Representing Curves and Surfaces

Curves and surface modeling are very rich topics in computer graphics. We will touch on three methods to approximate curves and curved surfaces

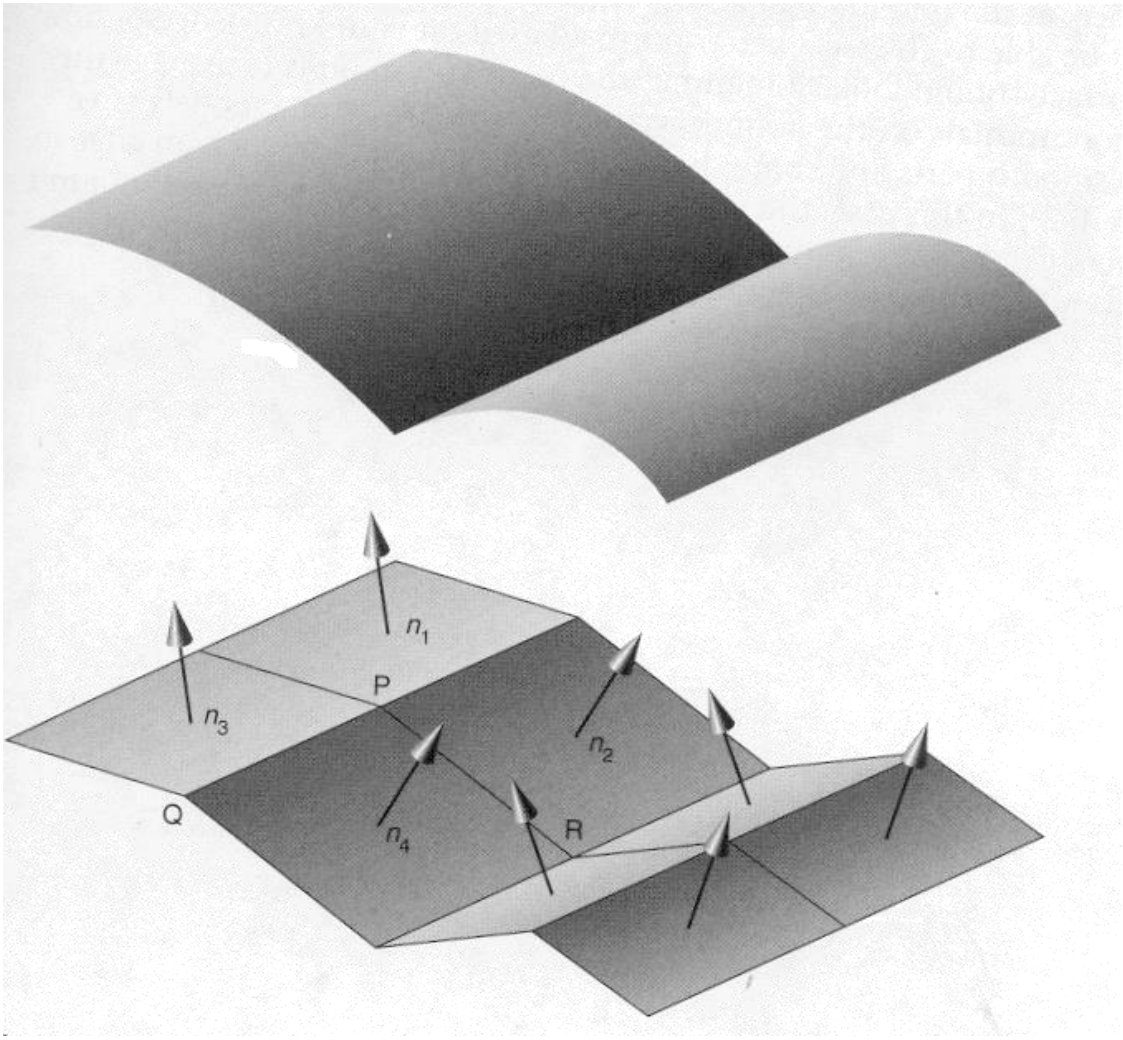
- polygonal meshes
- parametric curves and surfaces
- quadric surfaces.

Parametric curves are introduced as a preparatory step in understanding parametric surfaces.

OVERVIEW

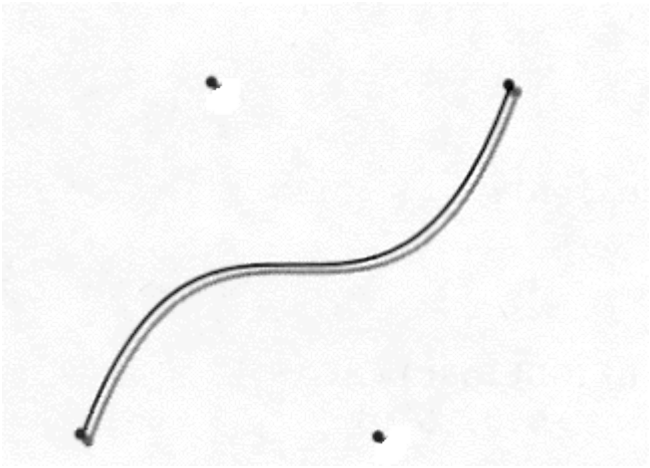
Polygon Mesh

A polygon mesh is a set of connected polygonally bounded planar surfaces. A mesh can be used to represent polygonal objects in a natural manner. A mesh can also be used to approximate a curved surface. Refer to figures below.



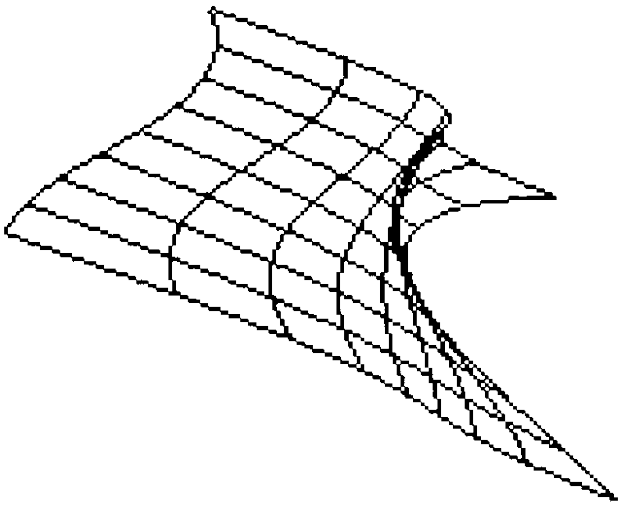
Parametric Curve

A parametric polynomial curve is used to represent a 3D curve. Points (x, y, z) on the curve are defined by three polynomials, one each for x, y, z. The polynomials are in one parameter, t. Typically, a third degree polynomial is used which produces a cubic curve. This figure depicts a Bezier curve.



Parametric Patches (surfaces)

A parametric bivariate polynomial surface patch is used to represent a 3D curved surface. Points on the patch are defined by three bivariate polynomials, one each for x, y, z. The boundaries of the patch are parametric curves. Typically, a third degree bivariate polynomial is used which produces a bicubic patch. This is a much more efficient representation of a curved surface than a polygon mesh (fewer parametric patches needed for accuracy). This figure depicts a Bezier surface.



Quadric Surfaces

A quadric surface is implicitly defined by an equation f(x, y, z) = 0, where f is a quadric polynomial in x,y,z. Example quadrics are circles, spheres, ellipsoids, etc.

Polygon Meshes

A mesh is a set of vertices, edges, and polygons. Each edge in the mesh is shared by at most two polygons. An edge connects two vertices. A polygon is a closed sequence of edges.

The key issue with meshes is one of representation and the ensuing space/time tradeoffs.

An explicit mesh representation

- requires that each polygon be specified by a list of vertices. Such a representation duplicates many vertices and does not identify shared vertices and edges. Rendering such a mesh would result in redrawing all shared edges. Thus, this representation is neither space nor time efficient.

A vertex pointer representation

- requires that each polygon be specified as a list of pointers (indices) into a list of vertices. Each vertex is stored exactly once. The space overhead is the list of pointers maintained for each polygon. The representation identifies shared vertices but still does not identify shared edges so redrawing occurs. This representation is space efficient/time inefficient.

Example

VList = {V1, V2, V3, V4} where Vi = (xi, yi, zi) and Polygon1 = {1, 2, 4}, Polygon2 = {2, 3, 4}, ie. Two triangles are specified with a shared edge incident on vertices V2, V4.

End Example

An edge list representation

- requires that each polygon be specified as a list of pointers (indices) into a list of edges. Each edge is stored exactly once. An edge is composed of its two vertices and the polygon to which it belongs. Each vertex is stored exactly once. The space overhead is the list of edges and the list of pointers for each polygon. To render an

outlined mesh, it is necessary to render only the edge list. To render filled polygons is equally efficient. Shared edges are not redrawn. This representation is moderately space efficient and very time efficient.

Example

VList'={V1, V2, V3, V4} where Vi = (xi, yi, zi) and EList" ={E1, E2, E3, E4, E5} where E1 = {1', 2', P1, NULL}, E2 = {2', 4', P1, P2}, E3 = {1', 4', P1, NULL}, E4 = {2', 3', P2, NULL}, E5 = {3', 4', P2, NULL}, and P1 = {1", 2", 3"}, P2 = {2", 4", 5"}. Two triangles are specified with a shared edge E2 incident on vertices V2, V4.

End Example

Because of the complexity of representing meshes correctly, consistency checks are frequently carried out for a constructed mesh.

Issues of concern are : each edge is used, each vertex is incident on two edges, each polygon is closed, the mesh is connected, no holes in mesh, each polygon is planar, etc. The edge list representation is the most useful for such checks.

One neat application of polygonal meshes is the fractal generation of terrains such as mountain ranges. Troy Kozee's I.S. investigated this topic. The mesh is generated by starting with a unit square in the x-z plane. Recursive bilinear interpolation is used to construct an n x n grid of vertices within the unit square. The depth of the recursion defines the granularity of the mesh (the deeper the recursion the more polygonal patches generated). The y-coordinate of each vertex is then modulated using a fractal function to offset its value from the origin. Many kinds of functions can be applied to modulate y, but Troy used Fractal Brownian motion which produces very "noisy" results, ie. rugged, peaky mountains like the Rockies.

Parametric Cubic Curves

Parametric curves are represented by three equations

$x = x(t), y = y(t), z = z(t)$

which allows a curved surface to be approximated by a piecewise polynomial curve. The three equations above are cubic polynomials in the parameter t.

Cubic polynomials are used because this is the lowest order polynomial which allows non-planar curves to be expressed.

What is a non-planar curve? Why is a cubic polynomial the lowest order polynomial for non-planar curves?

- If we take the derivative of a quadratic polynomial at point (x, y, z), what do we get?

    - The slope of the tangent line at (x, y, z). So the "movement" along the curve at (x, y, z) is linear

- If we take the derivative of a cubic polynomial at point (x, y, z), what do we get?
    - A quadratic polynomial. So the "movement" along the curve at (x, y, z) is non-linear and may cross itself

Each cubic polynomial has the general form :

$at^3 + bt^2 + ct + d$

The four unknown coefficients (a, b, c, d) are solved for using four knowns (which might be the two endpoints of the curve and the derivatives of the endpoints or the *unit tangent vectors at the endpoints*).

The cubic polynomials required to specify a curved segment Q(t) are (EQS1):

$x(t) = axt^3 + bxt^2 + cxt + dx$

$y(t) = ayt^3 + byt^2 + cyt + dy$

$z(t) = azt^3 + bzt^2 + czt + dz$

Because we are dealing with finite curves, t is in the interval [0,1].

We can more compactly represent Q(t) as T*C where T is the row vector :

$[\ t^3 \ \ t^2 \ \ t \ \ 1]$

and C is the 4 X 3 matrix :

ax	ay	az
bx	by	bz
cx	cy	cz
dx	dy	dz

We can talk about the geometric continuity of two curves that meet at a join point. The continuity is expressed as a relation between the tangent vectors of the two curves. The tangent vector of a cubic curve is Q`(t) or T` * C =

$[\ 3t^2 \ \ 2t \ \ 1 \ \ 0\ ] * C$

- Go continuity exists if two curves join at a join point.

- G1 continuity exists if the direction of the two tangent vectors is equal.
- C1 continuity exists if the direction and magnitude of the two vectors is equal. Generally, C1 continuity implies G1 continuity
- Cn continuity exists if the two vectors are equal through the nth derivative. Refer to figures

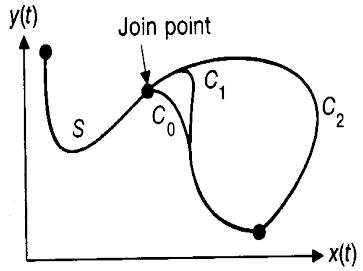


Fig. 11.8 Curve segment S joined to segments C_0, C_1, and C_2 with the 0, 1, and 2 degrees of parametric continuity, respectively. The visual distinction between C_1 and C_2 is slight at the join, but obvious away from the join.

below.

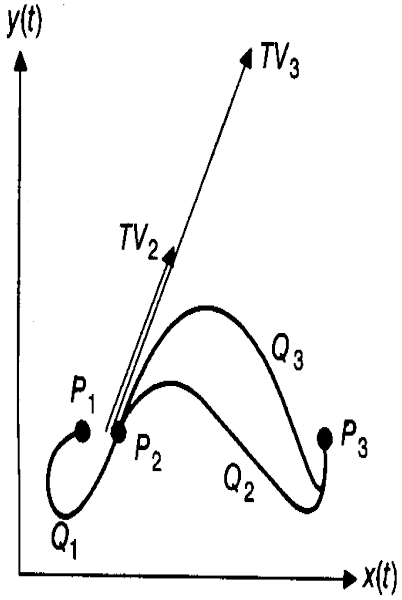


Fig. 11.9 Curve segments Q_1, Q_2, and Q_3 join at the point P_2 and are identical except for their tangent vectors at P_2. Q_1 and Q_2 have equal tangent vectors, and hence are both G^1 and C^1 continuous at P_2. Q_1 and Q_3 have tangent vectors in the same direction, but Q_3 has twice the magnitude, so they are only G^1 continuous at P_2. The larger tangent vector of Q_3 means that the curve is pulled more in the tangent-vector direction before heading toward P_3. Vector TV_2 is the tangent vector for Q_2, TV_3 is that for Q_3.

A curve Q(t) is defined by constraints on endpoints, tangent vectors, and continuity between curve segments.

The three types of cubic curves are

- hermite defined by two endpoints and two tangent vectors
- bezier defined by two endpoints and two other points that control the endpoint tangent vectors
- splines which are defined by 4 endpoints.

To see how the coefficients of EQS1 can depend on 4 unknowns rewrite

Q(t) = T*C as T*M*G where the coefficient matrix C = M*G.

M is a 4 x 4 basis matrix and G is a 4 element column vector of geometric constraints (endpoints or tangent vectors) called the geometry vector.

The elements of M and G are constants. Both M and G are different for each type of curve.

$$Q(t) = [x(t) \quad y(t) \quad z(t)] = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} \quad (11.9)$$

We now have to determine how to compute M for each type of curve.

Hermite Curves

The hermite curve is determined by constraints on the two given endpoints, P1, P4, and by constraints on the two tangent vectors, R1, R4, to the given endpoints.

To find the basis matrix, Mh, for a hermite curve

- construct four equations, one for each constraint, and solve for the four unknowns. The process is stated below for x(t) only, must be carried out for y(t) and z(t).

1.The parametric equation for x(t)

$x(t) = axt^3 + bxt^2 + cxt + dx = T*Cx = T*Mh*Gx$

2. The geometry vector Gx captures constraints for the x coordinate of a point on the curve.

Remember that t is in the interval [0,1]. Substituting t = 0 into the parametric equations will give us point P1 on the curve. Substituting t = 1 into the equations will give us point P4 on the curve. Thus we constrain the curve to be finite and end on the given points.

T row vector

$[t^3 \;\; t^2 \;\; t \;\; 1]$

Substitute t = 0 into the row vector for T to get, x(0) = P1x = | 0 0 0 1| *Mh*Gx. This is the constraint on x when t = 0.

Substitute t = 1 into the vector to get x(1) = P4x = |1 1 1 1|*Mh*Gx. This is the constraint on x when t = 1.

3. Determine the tangent vectors at P1 and P4 by calculating x`(t) =

$[3t^2 \;\; 2t \;\; 1 \;\; 0]$

Substitute t = 0 into x`(t) to find x`(0) = R1x = |0 0 1 0|*Mh*Gx. This is the constraint on the tangent vector at P1 when t = 0.

Substitute t = 1 into x`(t) to find x`(1) = R4x = |3 2 1 0| *Mh*Gx. This is the constraint on the tangent vector at P4 when t = 1.

4. The four constraints for Gx =

P1x = 0 0 0 1

P4x = 1 1 1 1 * Mh * Gx

R1x = 0 0 1 0

R4x = 3 2 1 0

5. Notice that the above equation is Gx = M*Mh*Gx where M is the expanded 4 X 4 matrix in step 4. The only way this equation can hold is if the basis matrix, Mh, is the inverse of M. We now have the method to solve for the basis matrix which is not done here, but is simply given.

Invert M to get Mh =

2 -2 1 1

-3 3 -2 -1

0 0 1 0

1 0 0 0

If we wish to carry out the Hermite curve evaluation in matrix form then the structure is

$$
\begin{array}{lll}
x(t) = & 2\ -2\ 1\ 1 & P1x\ P1y\ P1z\ P1w \\
y(t) = & [t^3\ t^2\ t1] * \begin{matrix} -3 & 3 & -2 \\ -1 \end{matrix} * & P4x\ P4y\ P4z\ P4w \\
z(t) = & 0\ 0\ 1\ 0 & R1x\ R1y\ R1z\ R1w \\
w(t) = & 1\ 0\ 0\ 0 & R4x\ R4y\ R4z\ R4w
\end{array}
$$

T * Mh gives us the four Hermite blending functions, b0 = ($2t^3 - 3t^2 + 1$), b1 = ($-2t^3 + 3t^2$), b2 = ($t^3 - 2t^2 + t$) and b3 = ($t^3 - t^2$)

and

$$
\begin{matrix} [& b0 & b1 \\ b2 & b3] \end{matrix} * \begin{matrix} P1x\ P1y\ P1z\ P1w \\ P4x\ P4y\ P4z\ P4w \\ R1x\ R1y\ R1z\ R1w \\ R4x\ R4y\ R4z\ R4w \end{matrix}
$$

gives us Q(t) = b0*P1 + b1*P4 + b2*R1 + b3*R4

a point (x(t), y(t), z(t))on the curve, Q(t), is given by a weighted sum for each coordinate. For example

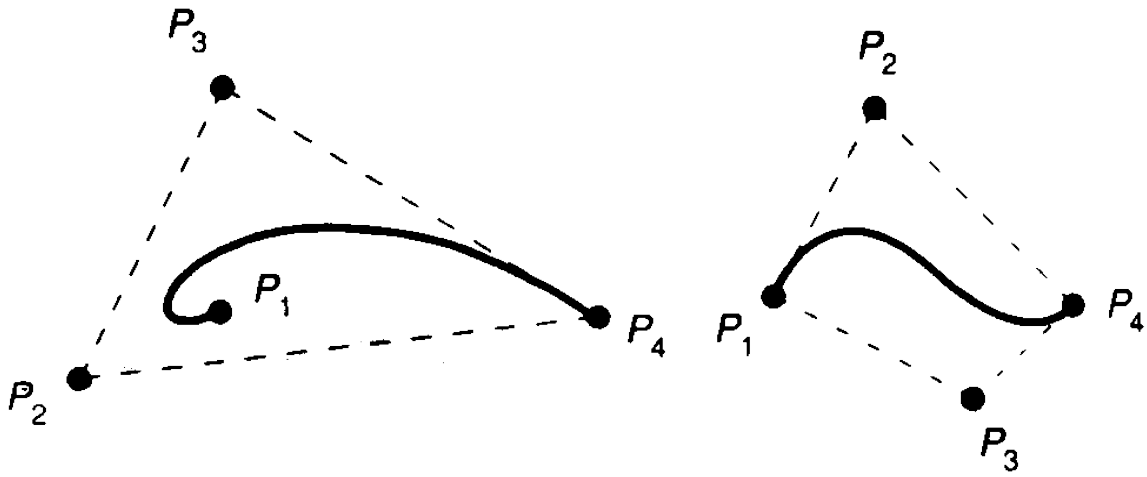
x(t)	=	b0*P1x	+	b1*P4x	+	b2*R1x	+	b3*R4x
y(t)	=	b0*P1y	+	b1*P4y	+	b2*R1y	+	b3*R4y
z(t)	=	b0*P1z	+	b1*P4z	+	b2*R1z	+	b3*R4z

w(t) is ignored.

Bezier Curves

Bezier curves are specified using the two endpoints of the curve and two points not on the curve. The 4 points are referred to as control points. The control points define the convex hull of the curve where we might think of the hull as a bounding box that completely contains the curve. Note that all four control points do not have to lie on the convex hull boundary. Refer to figures below.

.



Let us refer to P1, P4 as the endpoints of the curve. P2, P3 are the control points which are used to define the tangent vectors at P1 and P4. The tangent vectors are determined by the vectors P2 P1 and P4 P3. These two vectors are related to the tangent vectors as follows :

R1 = Q`(0) = 3 (P2-P1) and R4 = Q`(1) =3(P4-P3), EQs2.

The constant 3 is used to ensure that the curve has a constant velocity from P1 to P4 (refer to the 1st derivative of Q(t) to see why 3 is used).

The Bezier geometry vector Gb is merely :

P1

P2

P3

P4

The matrix Mhb that defines the relation between the Hermite geometry vector Gh and the Bezier geometry vector Gb (Gh = Mhb * Gb) is

	P1		1	0	0	0	P1	
Gh=	P4	=	0	0	0	1	* P2	=Mhb*Gb
	R1		-3	3	0	0	P3	
	R4		0	0	-3	3	P4	

To find the Bezier basis matrix the following substitution is performed using the definition of a hermite curve:

Q(t) = T*Mh*Gh = T*Mh*(Mhb*Gb) [from above]

= T*(Mh*Mhb)*Gb = T*Mb*Gb. We know the basis matrix, Mh, for the hermite curve and Mhb is specified above so just carry out the multiplication to find Mb =

-1 3 -3 1

3 -6 3 0

-3 3 0 0

1 0 0 0

and the curve Q(t) = T*Mb*Gb is

$Q(t) = (1-t)^3P1 + 3t(1-t)^2P2 + 3t^2(1-t)P3 + t^3P4$

Note that the 4 weighting functions, $bo = (1-t)^3$, $b1 = 3t(1-t)^2$, $b2 = 3t^2(1-t)$ and $b3 = t^3$ are formed by

$$[\, t^3 \; t^2 \; t \; 1 \,] * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & -0 & 0 \end{bmatrix}$$

and are referred to as the Bernstein Polynomials. Note that $(1-t)^3 = -t^3 +3t^2 -3t +1$. This is what we get when we multiply T by column one of Mb above. The other weighting functions are produced similarly.
