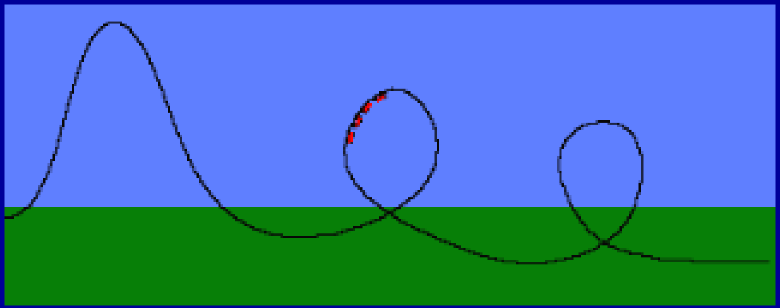
Parametric Curves

Modeling:
- **parametric curves (Splines)**
- **polygonal meshes**

Roller coaster

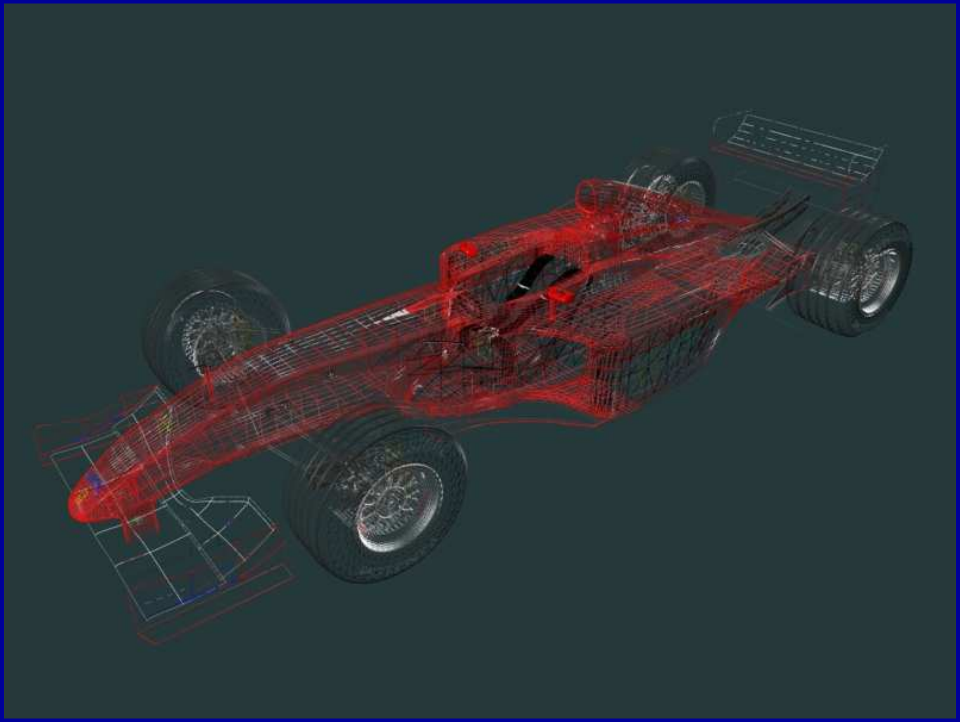
- Next programming assignment involves creating a 3D roller coaster animation
- We must model the 3D curve describing the roller coaster, but how?
- How to make the simulation obey the laws of gravity?



Movie 1:24

Modeling Complex Shapes

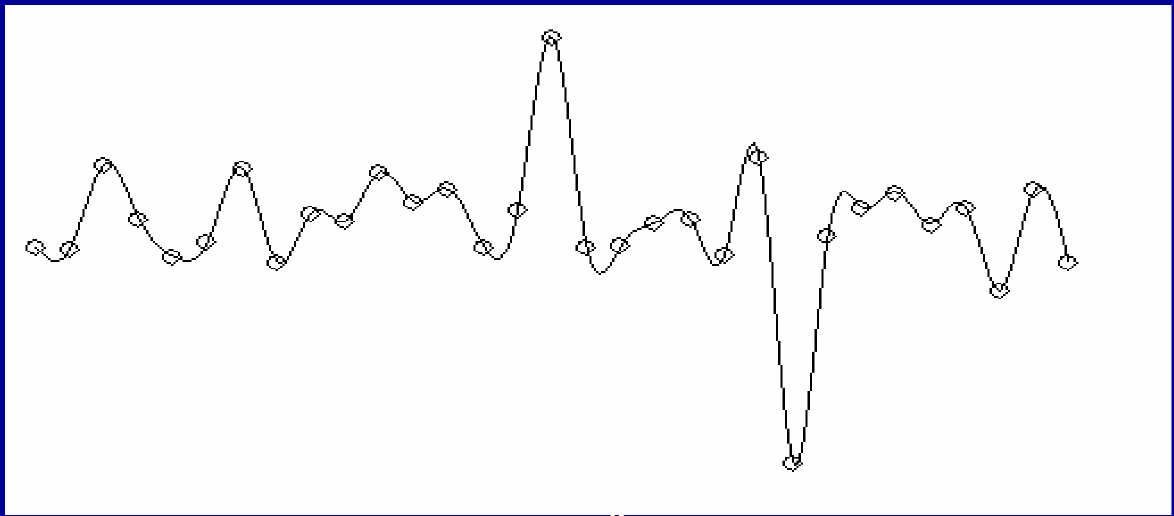
- We want to build models of very complicated objects
- An equation for a sphere is possible, but how about an equation for a telephone, or a face?
- Complexity is achieved using simple pieces
  - polygons, parametric curves and surfaces, or implicit curves and surfaces
  - This lecture: parametric curves



What Do We Need From Curves in Computer Graphics?

- Local control of shape (so that easy to build and modify)
- Stability
- Smoothness and continuity
- Ability to evaluate derivatives
- Ease of rendering

Demo

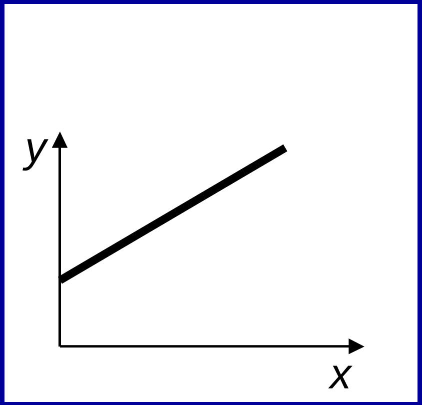


Curve Representations

- **Explicit: y = f(x)**

$$y = mx + b$$

  - **Easy to generate points**
  - **Must be a function: big limitation—vertical lines?**

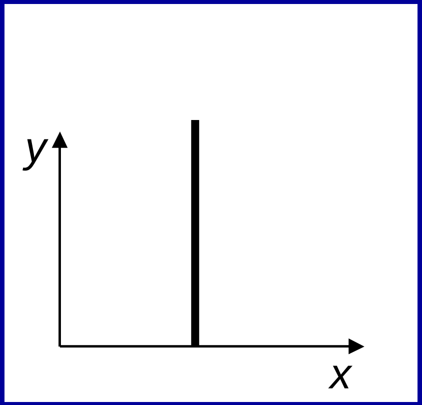


Curve Representations

- **Explicit: y = f(x)**

$$y = mx + b$$

  - **Easy to generate points**
  - **Must be a function: big limitation—vertical lines?**

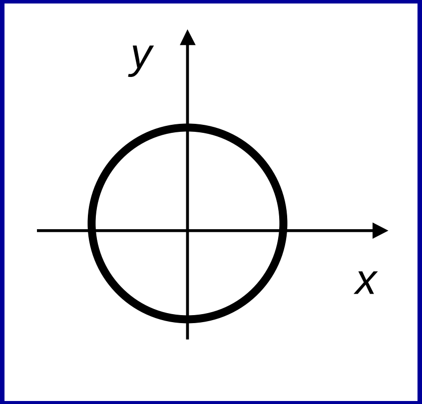


•Implicit: f(x,y) = 0

$$x^2 + y^2 - r^2 = 0$$

+Easy to test if on the curve

–Hard to generate points



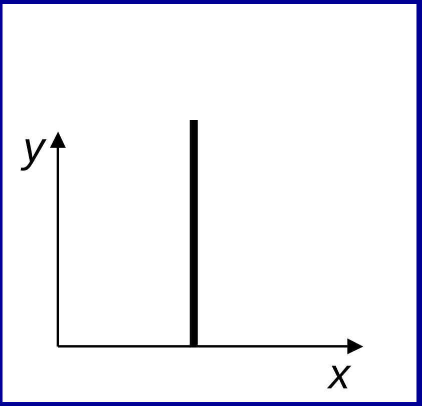
Curve Representations

- **Explicit: y = f(x)**

$$y = mx + b$$

+ Easy to generate points

– Must be a function: big limitation—vertical lines?

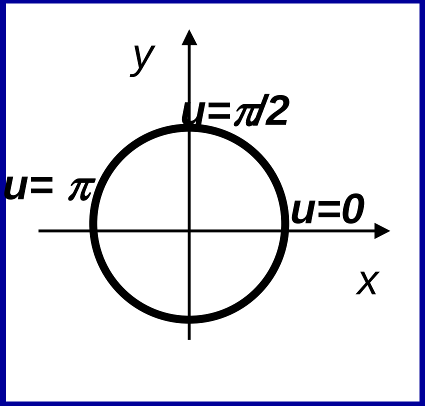


•Implicit: f(x,y) = 0

$$x^2 + y^2 - r^2 = 0$$

+Easy to test if on the curve

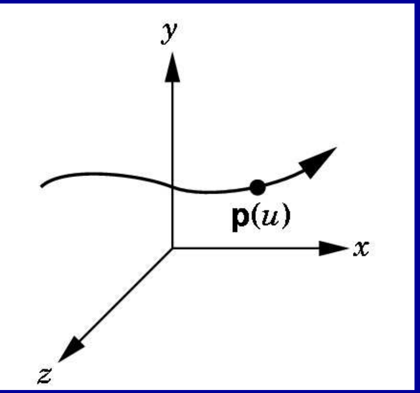
–Hard to generate points



•Parametric: (x,y) = (f(u), g(u))

$$(x, y) = (\cos u, \sin u)$$

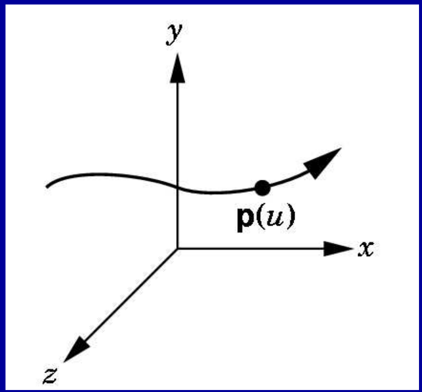
+Easy to generate points



Parameterization of a Curve

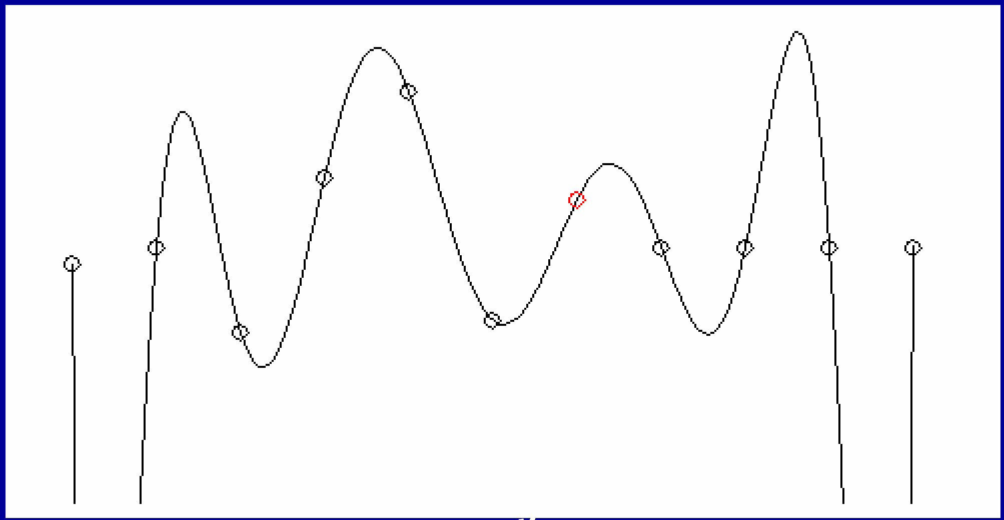
- ***Parameterization* of a curve: how a change in u moves you along a given curve in xyz space.**

- **There are an infinite number of parameterizations of a given curve. Slow, fast, speed continuous or discontinuous, clockwise (CW) or CCW…**



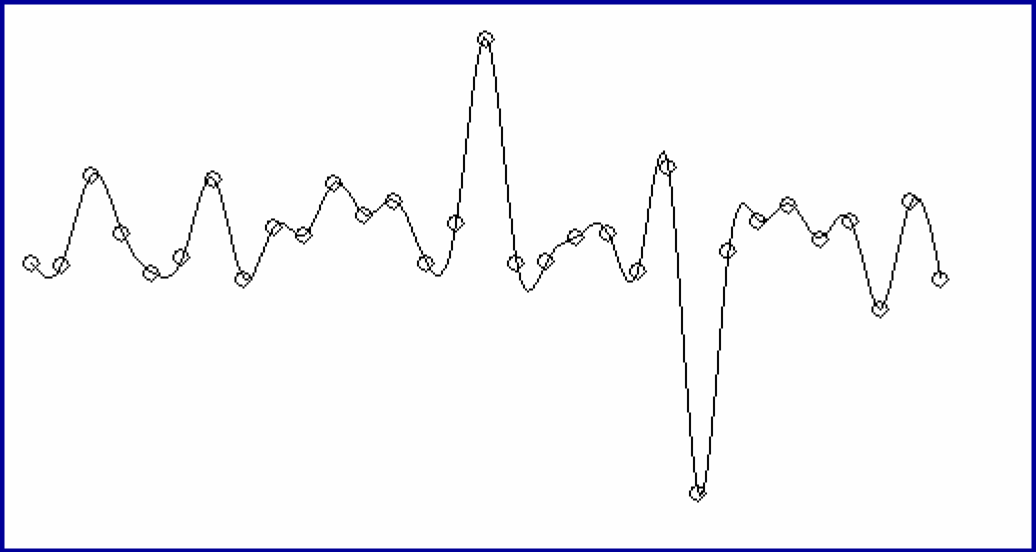
Polynomial Interpolation

- **An *n*-th degree polynomial fits a curve to n+1 points**
  - **called Lagrange Interpolation**
  - **result is a curve that is too wiggly, change to any control point affects entire curve (nonlocal) – *this method is poor***
- **We usually want the curve to be as smooth as possible**
  - **minimize the wiggles**
  - **high-degree polynomials are bad**



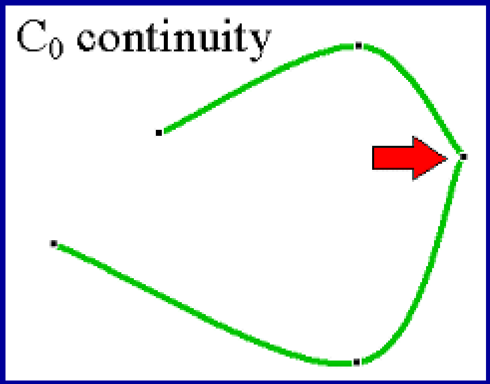
Splines: Piecewise Polynomials

- **A spline is a *piecewise polynomial* - many low degree polynomials are used to interpolate (pass through) the control points**
- ***Cubic piecewise* polynomials are the most common:**
  - **piecewise definition gives local control**

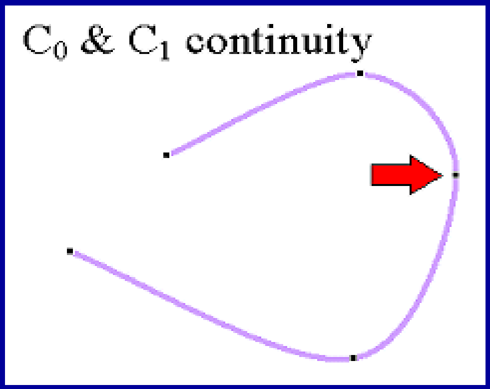


Piecewise Polynomials

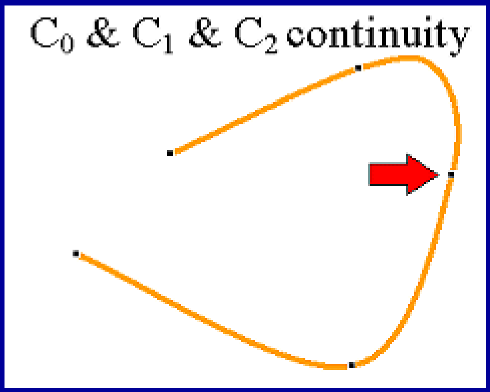
- Spline: lots of little polynomials pieced together
- Want to make sure they fit together nicely



Continuous in position



Continuous in position and tangent vector



Continuous in position, tangent, and curvature

Splines

- **Types of splines:**
  - **Hermite Splines**
  - **Catmull-Rom Splines**
  - **Bezier Splines**
  - **Natural Cubic Splines**
  - **B-Splines**
  - **NURBS**

Splines

chalkboard

The Cubic Hermite Spline Equation

- Using some algebra, we obtain:

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

point that gets drawn

basis

control matrix (what the user gets to pick)

- This form typical for splines
  - basis matrix and meaning of control matrix change with the spline type

The Cubic Hermite Spline Equation

- Using some algebra, we obtain:

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

point that gets drawn

basis

control matrix (what the user gets to pick)

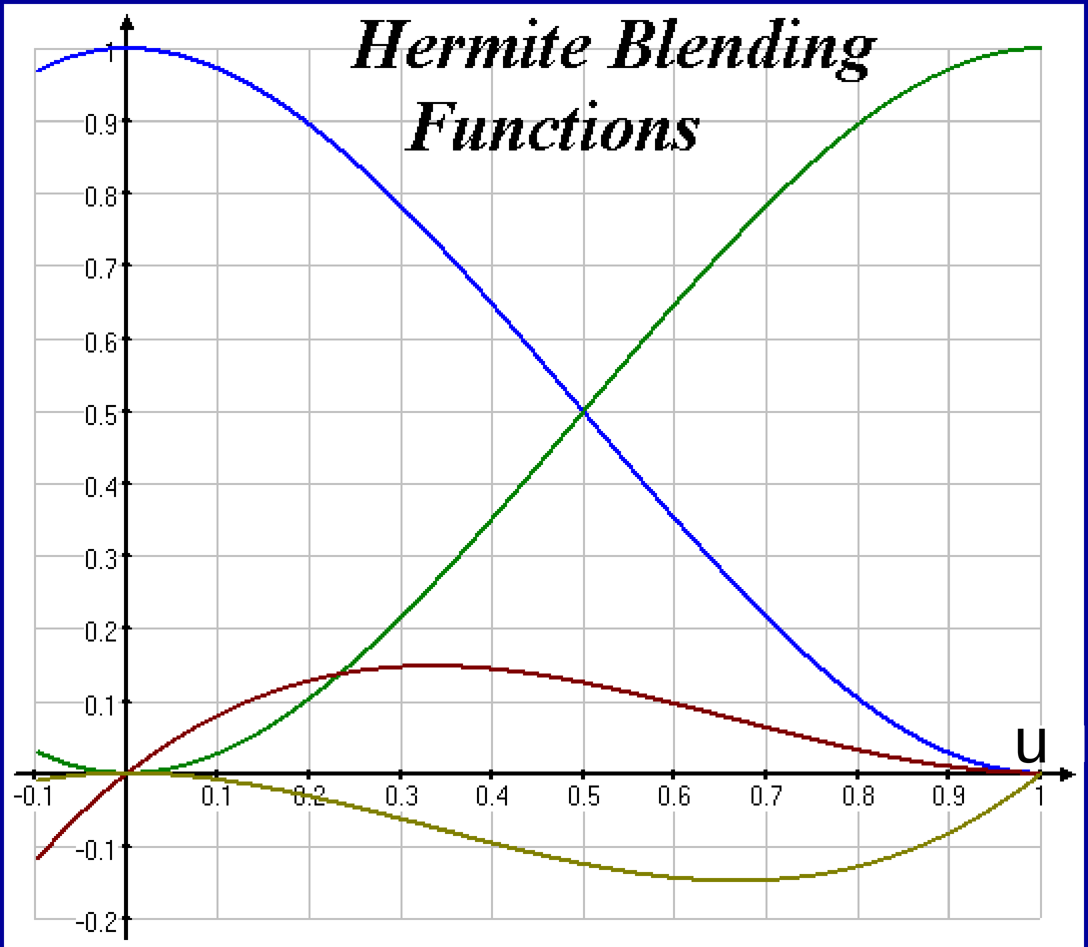
$$p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

4 Basis Functions

Four Basis Functions for Hermite splines

$$p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

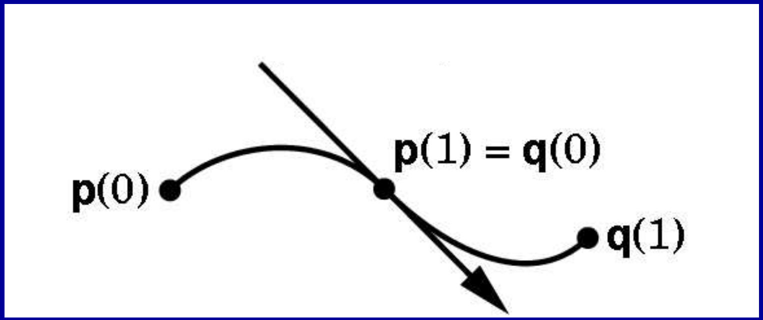
4 Basis Functions



Every cubic Hermite spline is a linear combination (blend) of these 4 functions

Piecing together Hermite Curves

- It's easy to make a multi-segment Hermite spline
  - each piece is specified by a cubic Hermite curve
  - just specify the position and tangent at each “joint”
  - the pieces fit together with matched positions and first derivatives
  - gives C1 continuity
- The points that the curve has to pass through are called *knots* or *knot points*



Catmull-Rom Splines

- **Use for the roller-coaster (next programming assignment)**
- **With Hermite splines, the designer must arrange for consecutive tangents to be collinear, to get C^1 continuity. This gets tedious.**
- **Catmull-Rom: an interpolating cubic spline with *built-in C^1 continuity*.**

chalkboard

Catmull-Rom Spline Matrix

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

spline coefficients **CR basis** **control vector**

- **Derived similarly to Hermite**
- **Parameter s is typically set to s=1/2.**

Catmull-Rom Spline Matrix

$$[x \;\; y \;\; z] = [u^3 \;\; u^2 \;\; u \;\; 1]\begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}$$

spline coefficients **CR basis** **control vector**