Curve and Surface Basics

- Implicit and parametric forms
- Power basis form
- Bezier curves
- Rational Bezier Curves
- Tensor Product Surfaces

Implicit and Parametric Forms

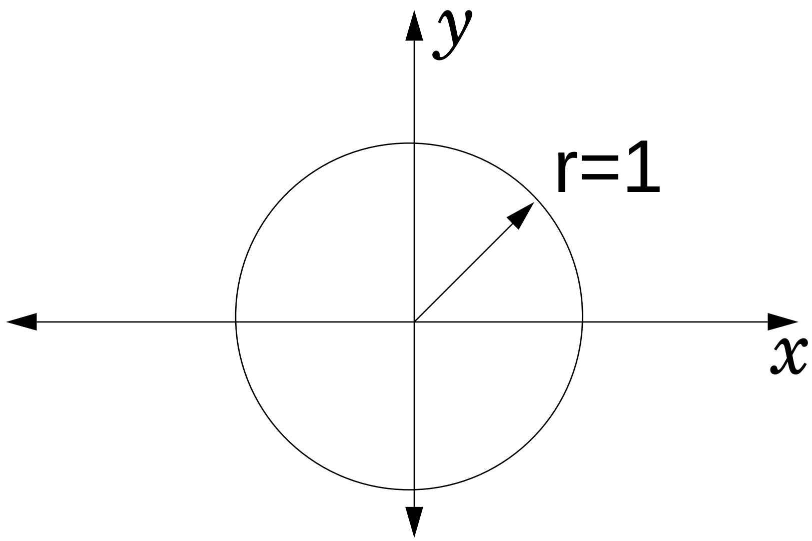
Implicit Form:

- implicit functional relationship between coordinates of points lying on a curve.

Implicit Form:

- example, circle of radius 1:

$$f(x, y) = x^2 + y^2 - 1 = 0$$



Parametric Form:

- each coordinate of points on the curve is represented separately as an explicit function of an independent variable, *u*:

$$\boldsymbol{C}(u) = (x(u), y(u)), \quad a \le u \le b$$

- the interval [*a, b*] is arbitrary. It is usually normalized to [0,1]

The first quadrant of the unit circle can be defined by the parametric functions:

$$\begin{aligned} x(u) &= \cos(u) \\ y(u) &= \sin(u) \end{aligned} \quad , 0 \le u \le \pi/2$$

- Note, setting $t = \tan(u/2)$, gives an alternative parametric form:

$$\begin{aligned} x(t) &= \frac{1 - t^2}{1 + t^2} \\ y(t) &= \frac{2t}{1 + t^2} \end{aligned} \quad , 0 \le t \le 1$$

- Thus the parametric representation of a curve is ***NOT*** unique.

We can think of a parametric curve ***C****(u)* as the path traced out by a particle as a function of time:

- *u* is the time variable,
- [*a, b*] is the time interval.

Then the first and second derivatives of ***C**(u)* are the velocity and acceleration of the particle, respectively, thus:

$$\boldsymbol{C}'(u) = (x'(u), y'(u)) = (-\sin(u), \cos(u))$$

$$\boldsymbol{C}'(t) = (x'(t), y'(t)) = \left(\frac{-4t}{(1+t^2)^2}, \frac{2(1-t^2)}{(1+t^2)^2}\right)$$

Note that the magnitude of the velocity vector ***C'***(u) is a constant:

$$\left|\boldsymbol{C}'(u)\right| = \sqrt{\sin^2(u) + \cos^2(u)}$$

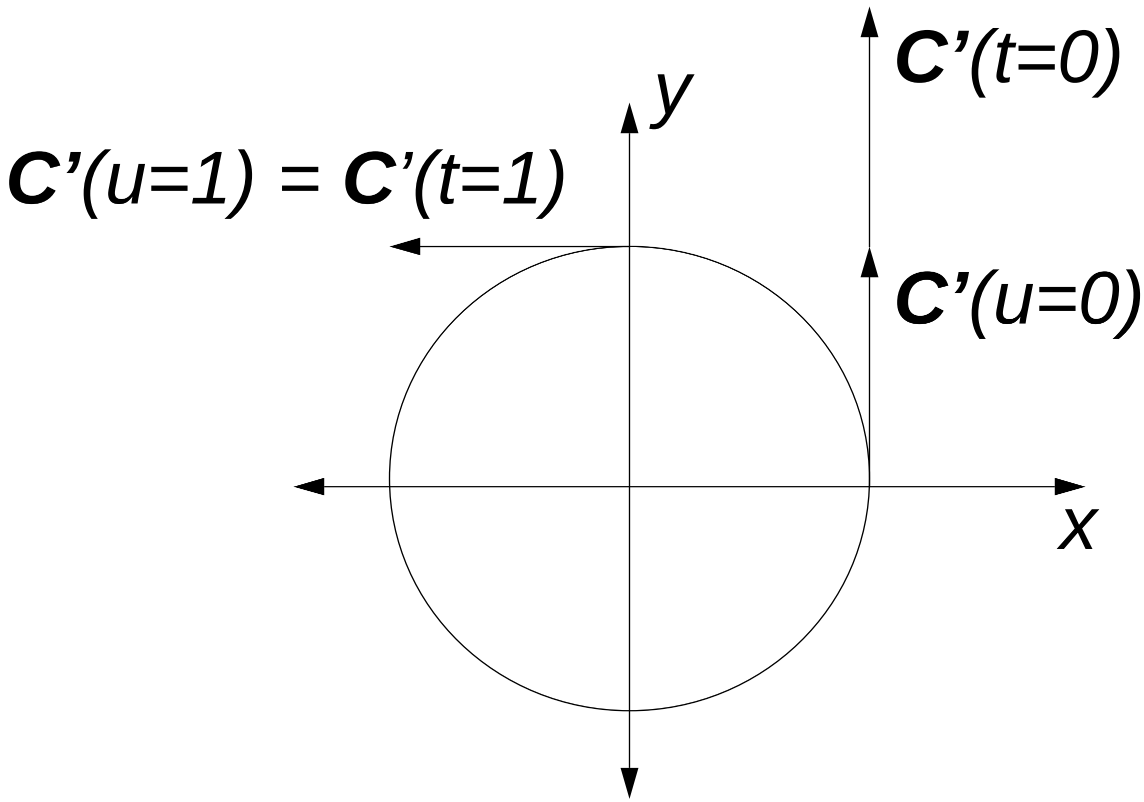
This is referred to as *uniform parametrization*.

Substituting $t = 0$ and $t = 1$ into $\mathbf{C}'(t)$ yields:

- $\mathbf{C}'(0) = (0,2)$ and
- $\mathbf{C}'(1) = (-1,0)$,

i.e., the particle's start speed is twice its end speed.

---> *non-uniform parameterization*


C'(t=0)
y
C'(u=1) = C'(t=1)
C'(u=0)
x

An implicit *surface* is defined by an equation of the form *f(x,y,z)* = 0.

For example, a sphere of radius 1, centered at the origin:

$$f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$$

A parametric representation (not unique) of the same sphere is given by

S(u,v) = $(x(u,v), y(u,v), z(u,v))$, where:

$$\begin{aligned} x(u,v) &= \sin(u)\cos(v) \\ y(u,v) &= \sin(u)\sin(v) \\ z(u,v) &= \cos(u) \end{aligned}$$

$$0 \le u \le \pi; \quad 0 \le v \le 2\pi$$

Note that two parameters are required to define a surface.

Denote the partial derivatives of $\boldsymbol{S}(u,v)$ by:

$$\boldsymbol{S}_u(u,v) = (x_u(u,v), y_u(u,v), z_u(u,v))$$

$$\boldsymbol{S}_v(u,v) = (x_v(u,v), y_v(u,v), z_v(u,v))$$

At any surface point at which the cross product $\boldsymbol{S}_u \times \boldsymbol{S}_v$ does not vanish, the unit normal vector is given by:

$$\boldsymbol{N} = \frac{\boldsymbol{S}_u \times \boldsymbol{S}_v}{\left|\boldsymbol{S}_u \times \boldsymbol{S}_v\right|}$$

- The existence of the normal, and the corresponding tangent plane, is a geometric property of the surface, independent of parameterization.
- Different parameterizations will give different partial derivatives, but the above equation will always yield ***N***.

Implicit and parametric forms have their advantages and disadvantages; both have been applied to useful modeling methods.

Summary

- By adding a z-coordinate, the parametric method is easily extended to represent arbitrary curves in 3D space. The implicit form can only represent curves in the coordinate planes (i.e., the xy-, xz-, or yz-planes).

- Boundedness is built into the parametric form through the bounds on the parametric interval. It is cumbersome to represent bounded curve segments (or surface patches) with the implicit form.

- Parametric curves possess a natural direction of traversal, implicit curves do not.

- Parametric forms can provide a natural method for designing and representing shape in a computer. Techniques exist which relate the coefficients of parametric functions to geometrically intuitive control “handles”

- Parametric form sometimes produces anomalies which are unrelated to the true geometry (e.g., vanishing normals)

- The complexity of many geometric operations depends greatly on the method of representation. Two classic examples are:

  1) Compute a point on a curve or surface (difficult in implicit form)

  2) Given a point, determine if it is on the curve or surface (difficult in the parametric form)

Power Basis Form of a Curve

The choice of basis for a parametric curve is arbitrary, but the choice involves trade-offs. Ideally, we want a class of functions which:

- are capable of representing precisely all curves needed
- are easily, efficiently and accurately processed in a computer

- numerical processing is insensitive to floating point operations
- functions should require little memory for storage

Polynomials satisfy the last two criteria - there are curve and surface types which cannot be represented using polynomials.

Two common methods of expressing polynomial functions are:

- Power basis (algebraic form)
- Bezier (a “geometric” form)

Although they are mathematically equivalent, the Bezier is better suited to representing and manipulating shape on a computer.

An *n*-th degree power basis curve is given by:

$$\boldsymbol{C}(u) = (x(u), y(u), z(u))$$
$$= \sum_{i=0}^{n} \boldsymbol{a}_i u^i, \quad 0 \le u \le 1$$

The $\boldsymbol{a}_i = (x_i, y_i, z_i)$ are vectors, so:

$$x(u) = \sum_{i=0}^{n} x_i u^i$$

$$y(u) = \sum_{i=0}^{n} y_i u^i$$

$$z(u) = \sum_{i=0}^{n} z_i u^i$$

Alternatively, the polynomial can be written in matrix form:

$$\boldsymbol{C}(u) = \begin{bmatrix}\boldsymbol{a}_0\ \boldsymbol{a}_1\ \dots\ \boldsymbol{a}_n\end{bmatrix}\begin{bmatrix}1\\ u\\ \dots\\ u^n\end{bmatrix} = (\boldsymbol{a}_i)^T(u_i)$$

Differentiation yields:

$$\boldsymbol{a}_i = \frac{\boldsymbol{C}^{(i)}(u)\big|_{u=0}}{i!}$$

where, $\boldsymbol{C}^{(i)}(u)|_{u=0}$ is the i-th derivative of $\boldsymbol{C}(u)$ at u=0.

- The n + 1 functions, $\{u^i\}$, are called the basis (or blending) functions, and the $\{a_i\}$, the coefficients of the power basis representation.

Given a parameter value u_i, the point $\boldsymbol{C}(u_i)$ on a power basis curve is most efficiently computed using *Horner's* method:

for,
- degree = 1: $C(u_i) = a_1 u_i + a_0$
- degree = 2: $C(u_i) = (a_2 u_i + a_1) u_i + a_0$
- :
- degree = n: $C(u_i) = ((\ldots (a_n u_i + a_{n-1}) u_i + a_{n-2}) u_i + \ldots + a_0$

The general algorithm is:

```
Algorithm A1.1
  Horner1(a, n, u, C)
  { /* Compute point on power
         basis curve */
    /* Input: a, n, u */
    /* Output: C */
  C = a[n];
  for (i=n-1; i>=0; i--)
    C = C*u + a[i];
  }
```

Bezier Curves

The power basis form has the following disadvantaged with respect to interactive geometric design:

- Coefficients $\{a_i\}$ convey little geometric insight. In addition, a designer typically wants to control both ends of the curve, not just the start point

- Numerically, it is a rather poor form; e.g., Horner’s method is prone to round-off error if the coefficients vary in magnitude

The *Bezier* form overcomes these deficiencies

An n-th degree Bezier curve is defined by:

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\boldsymbol{P}_i \ , \quad 0 \le u \le 1$$

The basis (blending) functions, $\{B_{i,n}(u)\}$ are the n-th degree Bernstein polynomials, given by:

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!}u^i(1-u)^{n-i}$$

The coefficients of this geometric form $\{\boldsymbol{P}_i\}$ are called control points.

Examples:

$n = 1$:

From the definition,

$$B_{0,1}(u) = 1 - u, \text{ and } B_{1,1}(u) = u$$

The Bezier curve takes the form:

$$\mathbf{C}(u) = (1 - u)\mathbf{P}_0 + u\mathbf{P}_1$$

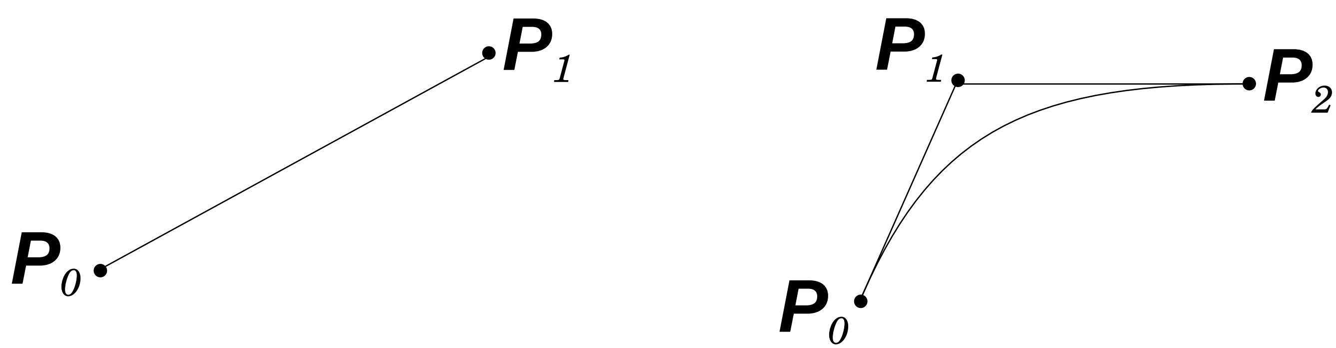
i.e., a parametric line segment from $\mathbf{P}_0$ to $\mathbf{P}_1$

$n = 2$:

From the definition,

$$\boldsymbol{C}(u) = (1 - u)^2\boldsymbol{P}_0 + 2u(1 - u)\boldsymbol{P}_1 + u^2\boldsymbol{P}_2 ,$$

which is a parabolic arc from $\boldsymbol{P}_0$ to $\boldsymbol{P}_2$.



Note that:

- the polygon formed by $\{\boldsymbol{P}_0, \boldsymbol{P}_1, \boldsymbol{P}_2\}$ approximates the curve. This polygon is called the *control polygon*.

- $\boldsymbol{P}_0 = \boldsymbol{C}(0)$ and $\boldsymbol{P}_2 = \boldsymbol{C}(1)$

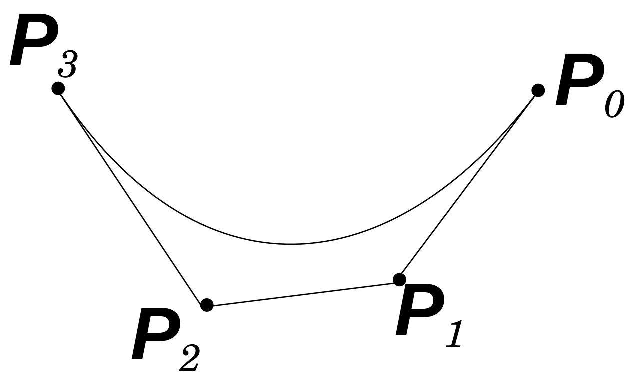
- The tangent directions to the curve at its endpoints are parallel to $\boldsymbol{P}_1 - \boldsymbol{P}_0$ and $\boldsymbol{P}_2 - \boldsymbol{P}_1$.

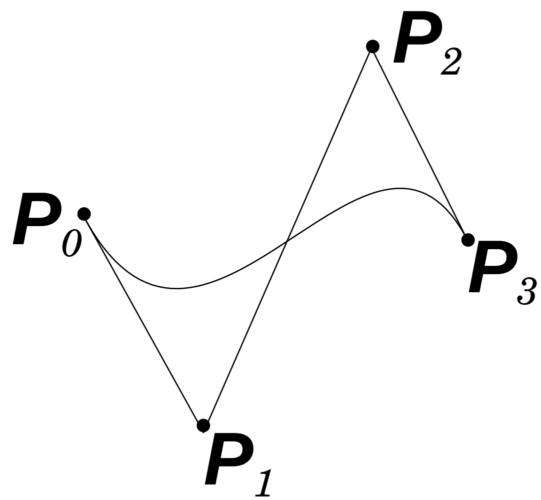
- the curve is contained in the triangle $\boldsymbol{P}_0\boldsymbol{P}_1\boldsymbol{P}_2$

$n = 3$:

$\mathbf{C}(u) =$

$$(1 - u)^3\mathbf{P}_0 + 3u(1 - u)^2\mathbf{P}_1 + 3u^2(1 - u)\mathbf{P}_2 + u^3\mathbf{P}_3$$





Note that:

- the control polygons approximate the shapes of the curves
- $\boldsymbol{P}_0 = \boldsymbol{C}(0)$ and $\boldsymbol{P}_3 = \boldsymbol{C}(1)$
- endpoint tangent directions are parallel to $\boldsymbol{P}_1 - \boldsymbol{P}_0$ and $\boldsymbol{P}_3 - \boldsymbol{P}_2$

- *Convex Hull Property*: the curves are contained in the convex hulls of their defining control polygons

- *Variation Diminishing Property*: no straight line intersects a curve more times than it intersects the curve's control polygon (for 3D replace the words straight line with plane)

- Initially (at u=0) the curve is turning in the same direction as $\boldsymbol{P_0P_1P_2}$. At u=1 it is turning in the direction of $\boldsymbol{P_1P_2P_3}$
- A loop in the control point polygon may or may not imply a loop in the curve. The transition is a curve with a cusp.
- Bezier curves are invariant under affine transformations (i.e., rotations, translations, scale). Simply transform the control points

Bezier Curves

Properties of Bezier basis functions, $\{B_{i,n}(u)\}$:

P1.1: Non-negativity: $B_{i,n}(u) \geq 0$ for all i, n and $0 \leq u \leq 1$

P1.2: Partition of unity: $\sum B_{i,n}(u) = 1$ for all $0 \leq u \leq 1$

P1.3: $B_{0,n}(0) = B_{n,n}(1) = 1$

P1.4: $B_{i,n}(u)$ attains exactly one maximum on the interval [0,1], i.e., at $u = i/n$

P1.5: Symmetry: For any n, the set polynomials, $B_{i,n}(u)$ is symmetric with respect to $u = 1/2$

P1.6: Recursive definition:

$B_{i,n}(u) = (1 - u)B_{i,n-1}(u) + uB_{i-1,n-1}(u)$.

We define $B_{i,n}(u) \equiv 0$ if $i < 0$ or $i > n$.

P1.7: Derivatives:

$$B'_{i,n}(u) = \frac{d}{du}B_{i,n}(u)$$
$$= n\left(B_{i-1,n-1}(u) - B_{i,n-1}(u)\right)$$

where,

$$B_{-1,n-1}(u) \equiv B_{n,n-1}(u) \equiv 0$$

Properties 1.6 and 1.7 provide the basis for recursive algorithms to compute Bezier blending functions and their derivatives.