Curves and Surfaces

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd

Todays Topics

- Final projects
- Surface representations
- Smooth curves
- Subdivision

Final Project Requirements

- A Project Proposal (DUE 4/16)
  - No extensions on the due date (there is not enough room in the schedule to slip)
  - Requirements
    - A written summary of your proposed final project
      - Equivalent of an abstract
      - Not to exceed two pages in length
      - Provide a description of the system
      - Discussion of the techniques employed, specifically those learned in COMP236
    - An outline of the minimal functionality that you expect
      - In the style of our previous programming projects this will define the 80% mark for your project. You should think carefully about what you put here, because you want be able to do any 11th hour juggling between required and optional items.
      - Be conservative here! (But, if you low-ball it I might request a few changes)
    - A list of optional features that you would like to provide in your project if time permits
      - Each optional feature should be somewhat "self-contained"
      - The system should function and be useful without any of these items.

Final Project Requirements (cont)

- A project Web site (DUE before midnight 5/3)
  - Include a short write-up
    - Slightly more than the abstract used in the proposal
      - Motivate the problem solved by your project
      - Discuss your approach and its limitations
  - Illustrative figures
    - Examples from your system (preferred)
    - Motivating examples from other people's work (include links if possible)
  - A simple user's guide
    - Explain all of the command-line arguments, menu-options, key strokes, and mouse modes and usage of your system.
    - Discuss the formats of all input files
  - Links to your code
    - Zipped files of any source code you developed and any libraries that you used, you should include everything needed to build your system, other than OpenGL and GLUT.
    - Example data for use in your system

Final Project Requirements (cont)

- A short Oral presentation and Demo (Final Exam: 5/4 12pm)
  - Limit yourself to 20 mins.
  - Format:
    - A short verbal explanation of your project
(ideally, this would kill time, as you start-up your demo)
    - A short discussion of the graphics methods used
(Should probably be done during the demo)
    - A live demo using either a laptop or the classroom machine
(verify that it works ahead of time)
    - Your project Web page and compiled demo are the only props allowed for your presentation
    - If your project cannot be presented using the facilities available in the classroom you need to
      - Let me know ahead of time
      - Perhaps show a video of the system working

Project Ideas

- Terrain generation
  - Something like Terragen
  - Simulate erosion
- A video game
  - Cow Quake
  - Cow Pong
- A GUI front end for your ray tracer
  - Place objects, lights, and camera in the scene
  - Edit material properties
  - Position texture maps
- Add global illumination to your ray tracer
  - path tracing
  - photon mapping (maybe just for caustics)
- Shadow maps or shadow volumes

Representing Surfaces

Here is everything in our toolbox for specifying geometry

✓ Vertices

- Points in 3 space
- Can have color, surface reflectance properties, texture coordinates

✓ Polygons

- Piecewise planar surface patches
- Can be used to approximate "smooth" surfaces
- Generally, "filled-in" by interpolating vertex properties

✓ Normals

- Represents the local derivative of a surface (tangent space)
- Used only for shading
- We can make the surface appear smoother, than it really is by making vertex normals different than the actual polygon normals

Classes of Smooth Surfaces

It is easiest to consider Surfaces as analytical entities. However, surfaces (like geometry in general) exist independent of a mathematical formulation.

In fact, under some formulations we will find that it is difficult, or impossible, to describe even the most "simple" shapes.

As a result, we need to arm ourselves with a variety of different formulation schemes for describing surfaces.

Simple Functions

Simple functions are probably the most common mathematical formulation of describing shapes. These are sometimes called "Explicit" representations.

Rules of an explicit representation:

1) Describe one "dependent" variable in terms of "independent" variables.
2) Each unique combination of "independent" variables specifies only one valid value of the "dependent" variable

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

A 3-D surface will have 2 independent variables.

Rendering Simple Functions

Simple Functions are easy to render,

Loop over the independent variables generating vertices and normals

$$v_{ij} = \begin{bmatrix} x_i \\ y_j \\ f(x_i, y_j) \\ 1 \end{bmatrix} \qquad n_{ij} = \begin{bmatrix} -\frac{\partial f(x_i, y_j)}{\partial x} \\ -\frac{\partial f(x_i, y_j)}{\partial y} \\ 1 \\ 0 \end{bmatrix}$$

but the class of surfaces they describe is too limited

Shortcomings of Simple Functions

Consider the following representations of a Plane as a simple function:

$$z = Ax + By + C$$

For any values of A, B, and C, the resulting surface will be a plane, however, not every plane can be specified in this form. For instance, the x-z or y-z planes.

Similarly, we cannot completely describe a sphere centered at the origin as a simple function:

$$z = \sqrt{r^2 - x^2 - y^2}$$

The function above only describes the upper hemisphere, and then only for values of x and y such that $x^2+y^2 <= r^2$.

What are alternatives are there to simple functions for describing surfaces?

Implicit Representations

Many surfaces can be described as implicit functions, in which all variables are independent and are the "zero-set" of a 3-D function.

$$0 = f(x, y, z)$$

This representation treats all dimensions equivalently. As a result, it can describe a wider class of surfaces. For instance, all planes can be described using an implicit function of the form:

$$Ax + By + Cz + D = 0$$

We've used this representation many times before. Likewise, we can describe spheres centered at the origin implicitly:

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

While it is easy to **verify** if a given point, p, satisfies an implicit function (i.e. lies on the surface), it is a difficult task to find such points in the first place.

Algebraic Surfaces

We can break implicit surfaces into interesting subclasses. One particularly interesting subclass are those for which f(x,y,z) is polynomial in the three independent variables. It is interesting, because it forms a vector space (remember what the requirements for a vector space were?). As a result, we can define operations like addition, and multiplication by a scalar for them.

The algebraic surfaces of degree 2, have the following form:

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$$

These surfaces are called the "quadrics", and they include spheres, ellipsoids, paraboliods, disks, and cones.

Implicit functions are more powerful than simple functions, but there is no simple procedural way to generate points on them.

Parametric Functions

In parametric formulations we define a general "parameter space" and provide separate simple functions for each variable as a function of these parameters.

$$x = f_x(u, v)$$

$$y = f_y(u, v)$$

$$z = f_z(u, v)$$

Parametric functions are mappings from a simple "parameter" space to the surface. A common example of a parametric mapping is the sphere:

$$x = r\cos(\theta)\cos(\phi) \quad y = r\sin(\theta)\cos(\phi) \quad z = r\sin(\phi)$$

Rendering Parametric Functions

Parametric functions, like simple functions, are easy to render. Step through the parameter space computing the vertices and normals:

$$v_{ij} = \begin{bmatrix} f_x(u_i, v_j) \\ f_y(u_i, v_j) \\ f_z(u_i, v_j) \\ 1 \end{bmatrix} \qquad n_{ij} = \begin{bmatrix} \frac{\partial f_x(u_i, v_j)}{\partial u} \\ \frac{\partial f_y(u_i, v_j)}{\partial u} \\ \frac{\partial f_z(u_i, v_j)}{\partial u} \\ 0 \end{bmatrix} \times \begin{bmatrix} \frac{\partial f_x(u_i, v_j)}{\partial v} \\ \frac{\partial f_y(u_i, v_j)}{\partial v} \\ \frac{\partial f_z(u_i, v_j)}{\partial v} \\ 0 \end{bmatrix}$$

There is also a special class of "polynomial parametric functions" of the form:

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

Where the degree of the function is m+n, and it has 3(n+1)(m+1) coefficients.

Surface Design

We now have a framework for specifying a wide range of surfaces. In particular, we can generate vertices and normals for a surface given an set of parametric functions. In the case of polynomial function, we need only provide a set of coefficients.

However, specifying a surface this way is very non-intuitive. In general, we would prefer to specify a surface more directly.

For instance we might want to specify points on the surface, or provide other various controls.

To simplify our discussion, we will first consider curves in the plane

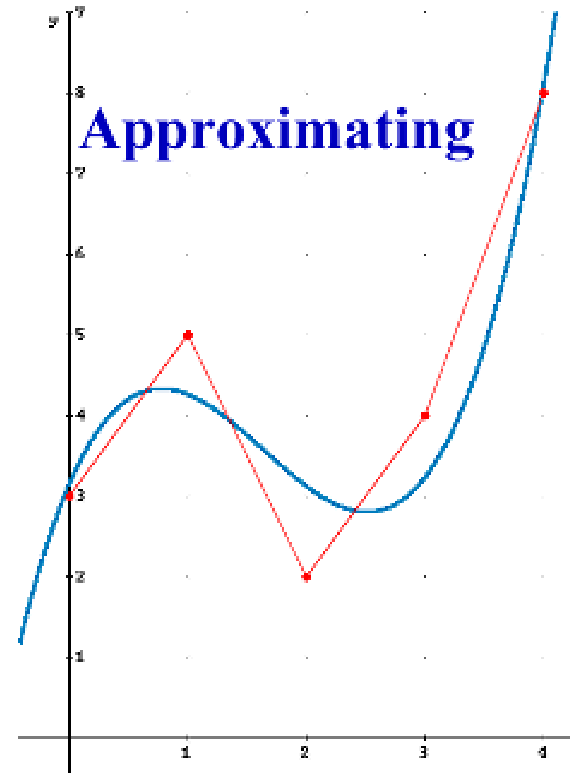
Specifying Curves

Control Points - a set of points that influence the curve's shape

Knots - control points that lie on the curve

Interpolating spline - curve passes through all control points (only knots)

Approximating spline - control points merely influence shape

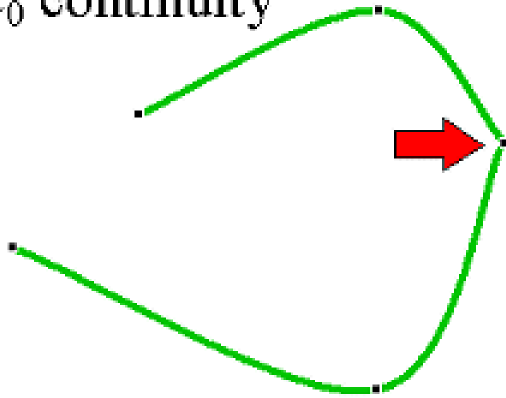


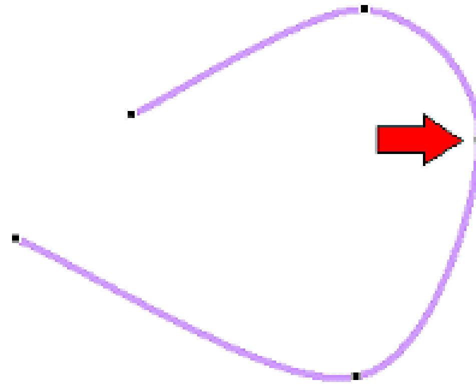
Piecewise Curve Segments

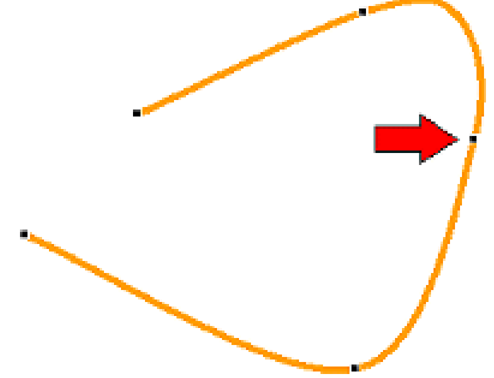
Often we will want to represent a curve as a series of curves pieced together

But we will want these curves to fit together reasonably...

Parametric continuity–







We say that a curve has C^k, or parametric, continuity in the interval $t \in [a,b]$, if all derivatives, up through the kth, exist and are continuous at all points within the interval.

Parametric Cubic Curves

In order to assure C_2 continuity our functions must be of at least degree 3

Here's what a parametric cubic spline function looks like:

$$x = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y$$

Alternatively, it can be written in matrix form:

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Solving for Coefficients

The whole story of polynomial splines is deriving their coefficients

How?

By satisfying constraints given knots and continuity conditions

An Illustrative Example

Cubic Hermite Splines:

Specified by 2 knots and 2 tangent vectors at the curve's endpoints

The Gradient of a Cubic Spline

Expressions for the tangent vectors can be found by taking derivatives of the parametric formula. These derivatives are also functions of unknown coefficients.

$$\begin{bmatrix} \frac{dx}{dt} & \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Hermite Specification

Here is the full specification of the Hermite constraints given in the form of a Matrix Equation

$[t^3, t^2, t, 1]$ evaluated at $t = 0$

$[t^3, t^2, t, 1]$ evaluated at $t = 1$

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

$[3t^2, 2t, 1, 0]$ evaluated at $t = 0$

$[3t^2, 2t, 1, 0]$ evaluated at $t = 1$

Solve for the Hermite Coefficients

Finding the coefficients it is a simple matter of algebra.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Spline Basis and Geometry Matrices

In this form, we give special names to each term of our spline specification

$$\underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Cubic Hermite Spline Equation

Now we have a full specification of our curve:

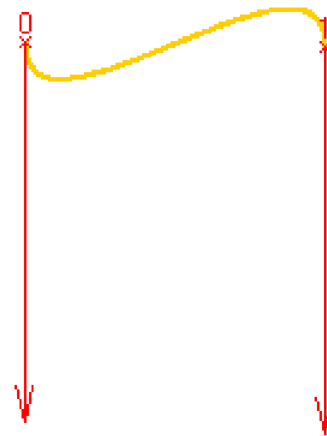
$$[x \quad y] = [t^3 \quad t^2 \quad t \quad 1] \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

Hermite Spline Demonstration

Discussion:

- Is a tangent vector *really* an intuitive control?

- Piecewise issues:
  - C_0 easy
  - C_1 reasonable



Another Way to Think About Splines

The contribution of each geometric factor can be considered separately, this approach gives a so-called *blending function* associated with each factor.

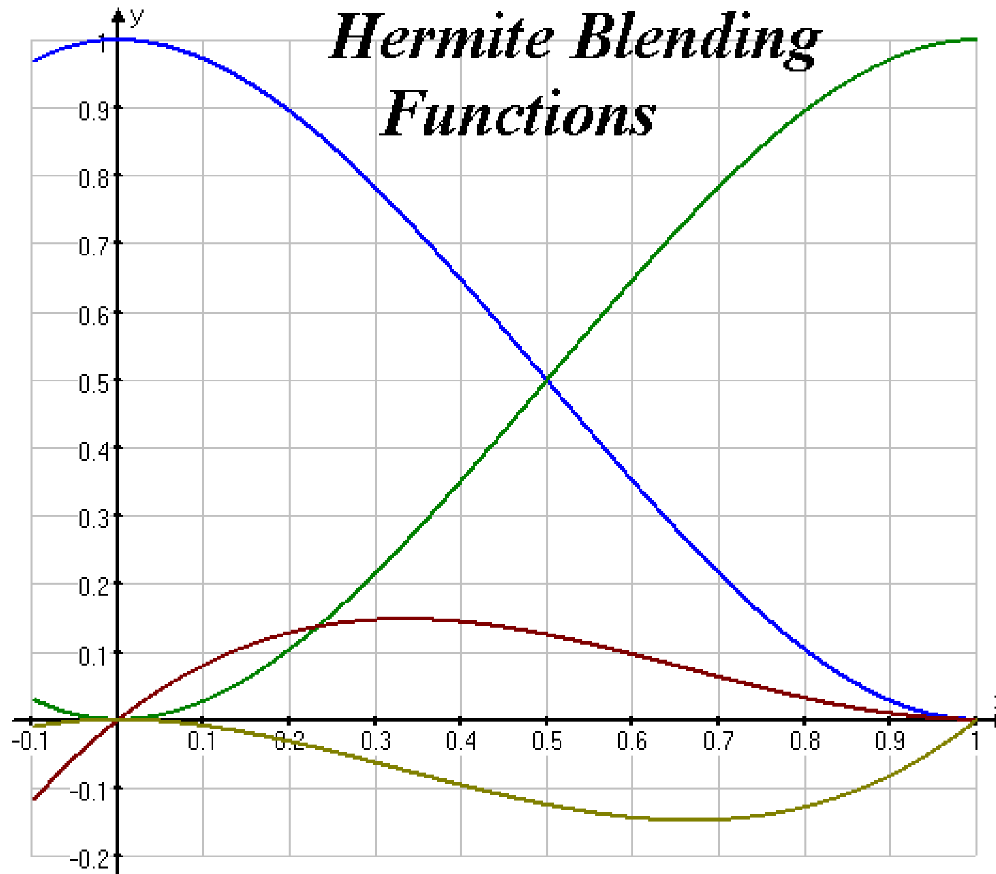
Beginning with our spline formulation:

$$[x \quad y] = [t^3 \quad t^2 \quad t \quad 1] \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

Reordering our multiplications gives:

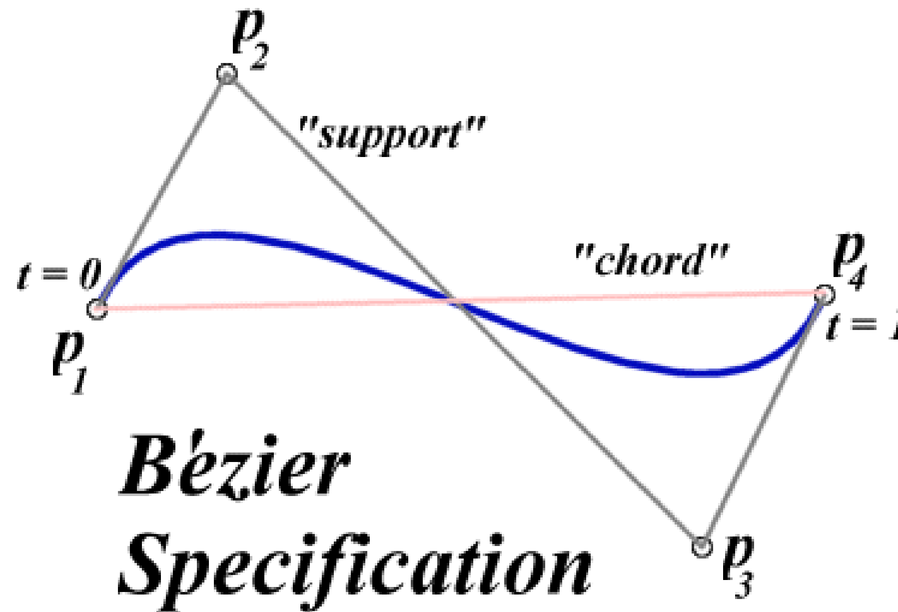
$$p(t) = \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

Hermite Blending Functions



One More Example

- Cubic Hermite Splines present some user friendliness problems. Next we will define a new Spline class that has more intuitive controls



- A Cubic Bezier Spine has four control points, two of which are knots. The other two form a "scaffolding" around the curve

Coefficients for Cubic Bezier Splines

It just so happens that the gradients at the knots of a Bezier Spline can be expressed in terms of the adjacent control points:

$$\nabla p_1 = 3(p_2 - p_1)$$
$$\nabla p_4 = 3(p_4 - p_3)$$

Note: this approach is entirely synthetic!
(Made up… you'll see the real motivation in a few of slides)

Using such a specification is reasonable, but what makes 3 a magic number?

Here's the Trick!

Knowing this we can formulate a Bezier spline in terms of the Hermite geometry spec

$$\underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

And substituting gives:

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Basis and Geometry Matrices for Bezier Splines

Now we can compute our spline coefficients given a Bezier Specification

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Bezier}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Bezier Blending Functions

The reasonable justification for Bezier spline basis can only be approached by considering its blending functions:

$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

This family of polynomials (called order-3 Bernstein polynomials) have the following unique properties:

- They are all positive in the interval [0, 1]
- Their sum is equal to 1 (Where have we seen this before?)

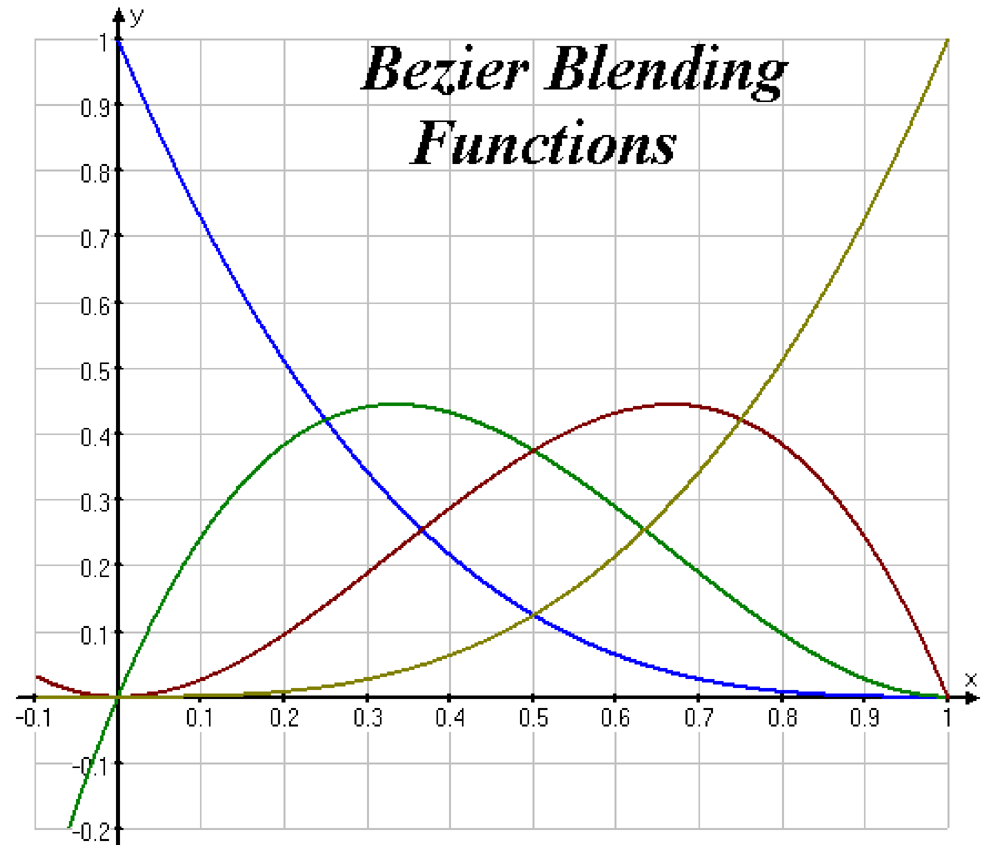
Plots of Bezier Blending Functions

Thus, *every* point on the curve is an Affine combination of the control points.

Since the sum of these blending weights is 1.

Moreover, the weights of this combination are all positive.

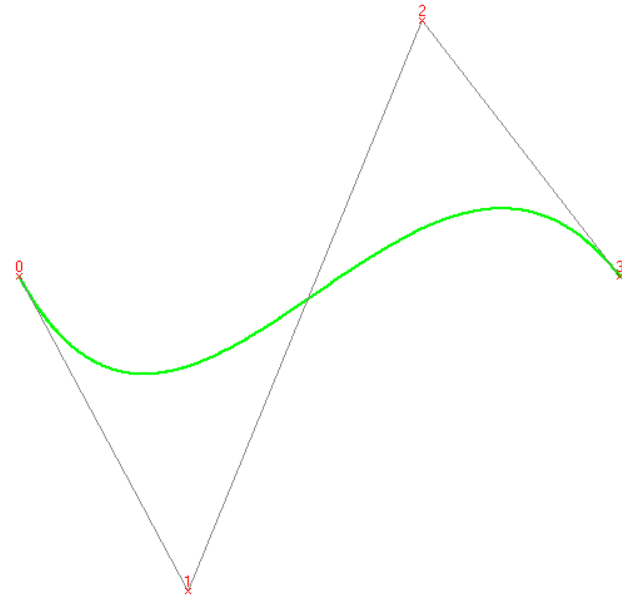
Thus, the curve is also a *Convex combination* of the control points!



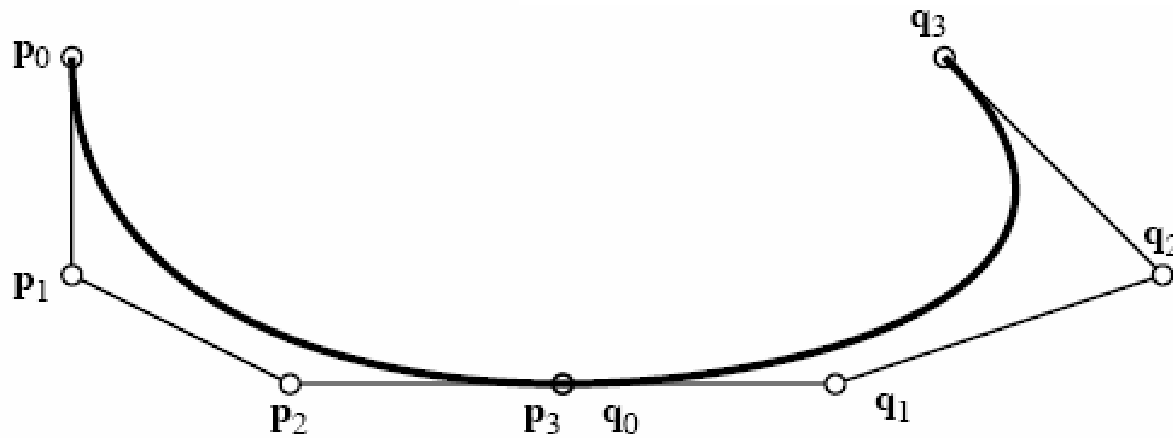
Bezier Demonstration

Discussion:

- Strange mix of points on and off the curve



- Piecewise issues:
- C_0 easy
- C_1 easy



Other Cubic Spline Types

We can specify the slope at knots points using the control points on either side, and parameterize the weighting factors:

$$\nabla p_2 = s(p_3 - p_1)$$
$$\nabla p_3 = s(p_4 - p_2)$$

Repeating the same derivation gives the specification of a Cardinal Spline:

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{cardinal}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{cardinal}}$$

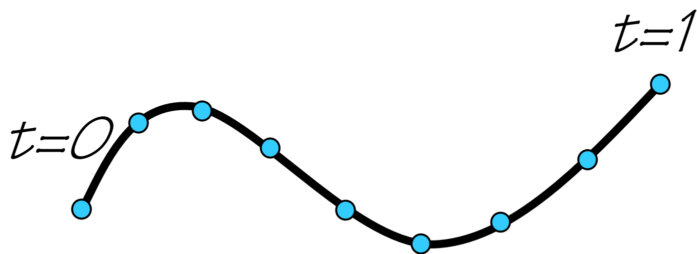
Spline Rendering: Take 1

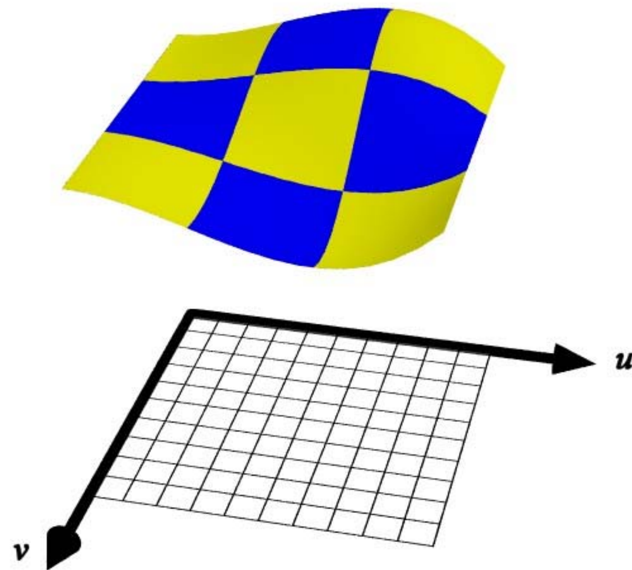
Step 1. Given a spline specification, compute the coefficients by multiplying the spline's basis matrix by the geometry vector.

Step 2. Take uniform steps in the parameter space (t = 0, 0.1, 0.2, …, 1.0), and generate new points on the curve

Step 3. Connect these points with line segments

This method can easily can be easily be adapted from curves to surfaces.

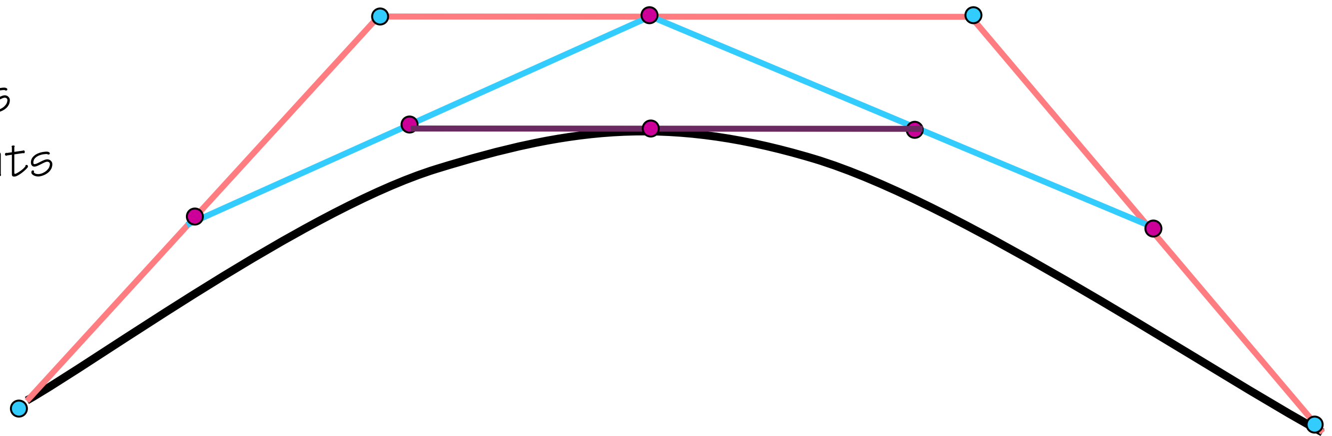




Spline Rendering: Take 2

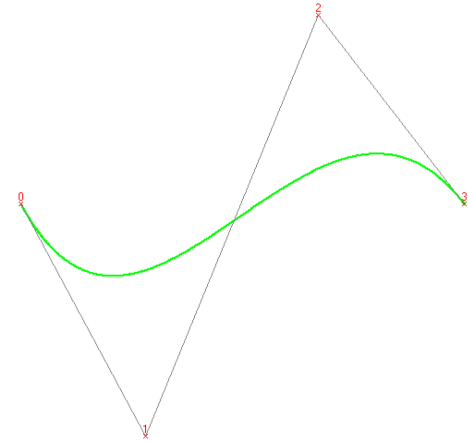
There is a very clever alternative method for rendering Bezier Splines using the "de Casteljau" Algorithm. Basically, we can take our set of control points and recursively generate new control points for arbitrary fractions of the domain.

1. Find midpoints of support
2. Connect with new segments
3. Find midpoints of new segments
4. Connect with new segment
5. Find its midpoint



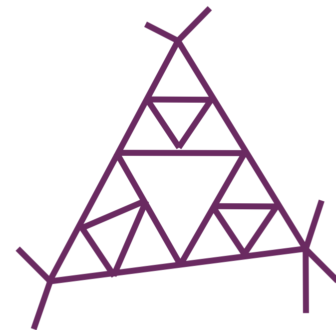
Secret: Subdivision

This process can be repeated recursively... The resulting scaffolding is a good approximation of the actual surface.



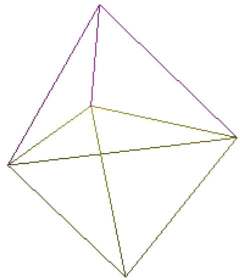
Why use subdivision (recursion) instead of uniform domain sampling (iteration)?

- Stopping conditions can be based on local shape properties (curvature)
- Subdivision can be generalized to non-square domains, in particular to triangular

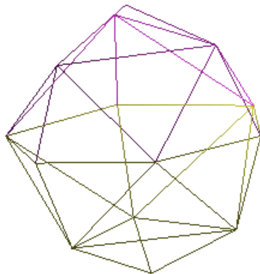


Example of Generalized Subdivision

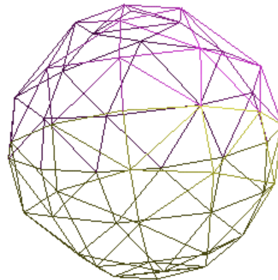
Here is a sample of generalized subdivision:



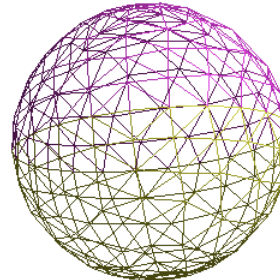
0-levels



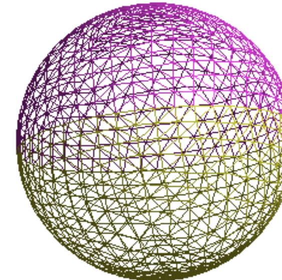
1-level



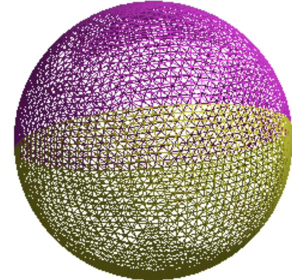
2-levels



3-levels

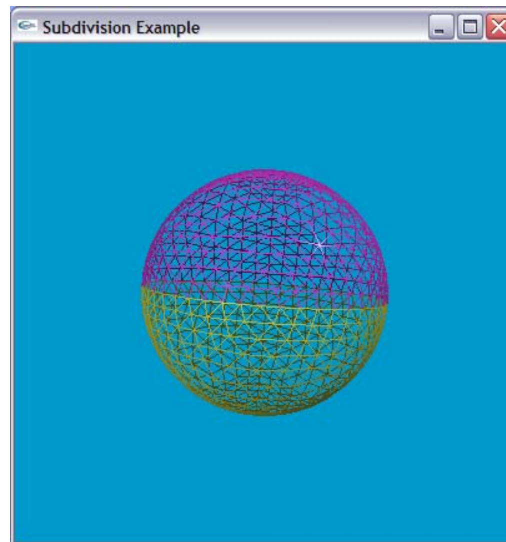


4-levels



5-levels

Here is a quick demo:



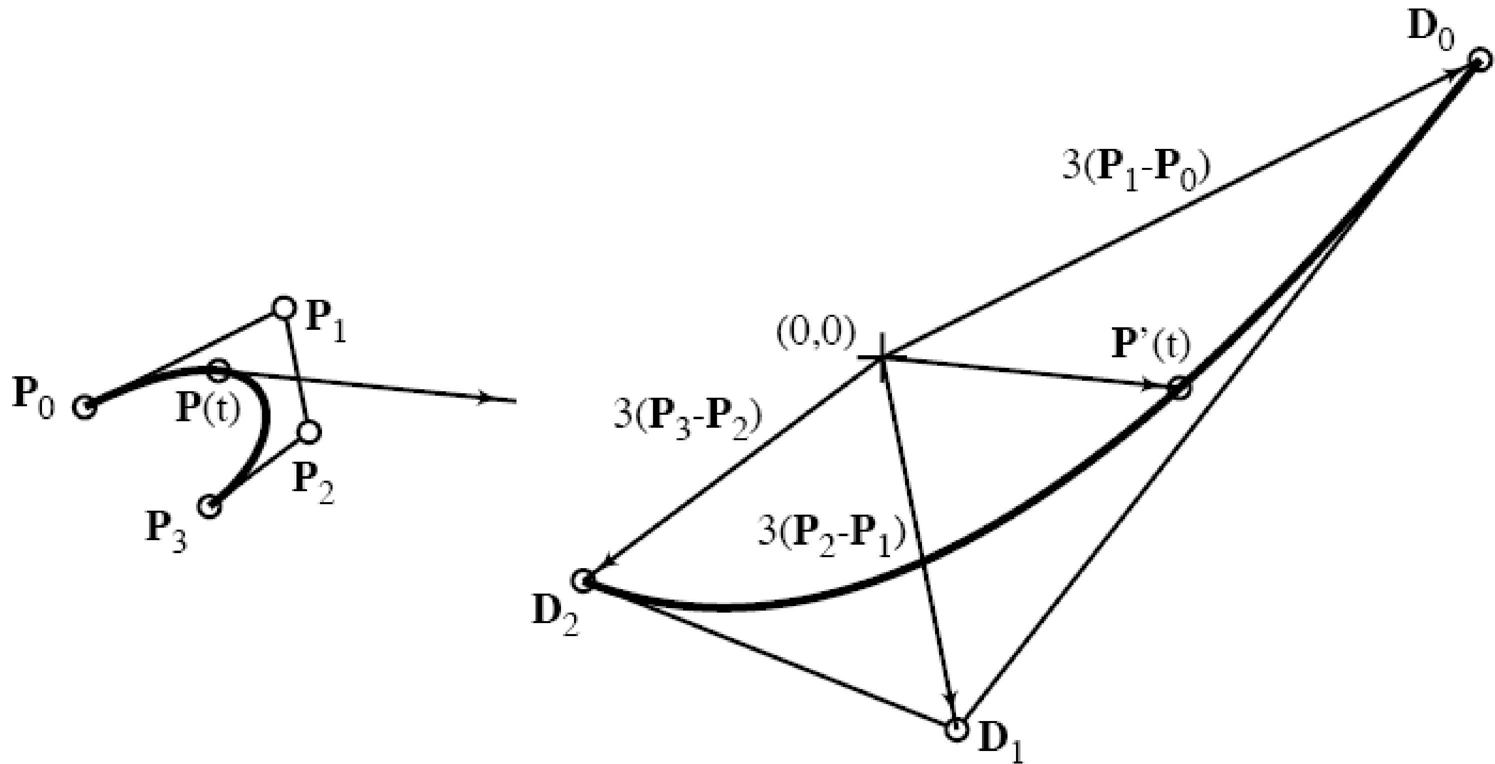
Implementing Subdivision

```
def drawSphere():
    v1 = Vector(0, 1, 0)
    v2 = Vector(1, 0, 0)
    v3 = Vector(math.cos(2*math.pi/3), 0, math.sin(2*math.pi/3))
    v4 = Vector(math.cos(4*math.pi/3), 0, math.sin(4*math.pi/3))
    v5 = Vector(0, -1, 0)
    # top
    Color = [0.7, 0.2, 0.7, 1.0]
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, Color)
    subSphere(v1, v2, v3, 0)
    subSphere(v1, v3, v4, 0)
    subSphere(v1, v4, v2, 0)
    # bottom
    Color = [0.7, 0.7, 0.2, 1.0]
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, Color)
    subSphere(v5, v2, v3, 0)
    subSphere(v5, v3, v4, 0)
    subSphere(v5, v4, v2, 0)
```

Implementing Subdivision

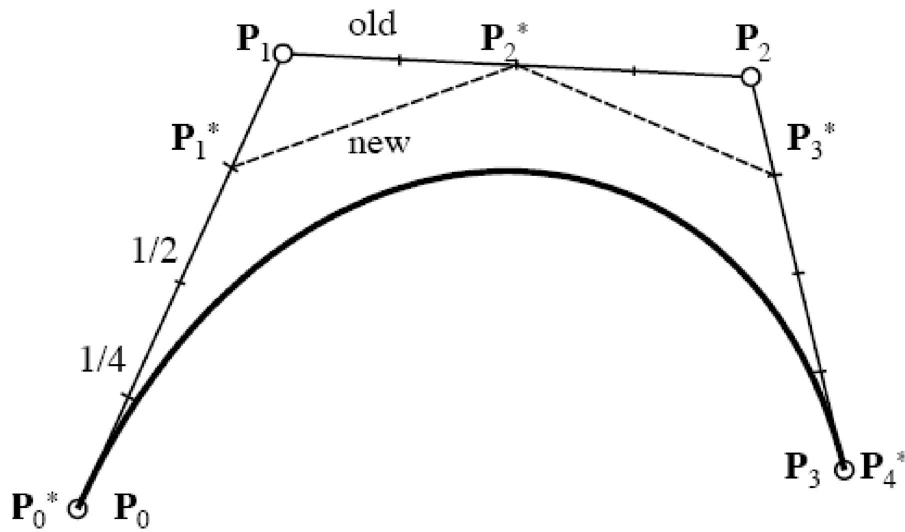
```
def subSphere(v1, v2, v3, depth):
    if (depth == level):
        t1 = Vector(v1)
        t2 = Vector(v2)
        t3 = Vector(v3)
        glBegin(drawMode)
        glNormal3d(v1.x, v1.y, v1.z)
        glVertex3d(t1.x, t1.y, t1.z)
        glNormal3d(v2.x, v2.y, v2.z)
        glVertex3d(t2.x, t2.y, t2.z)
        glNormal3d(v3.x, v3.y, v3.z)
        glVertex3d(t3.x, t3.y, t3.z)
        glEnd()
    else:
        v12 = Vector(0.5*(v1.x + v2.x), 0.5*(v1.y + v2.y), 0.5*(v1.z + v2.z)).normalize()
        v23 = Vector(0.5*(v2.x + v3.x), 0.5*(v2.y + v3.y), 0.5*(v2.z + v3.z)).normalize()
        v31 = Vector(0.5*(v3.x + v1.x), 0.5*(v3.y + v1.y), 0.5*(v3.z + v1.z)).normalize()
        subSphere( v1, v12, v31, depth+1)
        subSphere(v12,  v2, v23, depth+1)
        subSphere(v31, v12, v23, depth+1)
        subSphere(v31, v23,  v3, depth+1)
```

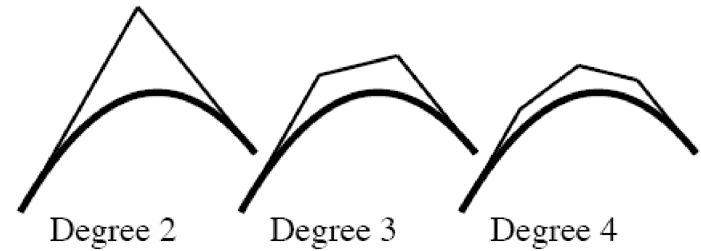
Derivatives of Bezier Curve

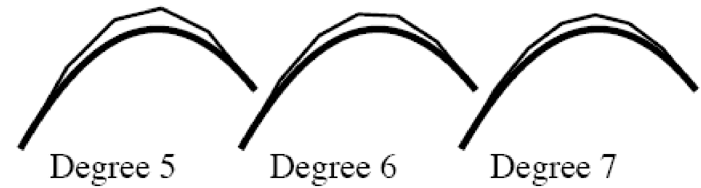


$$\mathbf{D}_i = n(\mathbf{P}_{i+1} - \mathbf{P}_i)$$

Degree Elevation





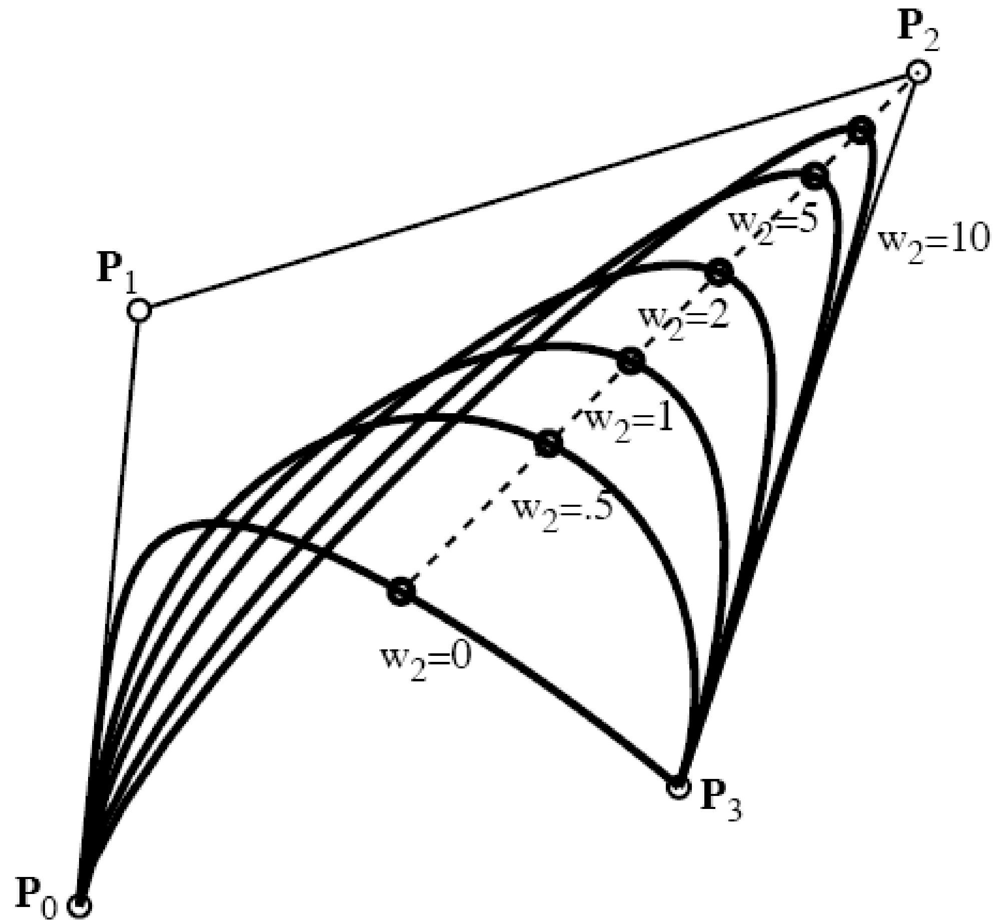


$$\mathbf{P}_i^* = \alpha_i \mathbf{P}_{i-1} + (1 - \alpha_i)\mathbf{P}_i, \qquad \alpha_i = \frac{i}{n+1}.$$

Rational Bezier Curves

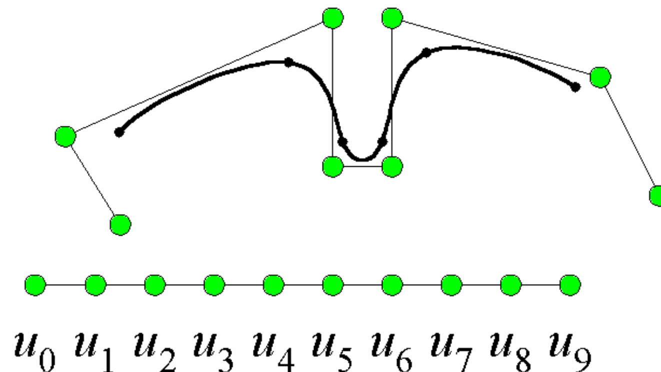
- Advantages
  - invariant under projection
  - can represent conic sections
  - added control

$$\frac{\sum_{i=0}^{n} w_i B_i^n(t) \mathbf{P}_i}{\sum_{i=0}^{n} w_i B_i^n(t)}$$



B-Splines

- Moving a control point on a Bezier affects the whole curve
- Can break curve up into a series of Bezier curves for more local control, but maintaining continuity is a hassle
- B-Splines are a generalization of a string of Bezier curves



Blossoms

- Blossoms have 3 important properties
  - Symmetry: Order of arguments doesn't matter

    $\mathbf{P}(0,1,2)=\mathbf{P}(1,2,0)=\mathbf{P}(0,2,1)$
  - Multi-affinity: We can generate points with new arguments by linear combinations with other points

    $$\mathbf{P}(\alpha r+\beta s,*)=\alpha\mathbf{P}(r,*)+\beta\mathbf{P}(s,*)$$
  - Diagonality: If all arguments are the same we get a point on the curve

    $\mathbf{P}(t,t,t)$
- de Casteljau

$$
\begin{array}{llll}
\mathbf{P}_0=\mathbf{P}(0,0,0) & & & \\
\mathbf{P}_1=\mathbf{P}(0,0,1) & \mathbf{P}(0,0,t) & & \\
\mathbf{P}_2=\mathbf{P}(0,1,1) & \mathbf{P}(0,t,1) & \mathbf{P}(0,t,t) & \\
\mathbf{P}_3=\mathbf{P}(1,1,1) & \mathbf{P}(t,1,1) & \mathbf{P}(t,t,1) & \mathbf{P}(t,t,t)
\end{array}
$$

B-Spline

- Consists of knot vector and control points
  - knot vector for cubic B-spline $[t_0 \quad t_1 \quad 1 \quad 2 \quad 4 \quad 5 \quad t_6 \quad t_7]$
    - specifies bezier curves over parameter values [1,2], [2,4],[4,5]
    - extra entries on the end specify end conditions
  - control points are:

$$\mathbf{P}(t_1, t_2, 1), \mathbf{P}(t_2, 1, 2), \mathbf{P}(1, 2, 4),$$
$$\mathbf{P}(2, 4, 5), \mathbf{P}(4, 5, t_6), \mathbf{P}(5, t_6, t_7)$$

- de Boor evaluation
  - suppose we have cubic B-spline with knot vector $[u_0 \quad u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5]$
  - to evalute at $u \in [u_2, u_3]$

$$\begin{array}{llll}
\mathbf{P}_0 = \mathbf{P}(u_0, u_1, u_2) & & & \\
\mathbf{P}_1 = \mathbf{P}(u_1, u_2, u_3) & \mathbf{P}(u, u_1, u_2) & & \\
\mathbf{P}_2 = \mathbf{P}(u_2, u_3, u_4) & \mathbf{P}(u, u_2, u_3) & \mathbf{P}(u, u, u_2) & \\
\mathbf{P}_3 = \mathbf{P}(u_3, u_4, u_5) & \mathbf{P}(u, u_3, u_4) & \mathbf{P}(u, u, u_3) & \mathbf{P}(u, u, u)
\end{array}$$

NURBS

- Non-Uniform Rational B-Splines
  - interval between knot values need not be 1
  - control points have weights

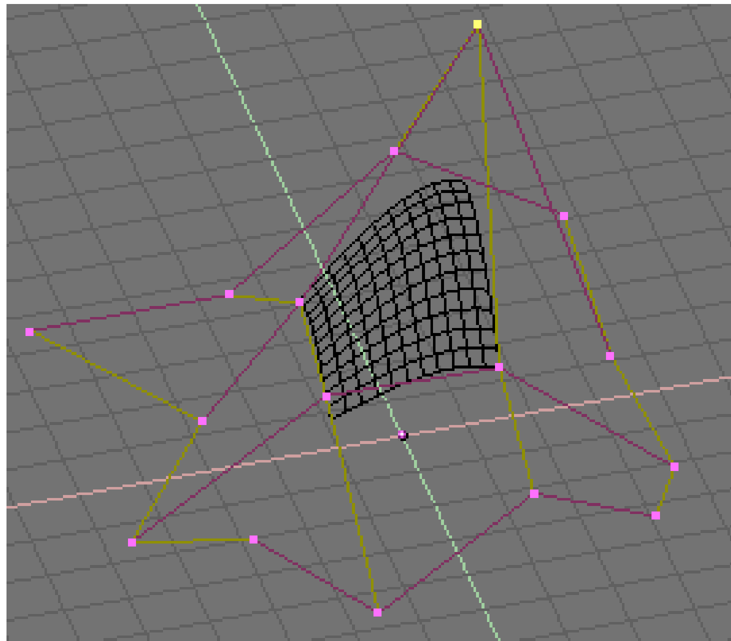


image from www.ecclectica.ca/issues/2004/2/baumgarnter.asp