

Spline Drawing & Polygon Filling

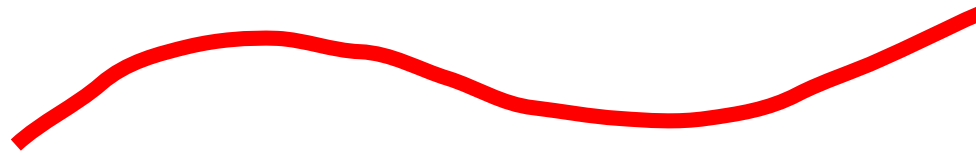
Eric C. McCreath

School of Computer Science
The Australian National University
ACT 0200 Australia

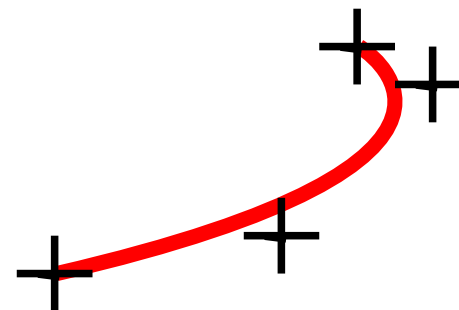
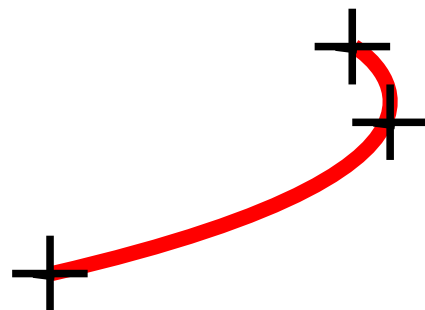
ericm@cs.anu.edu.au

- Spline Drawing
 - curves
 - cubic-splines (Natural, Hermite, and Cardinal)
 - Bezier
 - B-Spline
- Polygons
 - Interior points
 - Filling Polygons
- Intersecting line segments

- Spline curves are composite curves formed with polynomial sections.



- Spline Surfaces are two sets of orthogonal spline curves.
- Control points are used to describe the curve. These point are either:
 - interpolate – curve goes through the control point, or
 - approximate – direct the curve but the curve does not pass through it.

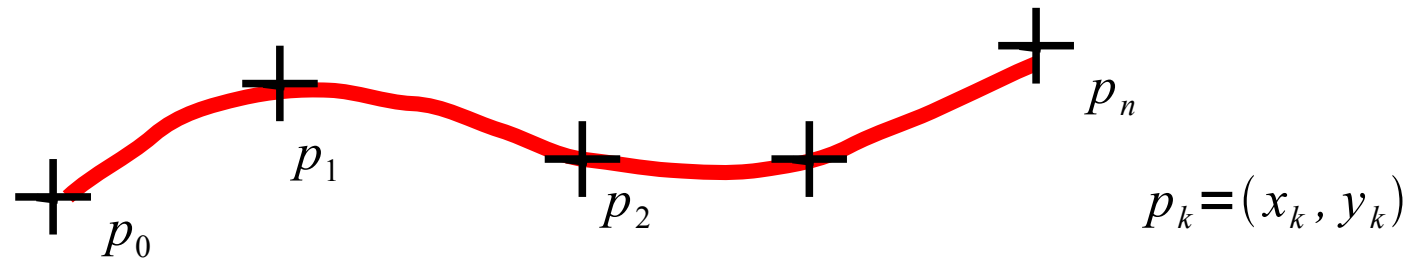


- Curves can be described using parametric functions.

$$\begin{aligned}x &= f_x(u) \\y &= f_y(u) \\u_{start} &\leq u \leq u_{end}\end{aligned}$$

- Curves can be characterised whether they are:
 - zero-order parametric continuous,
 - first-order parametric continuous, and
 - second-order parametric continuous.

- Often piecewise cubic parametric function can be used to describe a spline.



- If we had $n+1$ control points then we would end up with n different cubic parametric functions. These functions are used to describe the curves between the interpolated control points.

$$\begin{aligned} f_{x,k}(u) &= a_{x,k} u^3 + b_{x,k} u^2 + c_{x,k} u + d_{x,k} \\ f_{y,k}(u) &= a_{y,k} u^3 + b_{y,k} u^2 + c_{y,k} u + d_{y,k} \end{aligned} \quad 0.0 \leq u \leq 1.0$$

- Clearly you require:

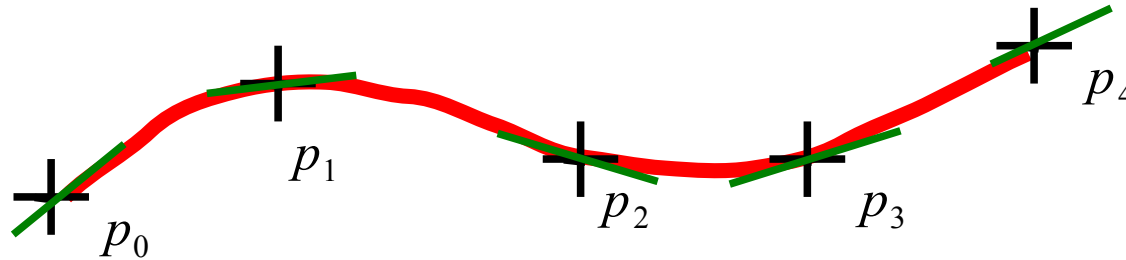
$$x_k = f_{x,k}(0.0) \wedge x_{k+1} = f_{x,k}(1.0)$$

$$y_k = f_{y,k}(0.0) \wedge y_{k+1} = f_{y,k}(1.0)$$

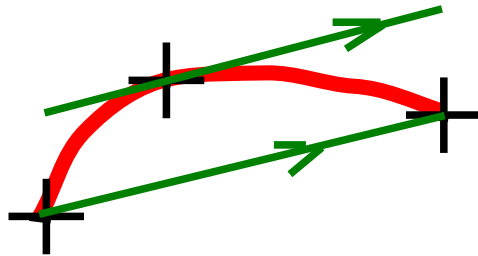
- **Natural cubic splines** have adjacent curve sections with the same 1st and 2nd derivatives.
- Assuming we have n sections and $n+1$ control points. Then in the x dimension we have:
 - $4n$ variables to calculate that describe the curve,
 - $2n$ equations from end points, $n-1$ equations from asserting 1st derivatives are the same at boundaries, $n-1$ equations from asserting 2nd derivatives are the same at boundaries.
 - => this leaves us with 2 degrees of freedom still to be determined!
- Two common approaches:
 - set 2nd derivatives to 0 at the ends, or
 - add points p_{-1} and p_{n+1} .

Note, changing one point effects the entire curve!

- **Hermite Splines** are piecewise cubic polynomial splines where the tangents at the interpolated control points are specified.



- **Cardinal Splines** are like Hermite splines but the tangents are not explicitly specified rather they are calculated from adjacent control points.



- Linear Bezier Curve (just a line segment)

$$B(u) = (1-u)P_0 + uP_1, 0 \leq u \leq 1$$

- Quadratic Bezier Curve

$$B(u) = (1-u)^2 P_0 + 2(1-u)u P_1 + u^2 P_2, 0 \leq u \leq 1$$

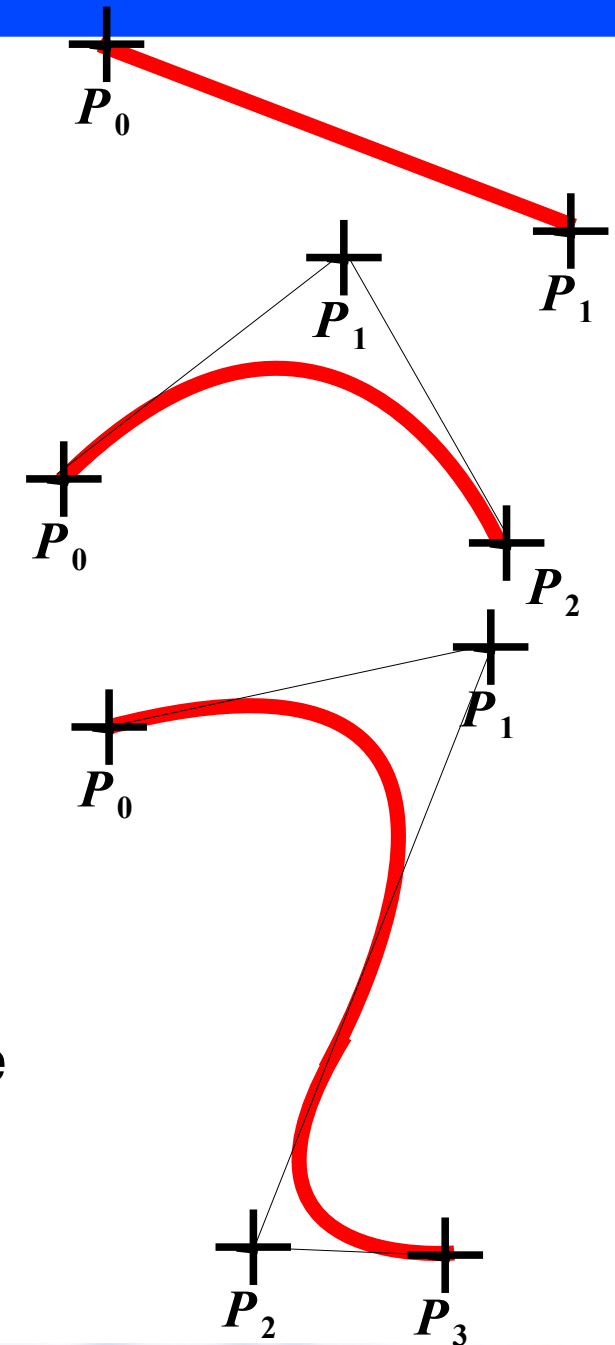
- Cubic Bezier Curve

$$B(u) = (1-u)^3 P_0 + 3(1-u)^2 u P_1 + 3(1-u)u^2 P_2 + u^3 P_3, \\ 0 \leq u \leq 1$$

- Generalised Bezier Curve

$$B(u) = \sum_{i=0}^n \binom{n}{i} (1-u)^{n-i} u^i P_i, 0 \leq u \leq 1$$

- Bezier curves can be joined together in a piecewise fashion to form a Bezier spline.
- Affine transformations on the control points transform the curve similarly.



- B-Splines are similar to Bezier curves in that they use a number of approximate control points. One advantage of B-Splines over Bezier curves is that the degree of the polynomial can be controlled independently from the number of control points. Also changes in a control point only effects the curve in that neighbourhood. However B-Splines are more complex to program then Bezier curves.
- With n control points, and setting d to the degree of the polynomials. The points on the curve are calculated via a blending function:

$$\mathbf{B}(u) = \sum_{i=0}^{n-1} \mathbf{P}_i B_{i,d}(u), u_{min} \leq u \leq u_{max}$$

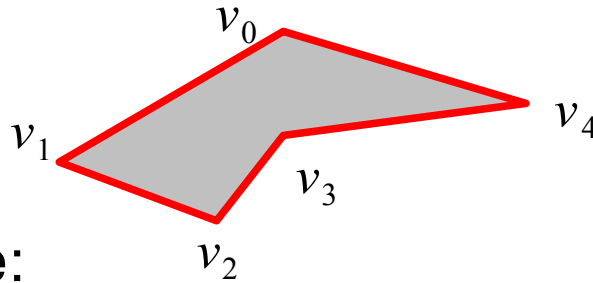
- $n+d-1$ real values are defined. These are called the knots:

$$u_0 \leq u_1 \leq u_2 \leq \dots \leq u_{n+d-2}$$

- The blending function is then defined recursively:

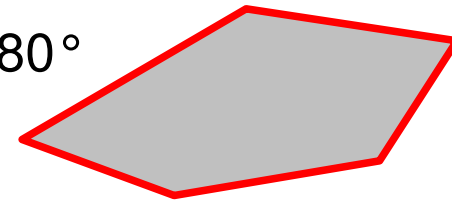
$$B_{i,0}(u) = \begin{cases} 1 & , \text{if } u_k \leq u < u_{k+1} \\ 0 & , \text{otherwise} \end{cases} \quad B_{k,d}(u) = \frac{u - u_k}{u_{k+d} - u_k} B_{k,d-1}(u) + \frac{u_{k+d+1} - u}{u_{k+d+1} - u_{k+1}} B_{k+1,d-1}(u)$$

- Polygons can be described by a list of vertices (normally counter-clockwise).

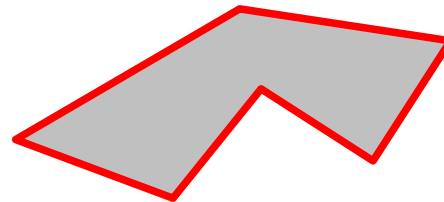


- Polygons can be:

- **convex** – internal angles all less than 180°

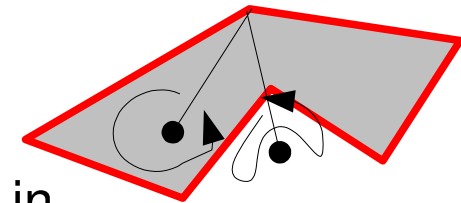
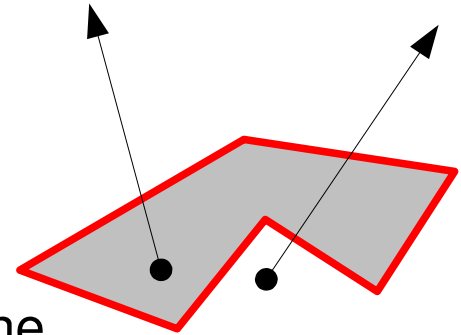


- **concave** – not convex



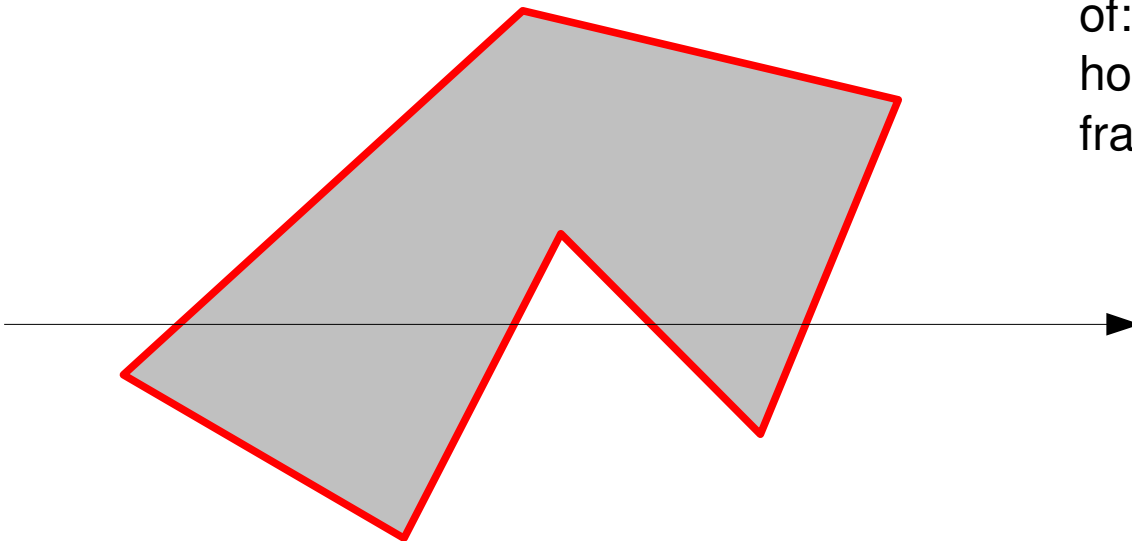
- When vertices are collinear or have repeated positions then the polygon is described as **degenerate**.
- To simplify rendering: concave polygons may be broken up into convex polygons, and convex polygons may be broken up into triangles.

- There are two ways of defining/identifying if a point is in the interior of a polygon or not:
 - odd-even rule – draw any line from the point in question to outside the coordinate extents of the polygon and count the number of edges of the polygon are crossed. If the count is even then the point lies outside the polygon, otherwise if the count is odd then the point is considered to be an interior point.
 - non-zero winding number – the winding number is calculated by counting the number of times the perimeter of the polygon travels around the point in a counter-clockwise direction. If this number is non-zero then the point is considered to be an interior point.



- The odd-even rule can be used for filling polygons.
 - For each scan line:
 - calculate intersections
 - sort in terms of their x values
 - use odd-even rule for filling

A few special cases to be careful of: the scan line crossing vertices, horizontal edges, fractional x value and determining interior.



- To determine if two line segments intersect one can use a parametric form for the line.

$$v_A = p_{A,1} - p_{A,0}$$

$$v_B = p_{B,1} - p_{B,0}$$

$$P_A(u) = u v_A + p_{A,0}, 0 \leq u \leq 1$$

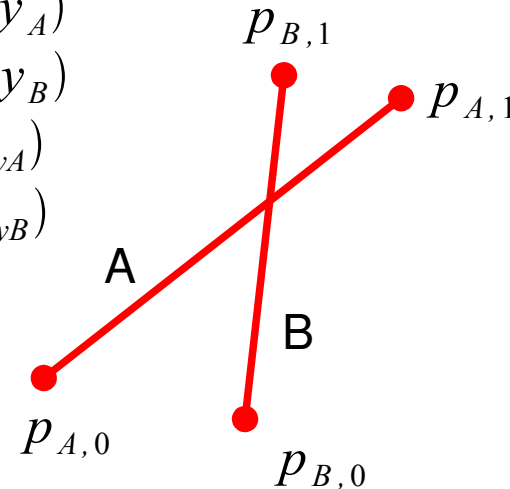
$$P_B(t) = t v_B + p_{B,0}, 0 \leq t \leq 1$$

$$p_{A,0} = (x_A, y_A)$$

$$p_{B,0} = (x_B, y_B)$$

$$v_A = (v_{xA}, v_{yA})$$

$$v_B = (v_{xB}, v_{yB})$$



We need to solve:

$$P_A(u) = P_B(t)$$

This gives us two equations and 2 unknowns:

$$u v_{xA} + x_A = t v_{xB} + x_B$$

$$u v_{yA} + y_A = t v_{yB} + y_B$$

The solutions are:

$$t = \frac{v_{xA}(y_B - y_A) - v_{yA}(x_B - x_A)}{v_{yA}v_{xB} - v_{xA}v_{yB}}$$

$$u = \frac{v_{xB}(y_A - y_B) - v_{yB}(x_A - x_B)}{v_{yB}v_{xA} - v_{xB}v_{yA}}$$

If $0 \leq t \leq 1 \wedge 0 \leq u \leq 1$ then the lines intersect!