

Surface Modelling

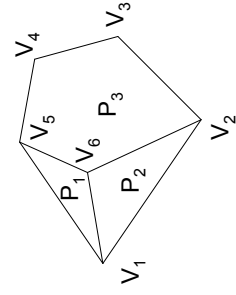
- Several methods are available to model surfaces.
 - Polygon meshes.
 - Surface is represented as a set of bounded planar surfaces.
 - Parametric polynomial curves.
 - Surface is defined by 3 equation in a parameter, t , one for each principal axis (x , y , z).
 - Parametric bivariate polynomial surface patches.
 - Similar to parametric polynomial curves but this time each equation has two variables.
 - Quadratic surfaces.
 - Surface is implicitly defined by an equation $f(x, y, z) = 0$.
- We will discuss polygon meshes and a particular form of parametric polynomial curves, the Bezier curve.

Polygon Meshes

- A polygon mesh is a set of bounded polygons such that each edge is shared by at most 2 polygons.
- Polygon meshes can be represented in several ways.
 - Explicit representation
 - Surface is a set/list of polygons.
 - Each polygon is represented by a list of vertices.
 - The order in which the vertices are stored defines the orientation of the polygon.
 - Inefficient use of memory as shared vertices are duplicated. Also no record of modified vertices or edges. If a vertex is modified the complete set of polygons need to be searched to see which polygons are affected.

Polygon Meshes (contd.)

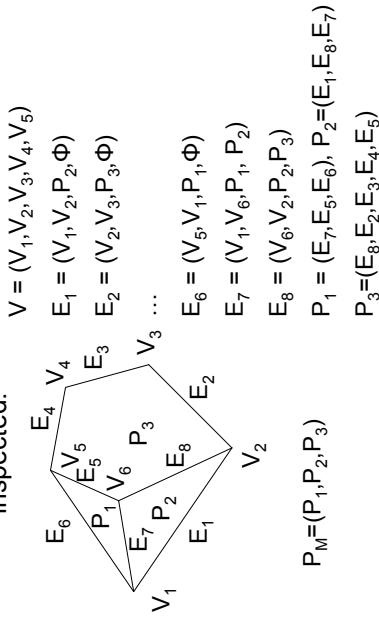
- Pointers to a vertex list
 - Each polygon is defined by a list of indices into a list of vertices.
 - More efficient memory usage as shared vertices are not duplicated.
 - A change to a vertex is immediately reflected in each polygon that uses that vertex.
 - Edges may still be duplicated, hence an edge that is shared between 2 polygon will be drawn twice.
 - Finding the 2 polygons that share an edge is still requires an $\text{Mlog}_2 N$ search (at best).



$$\begin{aligned}
 V &= (V_1, V_2, V_3, V_4, V_5) \\
 &= ((x_1, y_1, z_1), \dots, (x_5, y_5, z_5)) \\
 P_M &= (P_1, P_2, P_3) \\
 P_1 &= (1, 6, 5) \\
 P_2 &= (1, 2, 6) \\
 P_3 &= (6, 2, 3, 4, 5)
 \end{aligned}$$

Polygon Meshes (contd.)

- Pointers to an edge list
 - The polygon mesh is represented by a list of vertices, a list of edges and a set/list of polygons.
 - Each polygon is defined by its edges.
 - Each edge is defined by two indices in to the vertex list and the two polygons that share the edge, i.e. (V_1, V_2, P_1, P_2) . If the edge is used by only one polygon then either P_1 or P_2 is null.
 - Determining which edge is incident on a given vertex still requires all edges to be inspected.



$$\begin{aligned}
 V &= (V_1, V_2, V_3, V_4, V_5) \\
 E_1 &= (V_1, V_2, P_2, \Phi) \\
 E_2 &= (V_2, V_3, P_3, \Phi) \\
 &\dots \\
 E_6 &= (V_5, V_1, P_1, \Phi) \\
 E_7 &= (V_1, V_6, P_1, P_2) \\
 E_8 &= (V_6, V_2, P_2, P_3) \\
 P_1 &= (E_7, E_5, E_6) \quad P_2 = (E_1, E_8, E_7) \\
 P_3 &= (E_8, E_2, E_3, E_4, E_5)
 \end{aligned}$$

$$P_M = (P_1, P_2, P_3)$$

Polygon Meshes (contd.)

- When working with polygon meshes, it is important to ensure that the mesh representation is consistent.
 - All polygons are closed.
 - Each edge is used at least once and not more than twice.
 - Each vertex is referenced by at least two edges.
- Plane equations
 - If a polygon is defined by more than 3 vertices, the polygon may be non-planar.
 - If the polygon is defined by 3 vertices it is planar and defined by:
 $Ax + By + Cz + D = 0$
 where $[A, B, C]^T$ is the normal to the plane.
 - The normal is also defined by $P_1P_2 \times P_2P_3$ (or $P_2P_3 \times P_2P_1$ or $P_3P_1 \times P_3P_2$). Once you have A, B and C you can use any vertex to calculate D.

Parametric Cubic Curves

- A **spline curve** is defined by a set of **control points**. These control points are either:
 - **Interpolating control points**: The curve passes through these control points.
 - **Approximating control points**: These control points exert an influence of the curve.
- The parametric equation for a line is:

$$\left. \begin{aligned} x &= x_0(1-t) + x_1t \\ y &= y_0(1-t) + y_1t \end{aligned} \right\} 0 \leq t \leq 1$$

\Rightarrow

$$x = x_0 + (x_1 - x_0)t$$

$$y = y_0 + (y_1 - y_0)t$$

\Rightarrow

$$x = a_0 + a_1t$$

$$y = b_0 + b_1t$$

Parametric Cubic Curves (contd.)

- Extending this idea to cubic curves we have:

$$\left. \begin{aligned} x &= a_0 + a_1t + a_2t^2 + a_3t^3 \\ y &= b_0 + b_1t + b_2t^2 + b_3t^3 \end{aligned} \right\} 0 \leq t \leq 1$$
- Substituting $t=0$ at $(x,y)=(x_0,y_0)$ and $t=1$ at $(x,y) = (x_1,y_1)$ we get 4 equations with 8 unknowns.

$$x_0 = a_0$$

$$y_0 = b_0$$

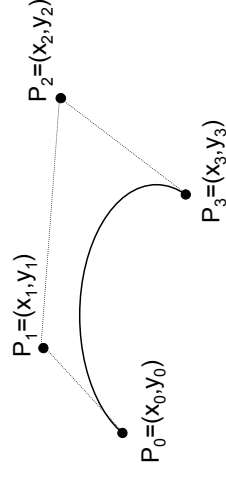
$$x_1 = a_0 + a_1 + a_2 + a_3$$

$$y_1 = b_0 + b_1 + b_2 + b_3$$

- We need more information to calculate the coefficients. When we join 2 curves, we usually want to control the slope of the curve at the join.
 - Curves that join have **geometric continuity**, G^0 . Curves whose tangents vectors have equal magnitude and direction at the join have **parametric continuity**, C^1 .

Bezier Curves

- A 4 point Bezier curve is defined by 4 control points; P_0, P_1, P_2, P_3 .
 - P_0 and P_3 are interpolating control points through which the curve must pass.
 - P_1 and P_2 are approximating control points that control the slope of the curve at P_0 and P_3 respectively.
- The slope of P_0P_1 defines the tangent gradient at P_0 and the slope of P_2P_3 defines the tangent gradient at P_3 .



Bezier Curves (contd.)

- The parametric equations for a 4 point Bezier curve is:

$$x = x_0(1-t)^3 + 3x_1(1-t)^2t + 3x_2(1-t)t^2 + x_3t^3$$

$$y = y_0(1-t)^3 + 3y_1(1-t)^2t + 3y_2(1-t)t^2 + y_3t^3$$

- Differentiating yields:

$$\frac{dx}{dt} = -3x_0(1-t)^2 + 3x_1(1-t) - 6x_2(1-t)t + 6x_2(1-t)t^2 - 3x_2t^2 + 3x_3t^2$$

$$\frac{dy}{dt} = -3y_0(1-t)^2 + 3y_1(1-t) - 6y_2(1-t)t + 6y_2(1-t)t^2 - 3y_2t^2 + 3y_3t^2$$

- At $t = 0$

$$\frac{dx}{dt} = -3x_0 + 3x_1$$

$$\frac{dy}{dt} = -3y_0 + 3y_1$$

$$\text{So } \frac{dy}{dx} = \frac{-3y_0 + 3y_1}{-3x_0 + 3x_1} = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

Bezier Curves (contd.)

- Similarly the slope of the curve at P_3 is:

$$\frac{dy}{dx} = \frac{-3y_2 + 3y_3}{-3x_2 + 3x_3} = \frac{(y_3 - y_2)}{(x_3 - x_2)}$$

- The slope of the line segments P_0P_1 and P_2P_3 define the tangents at P_0 and P_3 respectively.
- The general form of an n -point Bezier curve is:

$$x(t) = \sum_{r=0}^{n-1} B_r x_r (1-t)^{n-r-1} t^r$$

$$y(t) = \sum_{r=0}^{n-1} B_r y_r (1-t)^{n-r-1} t^r$$

where

$$B_r = \frac{n-r}{r} B_{r-1} \text{ and } B_0 = 1$$

Bezier Curves (contd.)

- Code to implement a Bezier curve.

```
void DrawBezier (int n, float[] x, float[] y)
{
    int B = new int [n];
    int i, r;
    float xCoeff = new float [n];
    float yCoeff = new float [n];
    float t, xt, yt, tToPowerR, tCompToPower;

    B[0] = 1;
    xCoeff[0] = B[0] * x[0];
    yCoeff[0] = B[0] * y[0];
    for (r = 1; r < n; r++)
    {
        B[r] = B[r-1] * (float)(n - r) / r;
        xCoeff[r] = B[r] * x[r];
        yCoeff[r] = B[r] * y[r];
    }
}
```

Bezier Curves (contd.)

```
for (i = 1; i < 100; i++) // from p0 to pn in 100 steps
{
    t = 0.01 * i;
    tToPowerR = 1;
    tCompToPower = 1 - t;
    for (r = 2; r < n; r++)
    {
        tCompToPower = tCompToPower * (1 - t);
    }
    xt = 0.0;
    yt = 0.0;
    for (r = 0; r < n; r++)
    {
        xt = xt + xCoeff[r] * tToPowerR * tCompToPower;
        yt = yt + yCoeff[r] * tToPowerR * tCompToPower;
        tToPowerR = tToPowerR * t;
        tCompToPower = tCompToPower / (1 - t);
    }
    // plot (xt, yt)
}
}
```

Bezier Surfaces

- Bezier curves can be extended into Bezier surfaces.
 - First add a $z(t)$ equation to make the Bezier curve into a Bezier space curve (a curve in 3 dimensions).

$$z(t) = \sum_{r=0}^{n-1} B_r z_r (1-t)^{n-r-1} t^r$$

- Two orthogonal space curves define a curved surface.

$$P(t, u) = \sum_{s=0}^{m-1} \sum_{r=0}^{n-1} P_{(s,r)} B_s (1-u)^{m-s-1} u^s B_r (1-t)^{n-r-1} t^r$$

