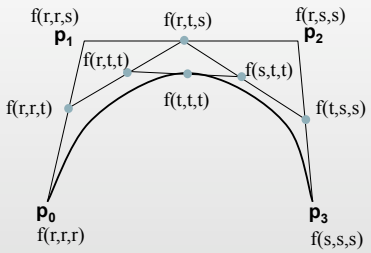
UCL

Drawing a Bézier curve

UCL

3rd degree Bézier curve

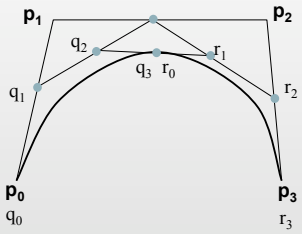
- p0, p1, p2, p3 = 4 original control points



UCL

Recursive interpolation

- From (p_0, p_1, p_2, p_3), we deduce 2 sets of control points: (q_0, q_1, q_2, q_3) and (r_0, r_1, r_2, r_3)



UCL

Algorithm for 3rd degree

```
void bezier(Point p[])
{
   Point q[ ], r[ ];
   if(colinear(p)) {
     draw_line(p[0],p[3])
   } else {
     /* split p into q and r */
     split(p,q,r);
     bezier(q);
     bezier(r);
   }
}
```

This is called DeCasteljau's Algorithm

UCL

Colinear

- Colinear checks if the 4 points p_0, p_1, p_2, p_3 are aligned
- Of course, you don't get the alignment exactly (numerical problems, degree of accuracy, etc…)
- We need to have an approximation
- Return true if the 4 points are approximately on a straight line

UCL

Colinear

- If (p_0,p_3) is given by $ax+by+c = 0$,
- Then compute the distance of p_1 and p_2 from this line
- The distance should be within a certain threshold
- If $D^2(x,y) = (ax+by+c)^2/(a^2+b^2)$, and $D^2(P_1)<\varepsilon$ and $D^2(p_2))<\varepsilon$ then the points are approximately colinear.

UCL

Alternative

- Check if
  - p[0], p[1], p[2], p[3] are very close together (2 pixels)
  - If yes, then draw line between points p[0] - p[3]
- Disadvantage
  - Need to apply recursion more often
- In any case colinear() needs to do this check as well

UCL

Split

- We need to compute R and Q from P, where
- q_0 = f(r,r,r), q_1=f(r,r,t), q_2=f(r,t,t), q_3=f(t,t,t)
- r_0 = f(t,t,t), r_1=f(t,t,s), r_2=f(t,s,s), r_3=f(s,s,s)
- You can split however you like, but it's convenient to split in 2 … using t=1/2

UCL

Drawing using Explicit Equation

- Remember explicit polynomial equation:
  - $f(t,t,t) = (1-t)^3 f(0,0,0) + 3t(1-t)^2 f(0,0,1) + 3(1-t)t^2 f(0,1,1) + t^3 f(1,1,1)$
    $= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3(1-t)t^2 P_2 + t^3 P_3$
- Why not just iterate over 't' and draw curve?
  - Increasing by t by Δt does not correspond to equidistant spacing of f(t) → f(t+Δt)
  - Unclear how to adjust step-size Δt appropriatelty

UCL

Conclusions

- Drawing is rather simple and efficient
- Boils down to splitting curve