Unit 3
Three Dimensional Object Representations

Graphical scenes can contain many different kinds of objects like trees, flowers, rocks, waters...etc. No single method can be used to describe objects that will include all features of those different materials. To produce realistic display of scenes, we need to use representations that accurately model object characteristics.

- Simple Euclidean objects like polyhedrons and ellipsoids can be represented by polygon and quadric surfaces.
- Spline surface are useful for designing aircraft wings, gears and other engineering objects.
- Procedural methods and particle systems allow us to give accurate representation of clouds, clumps of grass, and other natural objects.
- Octree encodings are used to represent internal features of objects such as medical CT images.

Representation schemes for solid objects are often divided into two broad categories:

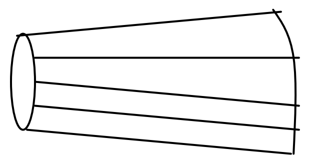
1. **Boundary representations**: describes a 3D object as a set of polygonal surfaces, separate the object interior from environment.
2. **Space-partitioning representation**: used to describe interior properties, by partitioning the spatial region, containing an object into a set of small, non-overlapping, contiguous solids. e.g. 3D object as Octree representation.

Boundary Representation

Each 3D object is supposed to be formed its surface by collection of polygon facets and spline patches. Some of the boundary representation methods for 3D surface are:

1. Polygon Surfaces

It is most common representation for 3D graphics object. In this representation, a 3D object is represented by a set of surfaces that enclose the object interior. Many graphics system use this method. Set of polygons are stored for object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.



A 3D object represented by polygons

The polygon surfaces are common in design and solid-modeling applications, since wire frame display can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.

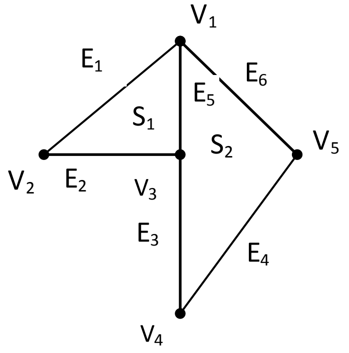
Polygon Table

A polygon surface is specified with a set of vertex co-ordinates and associated attribute parameters. A convenient organization for storing geometric data is to create 3 lists:

- A vertex table

- An edge table
- A polygon surface table.

Vertex table stores co-ordinates of each vertex in the object.
The edge table stores the Edge information of each edge of polygon facets.
The polygon surface table stores the surface information for each surface i.e. each surface is represented by edge lists of polygon.
Consider the surface contains polygonal facets as shown in figure (only two polygons are taken here)



S_1 and S_2 are two polygon surface that represent the boundary of some 3D object.
Foe storing geometric data, we can use following three tables

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

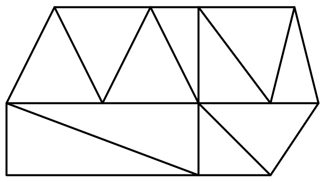
EDGE TABLE
E_1: V_1,V_2
E_2: V_2,V_3
E_3: V_3,V_4
E_4: V_4,V_5
E_5: V_1,V_3
E_6: V_5, V_1

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

The object can be displayed efficiently by using data from tables and processing them for surface rendering and visible surface determination.

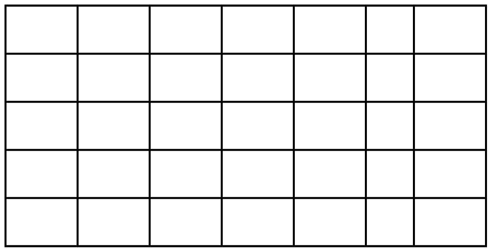
Polygon Meshes
A polygon mesh is collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons. An edge connects two vertices and a polygon is a closed sequence of edges. An edge can be shared by two polygons and a vertex is shared by at least two edges.

When object surface is to be tiled, it is more convenient to specify the surface facets with a mesh function. One type of polygon mesh is triangle strip. This function produce n-2 connected triangles.



Triangular Mesh

Another similar function is the quadrilateral mesh, which generates a mesh of (n-1) by (m-1) quadrilaterals, given the co-ordinates for an $n \times m$ array of vertices.



6 by 8 vertices array , 35 element quadrilateral mesh

If the surface of 3D object is planer, it is comfortable to represent surface with meshes.

Representing polygon meshes

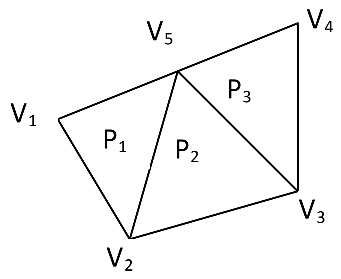
In explicit representation, each polygon is represented by a list of vertex co-ordinates.

$$P = ((x_1, y_1, z_1), (x_2, y_2, y_2), \ldots\ldots., (x_n, y_n, z_n))$$

The vertices are stored in order traveling around the polygon. There are edges between successive vertices in the list and between the last and first vertices.

For a single polygon it is efficient but for polygon mesh it is not space efficient since no of vertices may duplicate.

So another method is to define polygon with pointers to a vertex list. So each vertex is stored just once, in vertex list $V = \{v_1, v_2 \ldots\ldots v_n\}$ A polygon is defined by list of indices (pointers) into the vertex list e.g. A polygon made up of vertices $3,5,7,10$ in vertex list be represented as $P_1 = \{3,5,7,10\}$



Representing polygon mesh with each polygon as vertex list.

$P_1 = \{v_1, v_2, v_5\}$

$P_2 = \{v_2, v_3, v_5\}$

$P_3 = \{v_3, v_4, v_5\}$

Here most of the vertices are duplicated so it is not efficient.

Representation with indexes into a vertex list

$$V = \{v_1, v_2, v_3, v_4, v_5\} = \{(x_1, y_1, z_1), \ldots\ldots(x_5, y_5, z_5)\}$$

$$P_1 = \{1,2,3\}$$

$$P_2 = \{2,3,5\}$$

$$P_3 = \{3,4,5\}$$

Defining polygons by pointers to an edge list
In this method, we have vertex list V, represent the polygon as a list of pointers not to the vertex list but to an edge list. Each edge in edge list points to the two vertices in the vertex list. Also to one or two polygon, the edge belongs.
Hence we describe polygon as

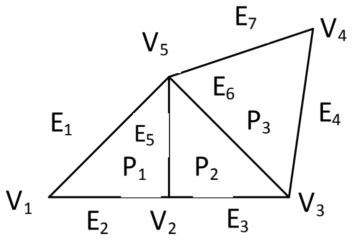
$$P = (E_1, E_2, \ldots . E_n)$$

and an edge as

$$E = (V_1, V_2, P_1, P_2)$$

2Here if edge belongs to only one polygon, either
Then P_1 or P_2 is null.

For the mesh given below,



$$V = \{v_1, v_2, v_3, v_4, v_5\} = \{(x_1, y_1, z_1), \ldots \ldots (x_5, y_5, z_5)\}$$

$E_1 = (V_1, V_5, P_1, N)$
$E_2 = (V_1, V_2, P_1, N)$
$E_3 = (V_2, V_3, P_2, N)$
$E_4 = (V_3, V_4, P_3, N)$
$E_5 = (V_2, V_5, P_1, P_2)$
$E_6 = (V_3, V_5, P_1, P_3)$ Here N represents Null.
$E_7 = (V_4, V_5, P_3, N)$
$P_1 = (E_1, E_2, E_3)$
$P_2 = (E_3, E_6, E_5)$
$P_3 = (E_4, E_7, E_6)$

Polygon Surface: Plane Equation Method
Plane equation method is another method for representation the polygon surface for 3D object. The information about the spatial orientation of object is described by its individual surface, which is obtained by the vertex co-ordinates and the equation of each surface. The equation for a plane surface can be expressed in the form:

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

Where (x, y, z) is any point on the plane, and A, B, C, D are constants describing the spatial properties of the plane. The values of A, B, C, D can be obtained by solving a set of three plane equations using co-ordinate values of 3 noncollinear points on the plane.

Let (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3) are three such points on the plane, then,

$Ax_1 + By_1 + Cz_1 + D = 0$
$Ax_2 + By_2 + Cz_2 + D = 0$
$Ax_3 + By_3 + Cz_3 + D = 0$

The solution of these equations can be obtained in determinant from using Cramer's rule as:-

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

For any points (x, y, z)
If $Ax + By + Cz + D \neq 0$, then (x, y, z) is not on the plane.
If $Ax + By + Cz + D < 0$, then (x, y, z) is inside the plane i. e. invisible side
If $Ax + By + Cz + D > 0$, then (x, y, z) is lies out side the surface.

2. Quadric Surface

Quadric Surface is one of the frequently used 3D objects surface representation. The quadric surface can be represented by a second degree polynomial. This includes:

1. Sphere: For the set of surface points (x, y, z) the spherical surface is represented as: $x^2+y^2+z^2 = r^2$, with radius r and centered at co-ordinate origin.
2. Ellipsoid: $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$, where (x, y, z) is the surface points and a, b, c are the radii on X, Y and Z directions respectively.
3. Elliptic parboiled: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = z$
4. Hyperbolic parboiled : $\frac{x^2}{a^2} - \frac{y^2}{b^2} = z$
5. Elliptic cone : $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$
6. Hyperboloid of one sheet: $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$
7. Hyperboloid of two sheet: $\frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$

3. Wireframe Representation:

In this method 3D objects is represented as a list of straight lines, each of which is represented by its two end points (x_1, y_1, z_1) and (x_2, y_2, z_2). This method only shows the skeletal structure of the objects. It is simple and can see through the object and fast method. But independent line data structure is very inefficient i.e. don't know what is connected to what. In this method the scenes represented are not realistic.

4. Blobby Objects:

Some objects don't maintain a fixed shape but change their surface characteristics in certain motions or when proximity to another objects e.g. molecular structures, water droplets, other liquid effects, melting objects, muscle shaped in human body etc. These objects can be described as exhibiting "blobbiness" and are referred as blobby objects.

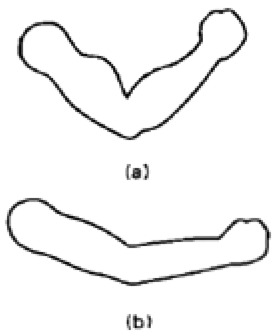


Fig: Blobby muscle shapes in a human arm

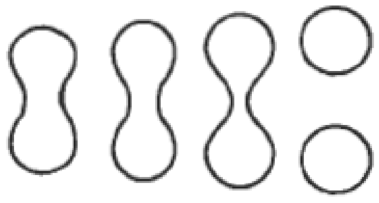


Fig: Molecular bonding (stretching and contracting into spheres)

Several models have been developed for representing blobby objects as distribution functions over a region of space. One way is to use Gaussian density function or bumps. A surface function is defined as:

$$f(x, y, z) = \sum_k b_{k0} e^{-a_k r_k^2} - T = 0$$

Where $r_k = \sqrt{x_k^2 + y_k^2 + z_k^2}$, T = Threshold, a and b are used to adjust amount of blobbiness.

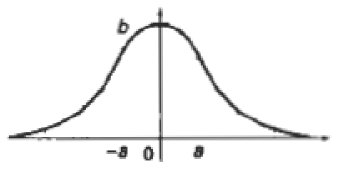


Fig: A three-dimensional Gaussian bump centered at position 0, with height band standard deviation a.

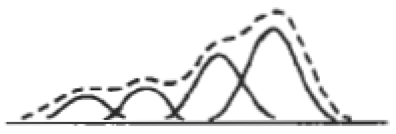


Fig: A composite blobby object formed with four Gaussian bumps

Other method for generating for generating blobby objects uses quadratic density function as:

$$f(r) = \begin{cases} b(1 - 3r^2/d^2), & \text{if } 0 < r \le d/3 \\ \frac{3}{2}b(1 - r/d)^2, & \text{if } d/3 < r \le d \\ 0, & \text{if } r > d \end{cases}$$

Advantages

- Can represent organic, blobby or liquid line structures
- Suitable for modeling natural phenomena like water, human body
- Surface properties cam be easily derived from mathematical equations.

Disadvantages:

- Requires expensive computation
- Requires special rendering engine
- Not supported by most graphics hardware

5. Spline Representation

A Spline is a flexible strips used to produce smooth curve through a designated set of points. A curve drawn with these set of points is spline curve. Spline curves are used to model 3D object surface shape smoothly.

Mathematically, spline are described as piece-wise cubic polynomial functions. In computer graphics, a spline surface can be described with two set of orthogonal spline curves. Spline is used in graphics application to design and digitalize drawings for storage in computer and to specify animation path. Typical CAD application for spline includes the design of automobile bodies, aircraft and spacecraft surface etc.

Interpolation and approximation spline

- Given the set of control points, the curve is said to **interpolate** the control point if it passes through each points.
- If the curve is fitted from the given control points such that it follows the path of control point without necessarily passing through the set of point, then it is said to **approximate** the set of control point.

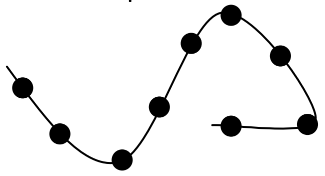


Fig: *A* set of nine control point **interpolated** with piecewise continuous polynomial sections.

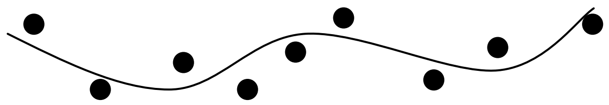


Fig: *A* set of nine control points **approximated** with the piecewise continuous polynomial sections

Spline Specifications

There are three equivalent methods for specifying a particular spline representation:

A. ***Boundary conditions*****:** We can state the set of boundary conditions that are imposed on the spline.

For illustration, suppose we have parametric cubic polynomial representation for the x-coordinate along the path of a spline section:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \qquad 0 \le u \le 1$$

Boundary conditions for this curve might be set, for example, on the endpoint coordinates *x(0)* and x(1) and on the parametric first derivatives at the endpoints *x'(0)* and x'(1). These four boundary conditions are sufficient to determine the values of the four coefficients a_x, b_x, c_x, and d_x.

B. ***Characterizing matrix:*** We can state the matrix that characterizes the spline.

From the boundary condition, we can obtain the characterizing matrix for spline. Then the parametric equation can be written as:

$$x(u) = [u^3 \; u^2 \; u \; 1] \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$
$$= \mathbf{U} \cdot \mathbf{C}$$

Where U is the row matrix of powers of parameter u**,** and C is the coefficient column matrix.

C. ***Blending Functions:*** We can state the set of blending functions (or basis functions) that determine how specified geometric constraints on the curve are combined to calculate positions along the curve path.

From matrix representation in B, we can obtain polynomial representation for coordinate x in terms of the geometric constraint parameters:

$$x(u) = \sum_{k=0}^{3} g_k . BF_k(u)$$

where g_k are the constraint parameters, such as the control-point coordinates and slope of the curve at the control points, and $BF_k(u)$ are the polynomial blending functions.

Cubic spline

- It is most often used to set up path for object motions o tot provide a representation for an existing object or drawing. To design surface of 3D object any spline curve can be represented by piece-wise cubic spline.
- Cubic polynomial offers a reasonable compromise between flexibility and speed of computation. Cubic spline requires less calculation with comparison to higher order polynomials and requires less memory. Compared to lower order polynomial cubic spline are more flexible for modeling arbitrary curve shape.

Given a set of control points, cubic interpolation spines are obtained by fitting the input points with a piecewise cubic polynomial curve that passes through every control points.

Suppose we have n+1 control points specified with co-ordinates.

$p_k = (x_k, y_k, z_k)$,   k = 0,1,2,3…………………n

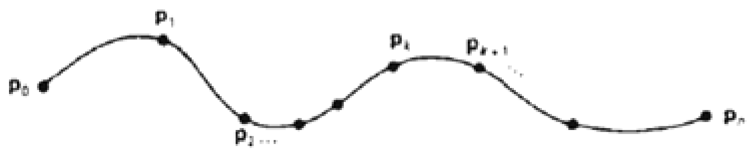


Fig: A piecewise continuous cubic-spline interpolation of *n* + 1 control points

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of parametric equations.

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \qquad (0 \le u \le 1) \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned}$$

For each of these three equations, we need to determine the values of the four coefficients a, b, c, and d in the polynomial representation for each of the *n* curve sections between the *n* + 1 control points. We do this by setting enough boundary conditions at the "joints" between curve sections so that we can obtain numerical values for all the coefficients.

6. Bezier curve and surface

This is spline approximation method, developed by the French Engineer Pierre Bezier for use in the design of automobile body. Beziers spline has a number of properties that make them highly useful and convenient for curve and surface design. They are easy to implement. For this reason, Bezier spline is widely available in various CAD systems.

Beziers curves

In general, Bezier curve can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of Bezier polynomial. The Bezier curve can be specified with boundary condition, with characterizing matrix or blending functions. But for general Bezier curves, blending function specification is most convenient.

Suppose we have n+1 control points: $p_k(x_k, y_k, z_k)$, $0 \le k \le n$. These co-ordinate points can be blended to produce the following position vector P(u) which describes path of an approximating Bezier polynomial function p_0 and p_n.

$$P(u) = \sum_{k=0}^{n} p_k.BEZ_{k,n}(u), \qquad 0\le u\le 1 \text{ ------------------------ } 1$$

The Bezier belending function $BEZ_{k,n}(u)$ are the Bernstein polynomial:

$$BEZ_{k,n}(u) = c(n,k)u^k(1-u)^{n-k}$$

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

$$C(n, k) = n! / k!(n-k)!$$

The vector equation (1) represents a set of three parametric equations for individual curve coordinates.

$$x(u) = \sum_{k=0}^{n} x_k.BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k.BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k.BEZ_{k,n}(u)$$

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.

Fig 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:

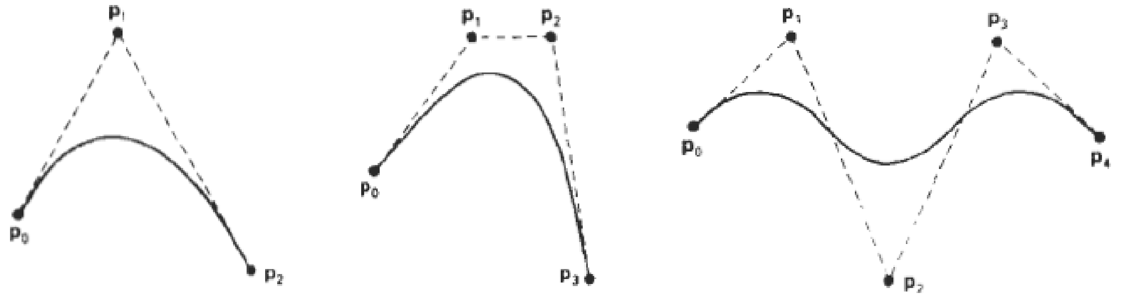


Fig: Examples of two-dimensional Bezier curves generated from three, four, and five control points

Properties of Bezier Curves

1. It always passes through initial and final control points. i.e $P(0) = p_0$ and $P(1) = p_n$.
2. Values of the parametric first derivatives of a Bezier curve at the end points can be calculated from control points as:

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

3. The slope at the beginning of the curve is along the line joining the first two points and slope at the end of curve is along the line joining last two points.
4. Parametric second derivative of a Bezier curve at end points are calculated as:

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

5. It lies with in the convex hull of the control points. This follows from the properties of Bezier blending functions: they are all positive and their sum is always 1.

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

Bezier surfaces

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:

$$\mathbf{P}(u, v) = \sum_{j=0}^{m} \sum_{k=0}^{n} \mathbf{p}_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

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

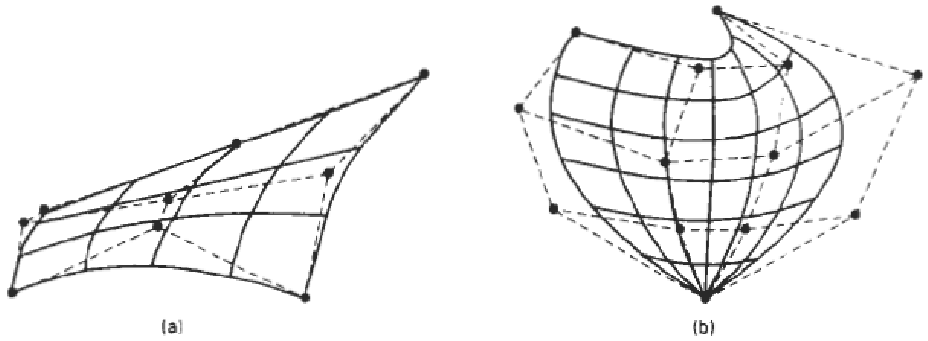


Fig: Bezier surfaces constructed tor (a) $m = 3$, $n = 3$, and (b) $m = 4$, $n = 4$. Dashed lines connect the control points

7. Octree Representation: (Solid-object representation)

This is the space-partitioning method for 3D solid object representation. This is hierarchical tree structures (octree) used to represent solid object in some graphical system. Medical imaging and other applications that require displays of object cross section commonly use this method. E.g. CT-scan

The octree encoding procedure for a three-dimensional space is an extension of an encoding scheme for two-dimensional space, called **quadtree encoding**. Quadtrees are generated by successively dividing a two-dimensional region (usually a square) into quadrants. Each node in the quadtree has four data elements, one for each of the quadrants in the region. If all pixels within a quadrant have the same color (a homogeneous quadrant) the corresponding data element in the node stores that color. In addition, a flag is set in the data element to indicate that the quadrant is homogeneous. Otherwise, the quadrant is said to be heterogeneous, and that quadrant is itself divided into quadrants

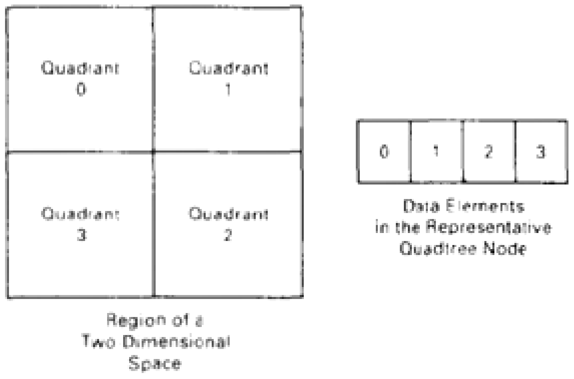

Quadrant 0
Quadrant 1
Quadrant 3
Quadrant 2
0
1
2
3
Data Elements in the Representative Quadtree Node
Region of a Two Dimensional Space

Fig: Region of a two-dimensional space divided into numbered quadrants and the associated quadtree node with four data elements

It provides a convenient representation for storing information about object interiors. An octree encoding scheme divides region of 3D space into octants and stores 8 data elements in each node of the tree. Individual elements are called volume element or voxels. When all voxels in an octant are of same type, this type value is stored in corresponding data elements. Any heterogeneous octants are subdivided into octants again and the corresponding data element in the node points to the next node in the octree. Procedures for generating octrees are similar to those for quadtrees: Voxels in each octant are tested, and octant subdivisions continue until the region of space contains only homogeneous octants. Each node in the octree can now have from zero to eight immediate descendants.

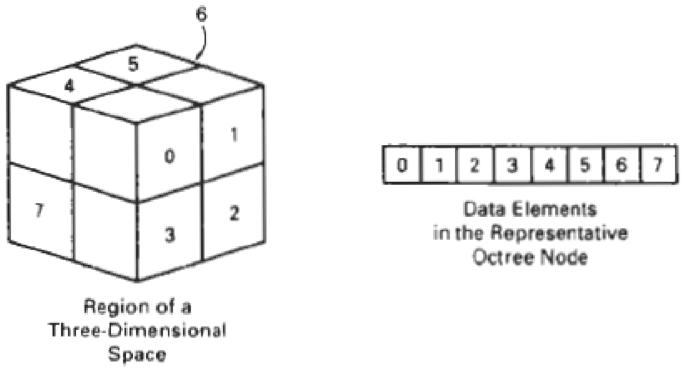

6
5
4
0
1
7
3
2
0
1
2
3
4
5
6
7
Data Elements in the Representative Octree Node
Region of a Three-Dimensional Space

Fig: Region of a three-dimensional space divided into numbered octants and the associated octree node with eight data elements

3D Viewing pipeline

The steps for computer generation of a view of 3D scene are analogous to the process of taking photograph by a camera. For a snapshot, we need to position the camera at a particular point in space and then need to decide camera orientation. Finally when we snap the shutter, the seen is cropped to the size of window of the camera and the light from the visible surfaces is projected into the camera film.

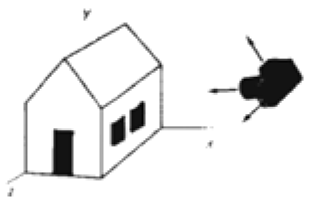


Fig: Photographing a scene involves selection of a camera position and orientation

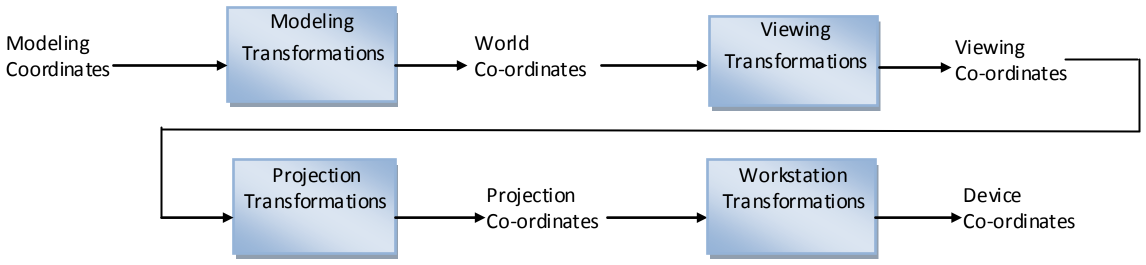


Fig: General three-dimensional transformation pipeline from modeling coordinates to final device coordinates

Projections

Once world co-ordinate description of the objects in a scene are converted to viewing co-ordinates, we can project the three dimensional objects onto the two dimensional view plane. There are two basic projection methods:

Parallel projection

In parallel projection, co-ordinates positions are transformed to the view plane along parallel lines.

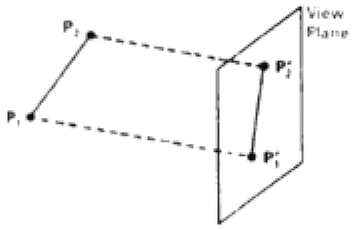


Fig: Parallel projection of an object to the view plane

Perspective projection

In perspective projection, objects positions are transformed to the view plane along lines that converge to a point called projection reference point (centre of projection). The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.

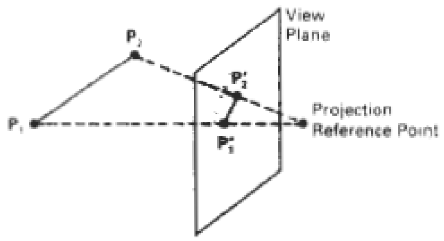


Fig: Perspective projection of an object to the view plane

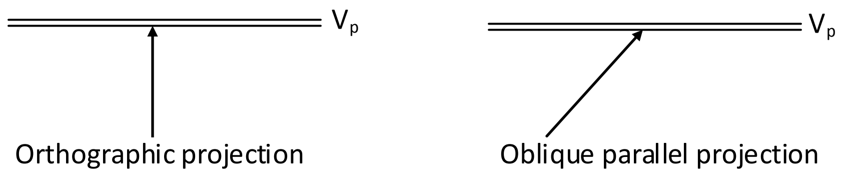
- A parallel projection preserve relative proportions of objects and this is the method used in drafting to produce scale drawings of three-dimensional objects. Accurate views of various sides of 3D object are obtained with parallel projection, but it does not give a realistic appearance of a 3D-object.
- A perspective projection, on the other hand, produces realistic views but does not preserve relative proportions. Projections of distance objects from view plane are smaller than the projections of objects of the same size that are closer to the projection place.

Both projection methods in detail:

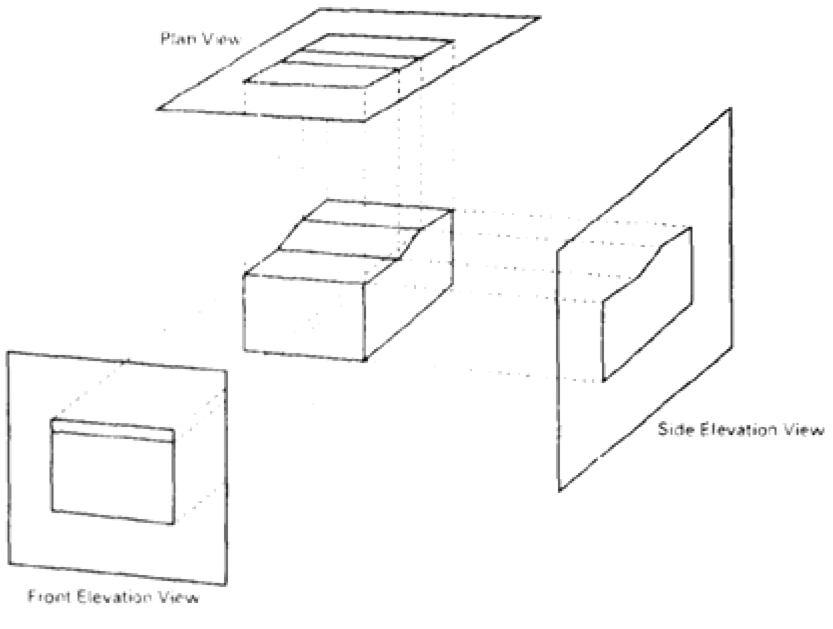
Parallel Projections

We can specify parallel projection with **projection vector** that specifies the direction of projection lines.

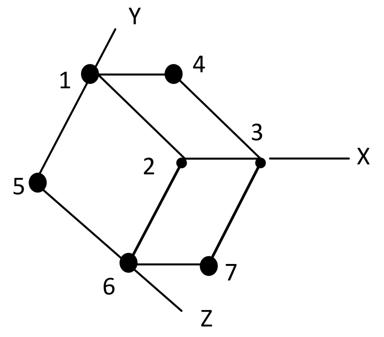
- When the projection lines are perpendicular to view plane, the projection is **orthographic parallel projection.**
- Otherwise it is **oblique parallel projection**.

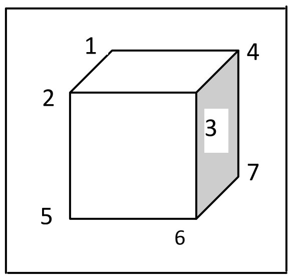


- Orthographic projections are most often used to produce the front, side, and top views of an object. Front, side, and rear orthographic projections of an object are called ***elevations;*** and a top orthographic projection is called a ***plain view.*** Engineering and Architectural drawings commonly employ these orthographic projections.



We can also form orthographic projections that display more than one face of an object. Such views are called **axonometric orthographic projections**. The most commonly used axonometric projection is the **isometric projection**.





View plane

The transformation equation for orthographic projection is

$x_p = x, \qquad y_p = y, \qquad z$ – Coordinate value is preserved for the depth information

Perspective projections

To obtain a perspective projection of a three-dimensional object, we transform points along projection lines that meet at a projection reference point. Suppose we set the projection reference point at position Z_{prp} along the Z_v axis, and we place the view plane at Z_{vp} as shown in fig:

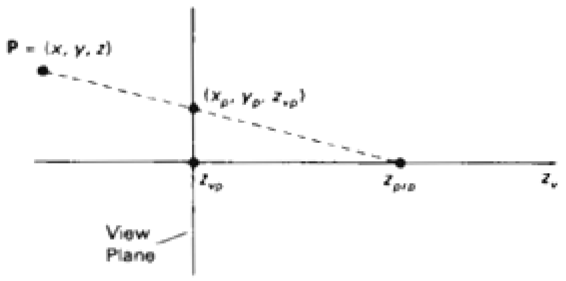


Fig: Perspective projection of a point $P(x, y, z)$ to position (x_p, y_p, z_{vp}) on the view plane.

We can write equations describing co-ordinates positions along this persspective projection line in parametric form as:

$$x' = x - xu$$
$$y' = y - yu$$
$$z' = z - (z - z_{prp})u$$

Where u takes values from 0 to 1 and coordinate position *(x', y', z')* represents any point along the projection line. If $u = 0$, we are at position $P = (x, y, z)$ and if $u = 1$, we have projection reference point $(0, 0, z_{prp})$.

On the view plane, $z' = z_{vp}$ and we can solve the z' equation for parameter *u* at this position along the projection line:

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

Substituting this value of *u* in equations for *x'* and *y'*

$$x_p = x - x\left(\frac{z_{vp} - z}{z_{prp} - z}\right) = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{dp}{z_{prp} - z}\right)$$

Similarly,

$$y_p = y(\frac{z_{prp} - z_{vp}}{z - z_{prp}}) = y(\frac{dp}{z_{prp} - z})$$

Where $dp = z_{prp} - z_{vp}$ is the distance of the view plane from projection reference point.

Using 3-D homogeneous Co-ordinate representation, we can write perspective projection transformation matrix as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/dp & \frac{z_{vp}}{z_{prp}/dp} \\ 0 & 0 & -1/dp & z_{prp}/dp \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In this representation the homogeneous factor is $h = \frac{z_{prp} - z}{dp}$ and the projection coordinates on the view plane are calculated from the homogeneous coordinates as,

$$x_p = {}^{x_h}\!\!/_{h}\,,\; y_p = {}^{y_h}\!\!/_{h}$$

where the original z-coordinate value would be retained in projection coordinates for visible-surface and other depth processing.

There are number of special cases for perspective transformation.
When z_{vp}= 0:

$$x_p = x(\frac{z_{prp}}{z_{prp} - z}) = x(\frac{1}{1 - {}^{z}\!/_{z_{prp}}})$$

$$y_p = y(\frac{z_{prp}}{z_{prp} - z}) = y(\frac{1}{1 - {}^{z}\!/_{z_{prp}}})$$

Some graphics package, the projection point is always taken to be viewing co-ordinate origin. In this ease, $z_{prp} = 0$

$$x_p = x(\frac{zvp}{z}) = x(\frac{1}{{}^{z}\!/_{zvp}})$$

$$yp = y(\frac{zvp}{z}) = y\ (\frac{1}{{}^{z}\!/_{zvp}})$$