UCLA

CS174A : Introduction to Computer Graphics

Royce 190
TT 4-6pm

Scott Friedman, Ph.D
UCLA Institute for Digital Research and Education

Curves

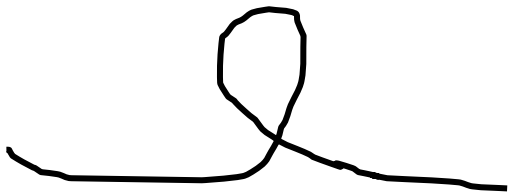
- So far we have only worked with flat surfaces
  - Even our sphere was approximated by repeatedly sub-dividing a flat sided object – a tetrahedron
  - This is convenient because OpenGL is designed to render flat objects (triangles) very efficiently.
- Here we introduce ways to model curves and surfaces directly
  - Their implementation will, ultimately, involve flat primitives (line, triangles) but their description will be as curves.

Curves

- Representations
  - **Explicit**
    - Familiar, but has problems for us
  - **Implicit**
    - Also familiar, but no analytic form we can use
  - **Parametric**
    - Less familiar, but most useful for our purposes

Curves

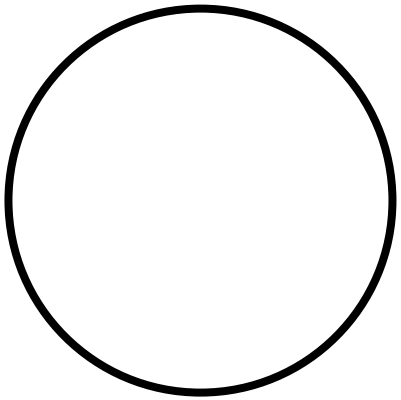
- Representations – **Explicit**
  - $y = f(x)$ form with one independent and one dependent variable.
  - No guarantee that this form exists for a given "curve"
  - For example, representation of a line $y=mx+b$
    - No way to represent a vertical line
  - Similarly, many curves have no explicit form
    - e.g.



Curves

- Representations – Explicit
  - A circle is even more obviously a problem
    - We can only represent *half* of the circle
    - Need to do define something like this

$$y = \sqrt{r^2 - x^2},\ y = -\sqrt{r^2 - x^2},\ \text{iff } 0 \le |x| \le r$$



Curves

- Representations – Explicit
  - The 3D case has similar problems as 2D
  - Curves require two equations (dependent variables)

$$y = f(x)$$
$$z = g(x)$$

  - Surfaces require two independent variables

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

  - Some Curves may not have a representation in this form either.

Curves

- Representations – Explicit
  - Again, some curves and surfaces may not have a representation in this form, same as the 2D case.
  - Lines cannot be described that exist on any plane of constant *x*, when defined in terms of *x*.
    - *Similar to vertical line problem*
  - Spheres cannot be described due to the same ambiguities as the 2D case without constraints.
    - i.e. $z=f(x,y)$ can generate 0, 1 or 2 points on the sphere.
  - Too many corner cases to be useful for more complex curves and surfaces (patches)

Curves

- Representations – **Implicit**
  - Curves are represented using the form
    - $f(x,y) = 0$
  - A line has the usual form we have seen
    - $ax + by + c = 0$
  - Circles look like this
    - $x^2 + y^2 - r^2 = 0$
  - No real representational limitations here - yay!

Curves

- Representations – Implicit
  - Except that the implicit form is more useful for testing membership.
    - Is the point on the line, curve, surface or not?
    - Recall collision detection discussion.
  - There is, generally, no convenient way to *analytically* determine x given a y value.
  - This difficulty limits its usefulness for rendering.

Curves

- Representations – Implicit
  - In 3D the same hold true
    - We can represent lines, curves and surfaces
  - Surfaces
    - $f(x,y,z) = 0$
  - Lines
    - $ax + by + cz + d = 0$
  - Curves are trickier
    - Intersection of two 3D surfaces, but the same in principle.
    - $f(x,y,z) = 0$
    - $g(x,y,z) = 0$

Curves

- Representations – Implicit
  - While just about any curve or surface for which we have a use has an implicit representation
    - Extracting points along/on them is very difficult to do.
    - We need those points in order to *draw* the curve!
  - What to do?

Curves

- Representations – **Parametric**
  - Represents each dimensional point on a curve with respect to a single *independent* variable, u.

$$x = x(u),$$
$$y = y(u),$$
$$z = z(u).$$

    - The 2D representation just drops the $z()$
  - By varying u we can generate the points that sweep out the curve.
  - Convenient for rendering
    - Not necessarily fast, however.

Curves

- Representations – Parametric
  - Surfaces are similarly represented

$$x = x(u,v), \quad y = y(u,v), \quad z = z(u,v). \qquad \text{or} \quad \mathbf{p}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$

  - By varying u and v we can sweep out a surface.
  - Recall that by taking the cross product of the tangent vectors at a point we get the normal of the surface.

$$\mathbf{n} = \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}$$

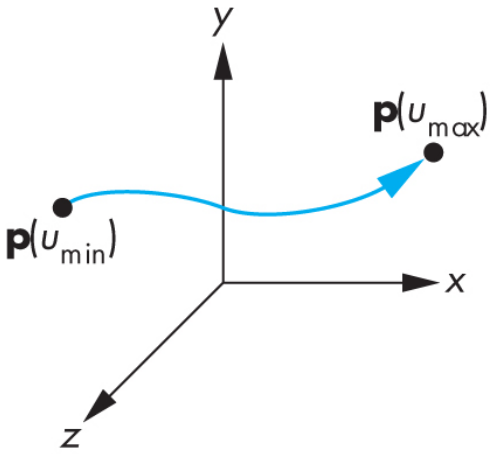
Curves

- Representations – Parametric
  - Curves represented as polynomials in terms of u and surfaces in terms of u and v are very convenient to use because they enable
    - Local control of their shape
    - Smoothness and continuity control – between curves/surfaces
    - Ability to evaluate derivatives
    - Behavioral stability
    - Ease of rendering

Curves

- Representations – Parametric
  - Consider a curve

$$\mathbf{p}(u)=\begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix}$$



  - The polynomial representation of degree n is (not dimension)

$$\mathbf{p}(u)=\sum_{k=0}^{n} u^k c_k$$

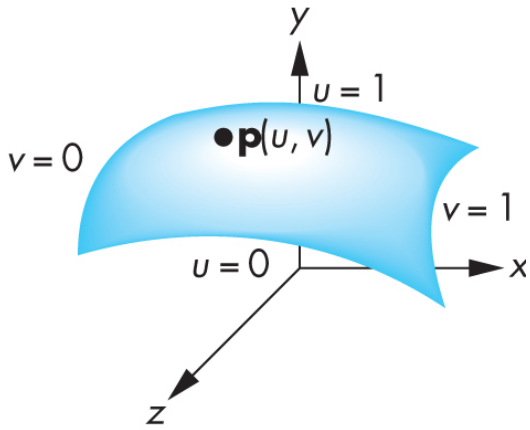
  - Each of the $n+1$ c_k's has independent coefficients

$$c_k=\begin{bmatrix} c_{xk} \\ c_{yk} \\ c_{zk} \end{bmatrix}$$

Curves

- Representations – Parametric
  - Surfaces are similar

$$\mathbf{p}(u,v)=\begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$



  - The polynomial representation of degree n is then

$$\mathbf{p}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} c_{ij}u^i v^j$$

  - Each of the $n+1$ and $m+1$ c_k's, again, has independent coefficients
  - Generally, $n=m$ generating a square shaped surface patch.

Curves

- Parametric – choosing the degree
  - As you can see, we can use whatever degree we like.
  - However a high degree can result in a complex curve that is hard to control.
  - A low degree can be too simple to represent what we need.
  - Solution, use a collection of simpler curves to represent more complex shapes.
  - In computer graphics the most common degree is 3 resulting in cubic polynomials.
    - Which is what you will almost always use.

Curves

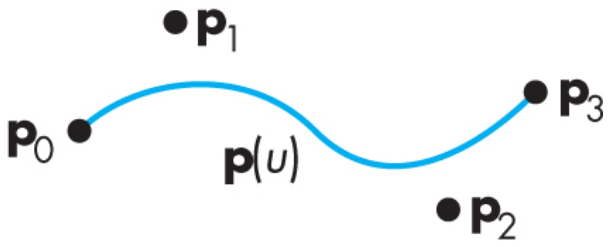
- Parametric – choosing the degree
  - A degree of 3 also allows us to keep shape control local.
  - It also allows us to manage the joints between curve segments more easily.
    - For a degree of 3, $\mathbf{p}(u)=c_0+c_1u+c_2u^2+c_3u^3=\sum_{k=0}^{3}c_ku^k=\mathbf{u}^T\mathbf{c}$,

where

$$\mathbf{c}=\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}, \quad \mathbf{u}=\begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$$

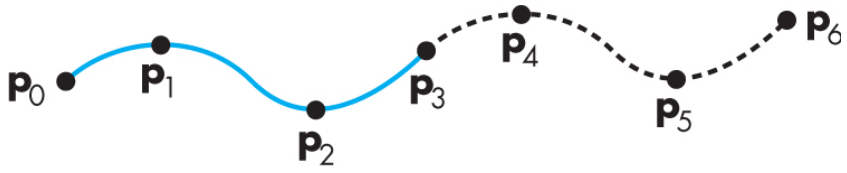
Curves

- Parametric – choosing the degree
  - The values for **c** are what we have to solve for
    - There are 12 equations in 12 unknowns (when degree is 3)
    - Because *x*, *y* and *z* are independent we can group the problem into three sets of 4 equations with 4 unknowns.
    - We use a set of control point data to help solve for the unknowns which will then allow us to generate the points along the curve



Curves

- Parametric – cubic interpolating polynomial
  - Not often used in CG but motivates our discussion
  - There are other curve types with beneficial properties for control and rendering.
    - As we will see
  - For now, we have four control points in 3D
  - We seek the coefficients **c** such that the polynomial $\mathbf{p}(u)=\mathbf{u}^T\mathbf{c}$ passes through the control points.



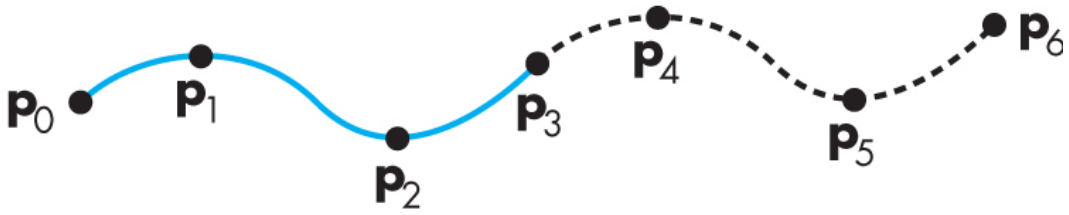
Curves

- Parametric – cubic interpolating polynomial
  - Our control points are defined by

$$\mathbf{p}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}$$

  - We also choose the values of u at each $\mathbf{p}$ to perform the interpolation, the following are convenient.

$$u = 0, \frac{1}{3}, \frac{2}{3}, 1$$



Curves

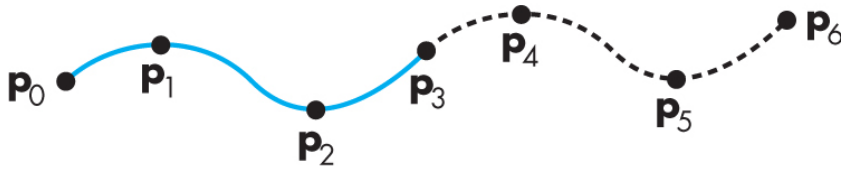
- Parametric – cubic interpolating polynomial
  - This gives the four conditions

$$\mathbf{p}_0 = \mathbf{p}(0) = \mathbf{c}_0,$$

$$\mathbf{p}_1 = \mathbf{p}\left(\frac{1}{3}\right) = \mathbf{c}_0 + \frac{1}{3}\mathbf{c}_1 + \left(\frac{1}{3}\right)^2 \mathbf{c}_2 + \left(\frac{1}{3}\right)^3 \mathbf{c}_3,$$

$$\mathbf{p}_2 = \mathbf{p}\left(\frac{2}{3}\right) = \mathbf{c}_0 + \frac{2}{3}\mathbf{c}_1 + \left(\frac{2}{3}\right)^2 \mathbf{c}_2 + \left(\frac{2}{3}\right)^3 \mathbf{c}_3,$$

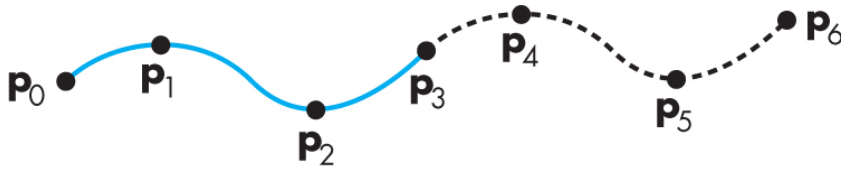
$$\mathbf{p}_1 = \mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3.$$



Curves

- Parametric – cubic interpolating polynomial
  - In matrix form

$$\mathbf{p} = \mathbf{Ac}, \quad \text{where} \quad \mathbf{p} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \text{and} \quad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{3} & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \frac{2}{3} & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$



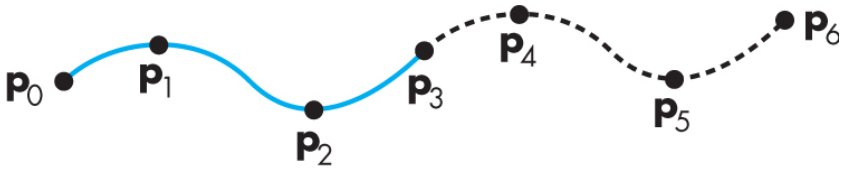
Curves

- Parametric – cubic interpolating polynomial
  - Inverting A gives us the *interpolating geometry matrix*

$$\mathbf{M}_I = \mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix}$$

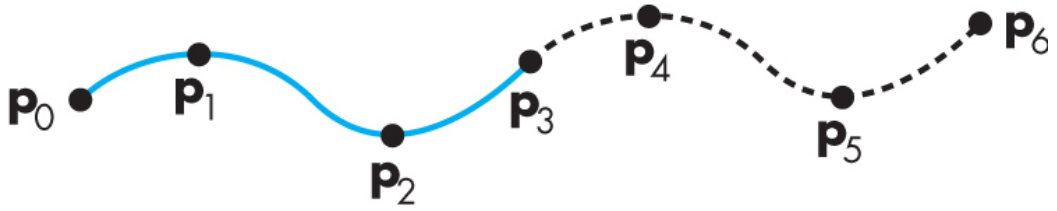
and our coefficients

$\mathbf{c} = \mathbf{M}_I \mathbf{p}$.



Curves

- Parametric – cubic interpolating polynomial
  - We can now evaluate the points along the curve.
  - If we wish to continue our curve we can define the next segment using $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$, and $\mathbf{p}_6$
  - this will ensure that the curve will achieve *continuity*.
  - However it does not ensure the *derivative* at the point, $\mathbf{p}_3$, where the curves *join* will be the same!



Curves

- Parametric – cubic interpolating polynomial
  - Is that a problem?
    - Maybe, maybe not – it depends on what you are trying to achieve.
    - Recall that earlier we said we desire the ability to be able to build up complex curves using simpler ones
    - The behavior of how the curves *join* says a lot about how we can achieve this desire.