Computer Science 112
The Art & Science of Computer Graphics
The College of Saint Rose
Spring 2013

# Topic Notes: Patch Meshes

When you can't get an interesting shape you want using simpler tools, you might need a *patch mesh*.

We construct it with a list of 16 points.

The patch is a quadrilateral in a very general sense – it is defined by 4 sides but the sides can be curved.

The points will define the surface we wish to model. The corners are nailed down and are the only ones we know for sure where they are.

We then specify a Bezier curve on each of the 4 sides of our quadrilateral.

These control points will get the edges to look the way we want.

We blend them into a surface with 4 interior control points that define 4 "internal" Bezier curves.

We can think of the points arranged as follows:

```
0--1--2--3
|  |  |  |
4--5--6--7
|  |  |  |
8--9--A--B
|  |  |  |
C--D--E--F
```

We specify them in this order in the Scheme code.

For example, we can define a square patch:

**Mead Example:** `PatchMesh`

We can adjust some of the control points to see what happens.

Note that we can keep the patches the same on corresponding edges aligned, allowing the possibility of "sewing together" a bunch of patches to get the very complex objects we want.

Note that a patch mesh is not a perfectly curved surface – it is approximated by a collection of rectangles. The number of rectangles is determined by the *refinement level* of the patch mesh.

- by default, the refinement level is 3

- you may wish to use lower refinement levels when rendering images as you develop your scene – it will look less smooth but will render much more quickly
- you may wish to use higher refinement levels to render final images – at the cost of a longer rendering time

To change the refinement level of a `PatchMesh`, send it the `refinement` message:

```
(object m PatchMesh
  (addPatch ... )
  (refinement 5)
)
```

**Mead Example:** `Waves`

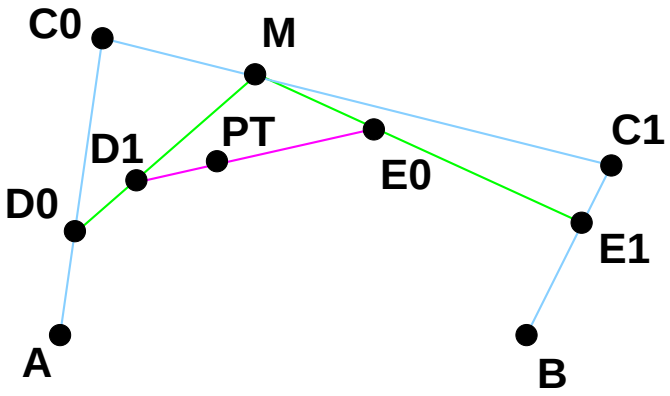
---

## Splitting Bezier Curves

We might also think about taking part of a patch and modeling it in more detail (like a quadtree type decomposition). Ideally, we can leave parts of it alone, but refine other parts.

We need to be able to break a Bezier curve into smaller Bezier curves. To achieve this, we go back to consider the Bezier curve "scaffolding" construction – the points we use to build that scaffolding will help us here.

So suppose we wish to split an existing cubic Bezier curve defined by 4 control points, `a`, `c0`, `c1`, and `b`, using the point `p` of the way through the curve as the split point.




Let's write a function that does just that. Our return should be a list of two sets of control points for Bezier curves that when put together form the original Bezier curve.

```
(define (splitBezier p a c0 c1 b)
  (let* ([d0 (blend p a c0)]  ; calculate p% from a to c0
         [m (blend p c0 c1)]  ; calculate p% from c0 to c1
```

```
          [e1 (blend p c1 b)]  ; calculate p% from c1 to b
          [d1 (blend p d0 m)]  ; calculate p% from d0 to m
          [e0 (blend p m e1)]
          [pt (blend p d1 e0)])
         (list (list a d0 d1 pt)
               (list pt e0 e1 b)
         )
  )
)
```

**Mead Example:** `SplitBezier`

We can use this to subdivide a patch and replace parts of it with 4 more detailed patches.

This is also useful for defining Bezier paths for other purposes, particularly when used to guide animations.