boundary.

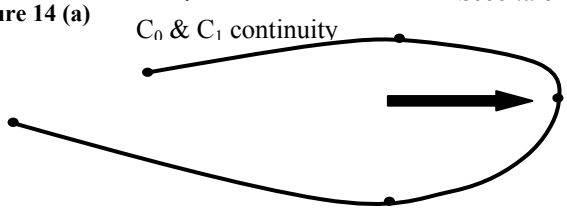
Zero order parametric continuity described by C^0 continuity means curves are only meeting as shown in *Figure 14* while first order parametric continuity referred as C^1 continuity, means that tangent of successive curve sections are equal at their joining point. Second order parametric continuity or C^2 continuity, means that the parametric derivatives of the two curve sections are equal at the intersection. As shown in *Figure 14* below **first order continuity have equal tangent vector but magnitude may not be the same.**



Zero order parametric Continuity
Figure 14 (a)



Second order parametric continuity
Figure 14 (b)



First order parametric continuity
Figure 14 (c)

With the second order continuity, the rate of change of tangent vectors for successive section are equal at intersection thus result in smooth tangent transition from one section to another.

First order continuity is often used in digitized drawing while second order continuity is used in CAD drawings.

Geometric continuity is another method to join two successive curve sections. G^0 continuity is the same as parametric continuity (i.e., two curves sections to be joined must have same coordinate position at the boundary point) i.e., curve section are joined together such that they have same coordinates position at the boundary point. First order geometric continuity $G^{(1)}$ means that the tangent vectors are the same at join point of two successive curves i.e., the parametric first derivative are proportional at the intersection of two successive sections while second order geometric continuity is G^2 means that both first and second order parametric derivates of the two curve sections are proportional at their boundary. In G^2 curvature of the two successive curve sections will match at the joining position

Note:

1) **The joining point on the curve with respect to the parameter based on second derivates of Q(t) is the acceleration. While the Q'(t) is an tangent vector stating velocity. i.e.,** the tangent vector gives the velocity along the curve . the camera velocity and acceleration at join point should be continuous, to avoid jerky movements in the resulting animation sequence.

2) **Curve segment having a continuity C^1 implies the G^1 continuity** but the converse is generally not true. That is, G^1 curves are less restrictive than the C^1 , so curves can be G^1 but not necessarily C^1 .

Curves such as Spline curve, cubic spline curve are curve fitting method used to produce a smooth curve given a set of points throughout out the path weight are distributed. Spline is used in graphics to specify animation paths, to digitize drawings for computer storage mainly CAD application, use them for automobile bodies design in aircraft design, etc.

Spline curve is constructed using Control points which control the shape of the curve Spline curve is a composite curve formed with sections which are polynomial in nature. These pieces are joined together in such a manner that continuity condition at boundary is maintained. A piecewise parametric polynomial section when passes through each control point curve is known to **Interpolate** the set of control points refer *Figure 15*. **On the other hand when polynomial is not fitted to the set to very control point it is known to approximate the set of control points.**

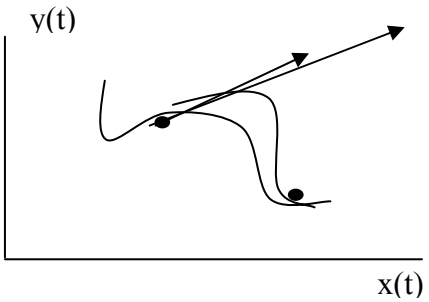


Figure 15 (a): Interpolating curve

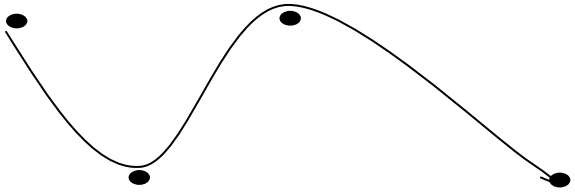


Figure 15 (b): Approximating Curve

Note: if P (u)→ = Bezier curve of order n and Q (u) → Bezier curve of order m

Then Continuities between P(u) and Q(u) are:

1) Positional continuity of 2 curves

$P(u) = \sum_{i=0}^{n} p_i B_{n,i}(u)$ & $Q(u) = \sum_{j=0}^{m} q_j B_{m,j}(u)$

is $p_n = q_0$

2) C^1 continuity of 2 curve P (u) & Q (u) says that point p_{n-1}, p_n on curve P(u) and q_0, q_1 points on curve Q(u) are collinear i.e.,

$$n(p_n - p_{n-1}) = m(q_1 - q_0)$$

$$q_1 = q_0 + \frac{n}{m}(p_n - p_{n-1})$$

$$\Rightarrow \left.\frac{dp}{du}\right|_{u=1} = \left.\frac{dq}{dv}\right|_{v=0}$$

$G^{(1)}$ continuity of 2 curves P(u) & Q(u) at the joining namely the end of P(u) with the beginning of q(u) is:

$$p_n = q_0$$

$n(p_n - p_{n-1}) = kn(q_1 - q_0)$, where k is a constant & k > 0

$\Rightarrow p_{n-1}, p_n = q_0, q_1$ are collinear

3) c2 continuity:

a) $C^{(1)}$ continuity
b) $m(m-1)(q_0 - 2q_1 + q_2) = n(n-1)(p_n - 2p_{n-1} + p_{n-2})$

i.e., points p_{n-2}, p_{n-1}, p_n of P(u) and points q_0 , q_1, q_2 of Q(u) must be collinear

further we can check whether both first and second order derivatives of two curve sections are the same at the intersection or not i.e.,

$$\left.\frac{dp}{du}\right|_{u=1} = \left.\frac{dq}{dv}\right|_{v=0}$$

and

$$\left.\frac{d^2p}{du^2}\right|_{u=1} = \left.\frac{d^2q}{dv^2}\right|_{v=0}$$

if they are same we can say we have C^2 continuity

Note: similarly we can define higher order parametric continuities

Example 8: An animation shows a car driving along a road which is specified by a Bezier curve with the following control points:

$\mathbf{X_k}$	0	5	40	50
$\mathbf{Y_k}$	0	40	5	15

The animation lasts 10 seconds and the key frames are to be computed at 1 second intervals. Calculate the position of the car on the road at the start of the 6th second of the animation.

Solution: Using similar methods to the previous exercise we can calculate the blending functions as:

1. $B_{03} = 3!/(0! \times (3-0)!)\ u^0(1-u)^{(3-0)} = 1u^0(1-u)^3 = \mathbf{(1-u)^3}$
2. $B_{13} = 3!/(1! \times (3-1)!)\ u^1(1-u)^{(3-1)} = 3u^1(1-u)^2 = \mathbf{3u(1-u)^2}$
3. $B_{23} = 3!/(2! \times (3-2)!)\ u^2(1-u)^{(3-2)} = 3u^2(1-u)^1 = \mathbf{3u^2(1-u)}$
4. $B_{33} = 3!/(3! \times (3-3)!)\ u^3(1-u)^{(3-3)} = 1u^3(1-u)^0 = \mathbf{u^3}$

The function x(u) is equal to $x(u) = \sum x_k B_k$ where k=0,1,2,3

$$x(u) = \sum x_k B_k = x_0B_{03} + x_1B_{13} + x_2B_{23} + x_2B_{33}$$
$$= (0)(1-u)^3 + 5\,[3u(1-u)^2] + 40\,[3u^2(1-u)] + 50\,u^3$$
$$= 15u(1-u)^2 + 120u^2(1-u) + 50u^3$$

similarly
$$y(u) = y_0B_{03} + y_1B_{13} + y_2B_{23} + y_2B_{33}$$
$$= (0)(1-u)^3 + 40\,[3u(1-u)^2] + 5\,[3u^2(1-u)] + 15\,u^3$$
$$= 120u(1-u)^2 + 15u^2(1-u) + 15u^3$$

At the start of the sixth second of the animation, i.e., when u=0.6, we can use these equations to work out that x(0.6) = **29.52** and y(0.6) = **16.92**.

The path of the car looks like this

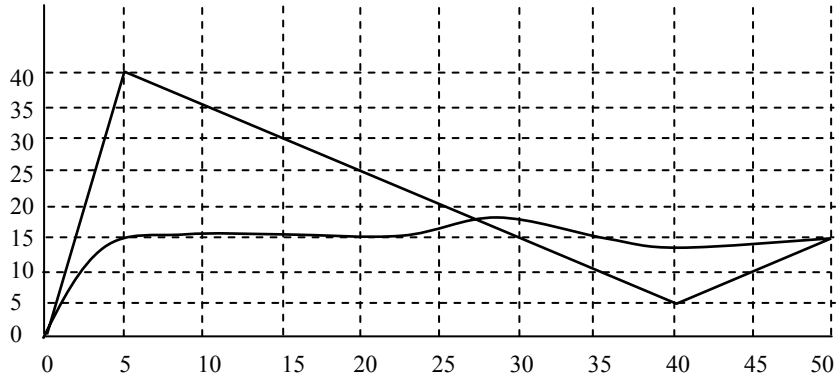


Figure 16

1.4.3 Bezier Surfaces

Two sets of Bezier curve can be used to design an object surface by specifying by an input mesh of control points. The Bézier surface is formed as the cartesian product of the blending functions of two Bézier curves.

$$P(u,v) = \sum_{j=0}^{m}\sum_{k=0}^{n} p_{j,k} B_{j,m}(v) B_{k,n}(u)$$

with $p_{j,k}$ specifying the location of the (m + 1) by (n + 1) control points.

The corresponding properties of the Bézier curve apply to the Bézier surface.

The surface does not in general pass through the control points except for the corners of the control point grid.

The surface is contained within the convex hull of the control points.

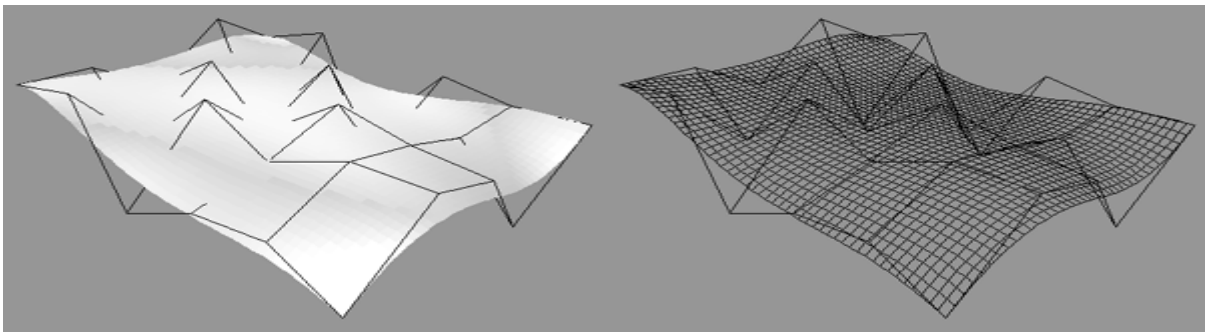


Figure 17

The control points are connected via a dashed line, and solid lenes shows curves of constant u and constant v. Each curve is ploted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval curves of constant v are plotted.

Figures (a), (b), (c) illustrate Bezier surface plots. The control points are connected by dashed lines, and the solid lines show curves of constant *u* and constant *v*. Each curve of constant *u* is plotted by varying *v* over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant *v* are plotted similarly.

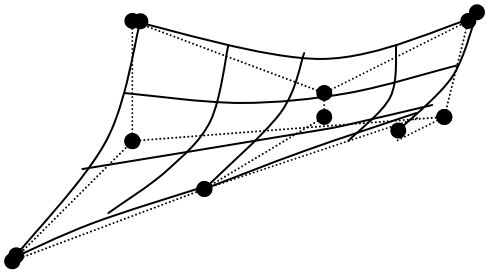


Figure 18 (a)
Bezier surfaces constructed for m=3, n=3, Dashed lines connect the control points

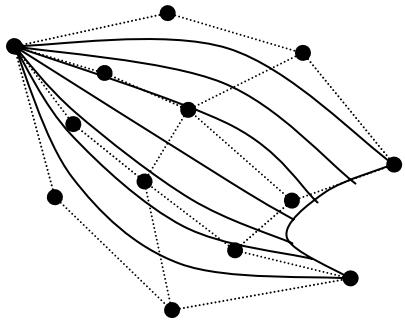


Figure 18 (b)
Bezier surfaces constructed for m=4, n=4. Dashed lines connect the control points.

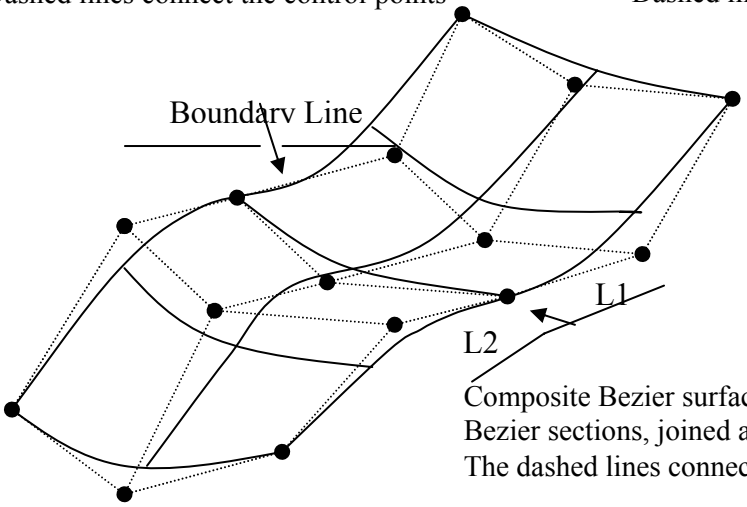


Figure 18 (c)
Composite Bezier surface constructed with two Bezier sections, joined at the indicated boundary line. The dashed lines connect specified control points.

In *Figure (c)* first-order continuity is established by making the ratio of length L_1 to length L_2 constant for each collinear line of control points across the boundary between the surface sections. *Figure (c)* also illustrates a surface formed with two Bezier sections. As with curves, a smooth transition from one section to the other is assured by establishing both zero-order and first-order continuity at the boundary line. Zero-order continuity is obtained by matching control points at the boundary. First-

order continuity is obtained by choosing control points along a straight line across the boundary and by maintaining a constant ration of collinear line segments for each set of specified control points across section boundaries.

☞ Check Your Progress 3

1) Based on the Bezier curve definition, derive the equation of the 3 point Bezier curve defined by the following control points. (-1,0), (0,2), and (1,0).

……

……

……

……

……

……

2) Discuss how a Bezier surface is created. In particular discuss
   (a) The basic structure of a Bezier surface.
   (b) What blending function is used.

……

……

……

……

……

……

……

……

1.5 SURFACE OF REVOLUTION

In the above sections we have learned various techniques of generating curves, but if we want to generate a close geometry, which is very symmetric in all the halves, i.e., front back, top, bottom; then it will be quite difficult for any person by doing it separately for each half. Under such a situation the concept of surface revolution is quite useful, because it helps in producing graphic objects such that they are very symmetric in every half. These geometries produced after Surface revolution are also known as Sweep Representations, which is one of the solid modeling construction techniques provided by many graphic packages. Sweep representations are useful for constructing three-dimensional objects that possess translational, rotational, or other symmetries. We can represent such objects by specifying a two-dimensional shape and a sweep that moves the shape through a region of space. A set of two-dimensional primitives, such as circles and rectangles, can be provided for sweep representations as menu options. Other methods for obtaining two-dimensional figures include closed spline-curve constructions and cross-sectional slices of solid objects. Let us discuss this simple technique of generating symmetric objects.

Say we are having a point P(x1,y1) in the X-Y plane (not at origin); if we make that point to revolve about Z axis then the point will trace circular geometry. Similarly, if the basic curve subjected to revolution is a Line segment then it will produce a

cylinder (symmetric about the axis of revolution). Therefore, different base curves we will get different surfaces of revolution, e.g.,

i) ***Base Curve*:** say just a point when it rotates about x axis it will trace a circular surface of revolution.

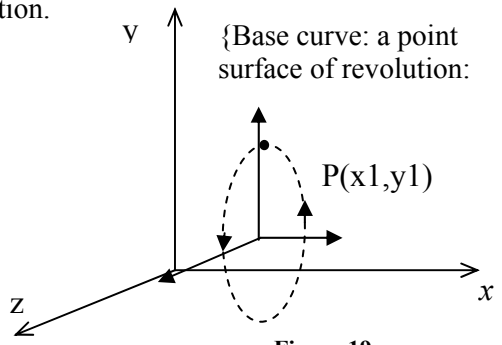


Figure 19

ii) If base curve ⇒ a line segment parallel to the *x* axis then a cylinder will be traced as a surface of revolution.

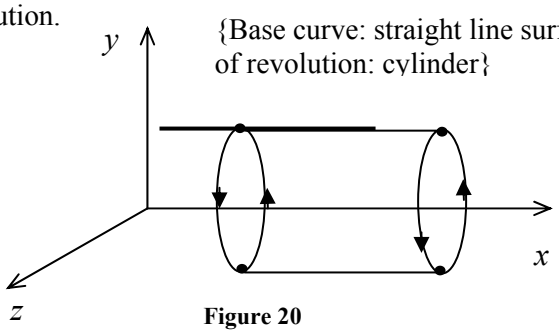


Figure 20

iii) If the line segment is inclined with one end point at the origin then a cone will be traced

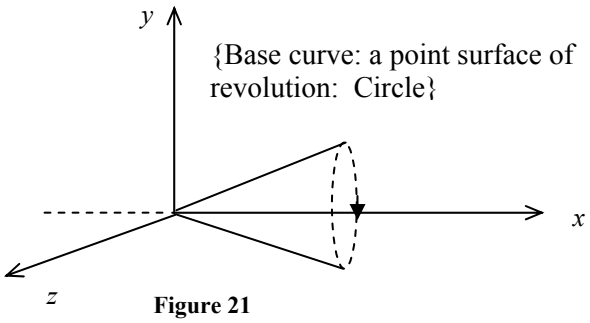


Figure 21

iv) If a semi-circle is rotated then a sphere is traced.

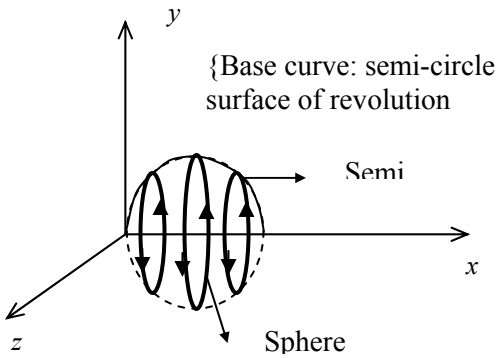


Figure 22

By this moment I hope it will be clear to you how a simple revolution of curve about any axis simplifies the creation of symmetric geometries You may use combination of basic curves to generate complex geometries. Now let us discuss the mathematics behind such creations.

How to find surface of revolution: Say there is an arbitrary base curve which when rotated about *x*-axis traces an arbitrary surface of revolution about the axis it is revolving

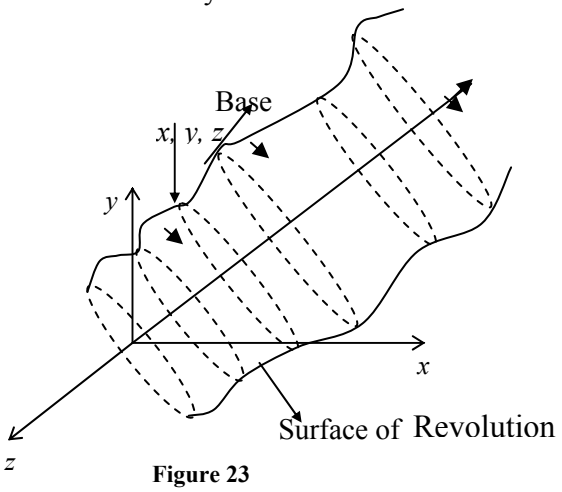


Figure 23

To find surface of revolution consider a point. (x, y, z) on base curve. Say curve is to revolve about *x*-axis $\Rightarrow$ Rotational transformation about *x*-axis is to be applied to (x, y, z).

$$(x, y, z) \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}; 0 \le \theta \le 2\pi.$$

As the revolution is about X axis so it has to be constant, for each section of the complete curve (you may change this value of X in steps, as you do in any programming language 'X++'; hence a continuous, symmetric curve of any arbitrary shape will be available).

$$\Rightarrow (x, y, z) \rightarrow (x, y, 0) \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}; 0 \le \theta \le 2\pi.$$

$\Rightarrow (x, y, z) \rightarrow (x, y \cos\theta, y \sin \theta)$. These points trace surface of revolution about *x*-axis.

Note: a) if a point on base curve is given by parametric form, i.e., $(x(u), y(u), z(u))$ then, surface of revolution about x-axis will be
$[x(u), y(u), z(u)] \rightarrow [x(u), y(u) \cos\theta, y(u) \sin\theta]$
$0 \le u \le 1; \quad 0 \le \theta \le 2\pi.$

b) Tracing an image involves movement of points from one place to another, i.e., translational transformation is to be used. If (x, y, z) is a point on a base curve then moving the respective points on base curve from one place to other traces an image.

c) If $\vec{d}$ $\Rightarrow$ the direction in which curve is to be moved and
v $\Rightarrow$ scalar quantity representing the amount by which curve is to be shifted.
Displacing the curve by amount $v\vec{d}$, the curve will be traced at a new position or is swept to a new position.

$(x(u), y(u), z(u)) \rightarrow$ coordinate points of base curve in parametric form ($u \rightarrow$parameter)

$(x(u), y(u), z(u)) \rightarrow (x(u), y(u), z(u)) + v\vec{d}$

$0 \le u \le 1; \qquad 0 \le v \le 1.$

In general, we can specify sweep constructions using any path. For rotational sweeps, we can move along a circular path through any angular distance from 0 to 360°. For noncircular paths, we can specify the curve function describing the path and the distance of travel along the path. In addition, we can vary the shape or size of the cross section along the sweep path. Or we could vary the orientation of the cross section relative to the sweep path as we move the shape through a region space.

Figure 24 illustrates construction of a solid with a translational sweep. Translating the control points of the periodic spline curve in (a1) generates the solid shown in (b1), whose surface can be described with point function P(*u, v*).

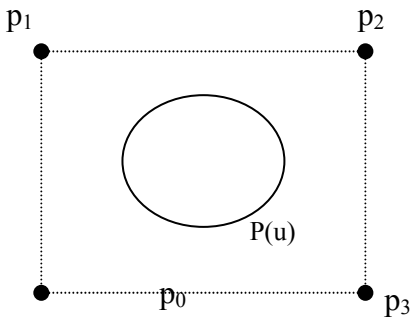


Figure 24 (a)

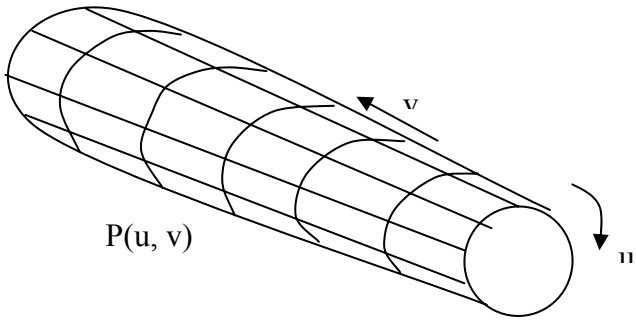


Figure 24 (b)

Figure 25 illustrates Construction of a solid with a rotational sweep. Rotating the control pints of the periodic spline curve in (a2) about the given rotation axis generates the solid shown in (b2), whose surface can be described with point function P(*u, v*).

Axis of Rotation

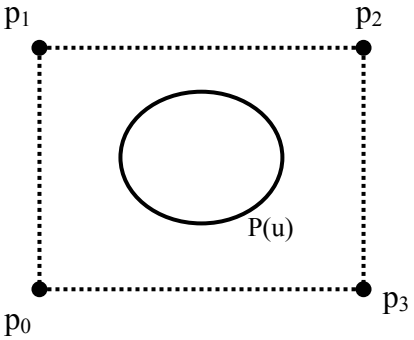


Figure 25 (a)

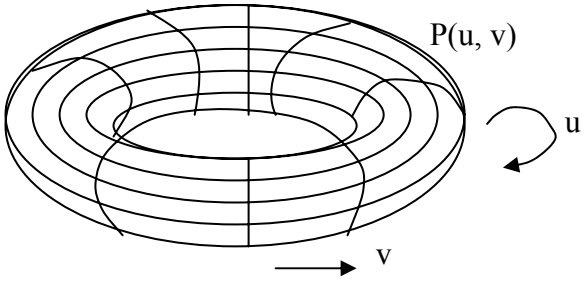


Figure 25 (b)

In the example of object shown in *Figure 25*, the designing is done by using a rotational sweep. This time, the periodic spline cross-section is rotated about an axis of rotation specified in the plane of the cross-section to produce the wireframe.

Any axis can be chosen for a rotational sweep. If we use a rotation axis perpendicular to the plane of the spline cross-section in *Figure 25(b)* we generate a two-dimensional shape. But if the cross section shown in this figure has depth, then we are using one three-dimensional object to generate another.

1.6 SUMMARY

This unit has covered the techniques of generating polygons, curves and closed surfaces. Under those techniques we have discussed various polygonal representational methods like tables, equations. Under the section of curves and surfaces after having brief discussion on mathematical representations of the curve through polynomial and parametric equation we have discussed the topic of Bezier curves and their applications. In the end of the unit we have discussed the concept of surface of revolution, which helps in generating 2D and 3D surfaces from 1D and 2D geometries.

1.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) So far as the accomplishment of the task to draw a polygon surface is concerned, it can be done with the help of Vertex table and Edge table but then there is a possibility that some edges can be drawn twice. Thus, the system will be doing redundant processing for the display of object. Hence the time of execution of task increases.

2) This extra information will help in rapid identification of the common edges between the polygon surfaces. This information in turn provides useful support in the process of surface rendering, which can be done smoothly when surface shading varies smoothly across the edge from one polygon surface to the next.

3) Yes, we can expand the vertex table, the additional information in the table will help in cross-referencing the vertices with corresponding edges. This will help in reducing he redundant processing of a vertex information associated with a polygon surface, hence reduces the execution time.

4) The more the information, the easier to check the error, if precisely all information is available in the table then you need not waste time in calculations and other tasks to diagnose an error in the polygon representation.

5) Left for the student to do.

Check Your Progress 2

1) Left as an exercise.

2) Left as an exercise.

Check Your Progress 3

1) To make the maths easier to understand we will first calculate the valuess of the blending function B_{02}, B_{12} and B_{22}.

$$B_{02} = 2!/(0! \times (2-0)!)\, u^0(1-u)^2 = 1u^0(1-u)^2 = \mathbf{(1-u)^2}$$

$$B_{12} = 2!/(0! \times (2-1)!)\, u^1(1-u)^{2-1} = 2u^1(1-u)^1 = \mathbf{2u(1-u)}$$

$$B_{22} = 2!/(0! \times (2-2)!)\, u^2(1-u)^{2-2} = 1u^2(1-u)^0 = \mathbf{u^2}$$

We can therefore obtain the following equations.

$$\begin{aligned} x(u) &= \sum x_k B_k = x_0 B_{02} + x_1 B_{12} + x_2 B_{22} \\ &= (-1)(1-u)^2 + 0\,[2u(1-u)] + 1u^2 \\ &= 2u - 1 \qquad \text{similarly } y(u) \end{aligned}$$

$$\begin{aligned} y(u) &= \sum y_k B_k = y_0 B_{02} + y_1 B_{12} + y_2 B_{22} \\ &= (0)(1-u)^2 + 2\,[2u(1-u)] + 0u^2 \\ &= 4u(1-u) \end{aligned}$$

2) a) A Bezier surface is a surface formed by two orthogonal Bezier curves such that they form a mesh of control points. Computing the value of each curve for a constant value of u, will produce the control points for the columns run across the curves.

b) The same blending function is used for Bezier surfaces that is used for Bezier splines. The equation for a surface is $P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} B_{j,m}(v) B_{k,n}(u)$