

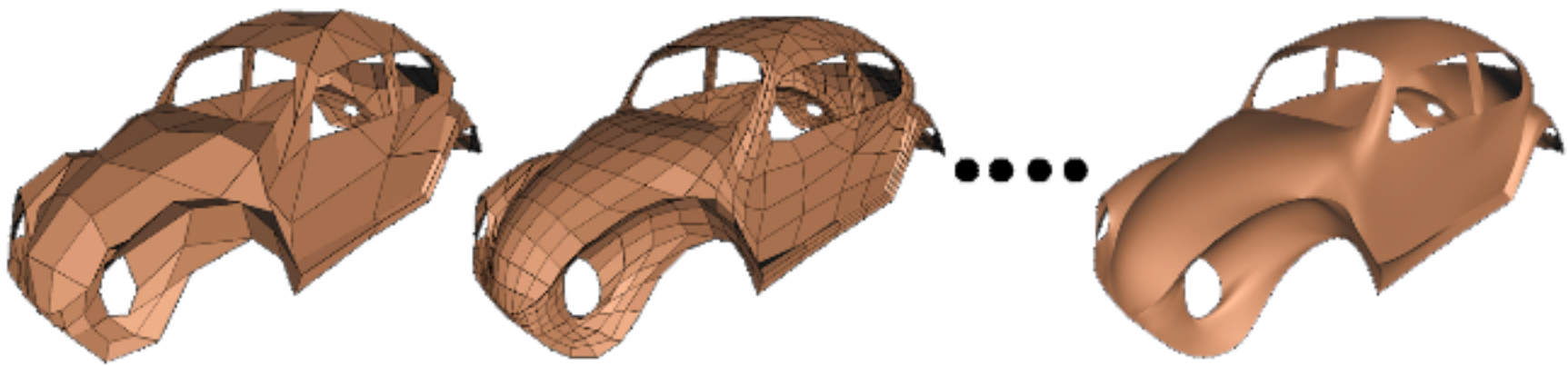
**An Introduction to  
Geometric Modeling**

**Ron Goldman**

**Department of Computer Science**

**Rice University**

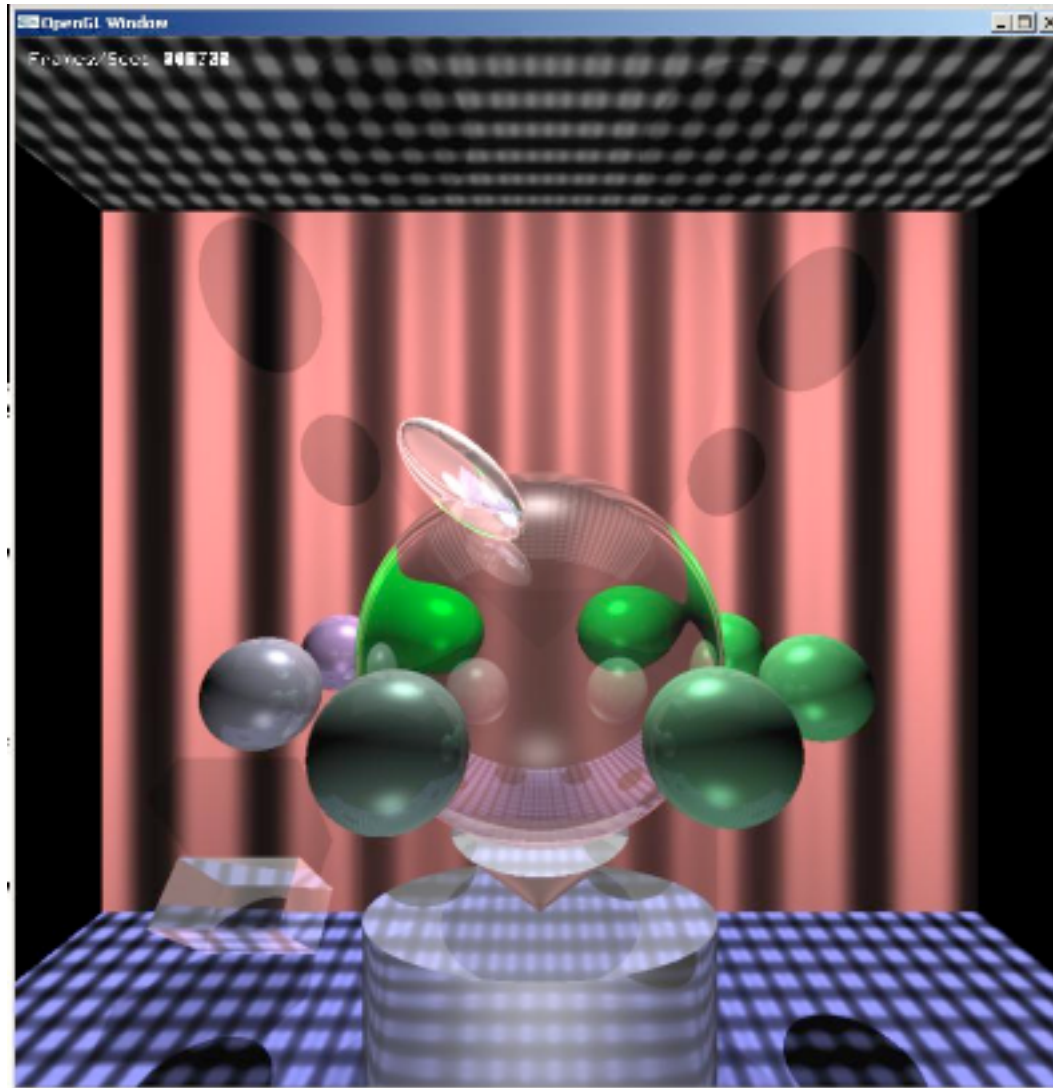
## Car Bodies -- Scott Schaefer



## Geri's Game -- Disney/Pixar



# Computer Graphics -- Anonymous Rice Undergraduate



## **Geometric Modeling**

### *Curves and Surfaces (and Solids)*

- Representation -- Implicit, Parametric, Algorithmic
- Manipulation -- Affine and Projective Transformations, Deformations
- Analysis -- Intersection, Curvature, Singularities, Mass Properties
- Reconstruction -- Reverse Engineering, Data Compression (Wavelets)

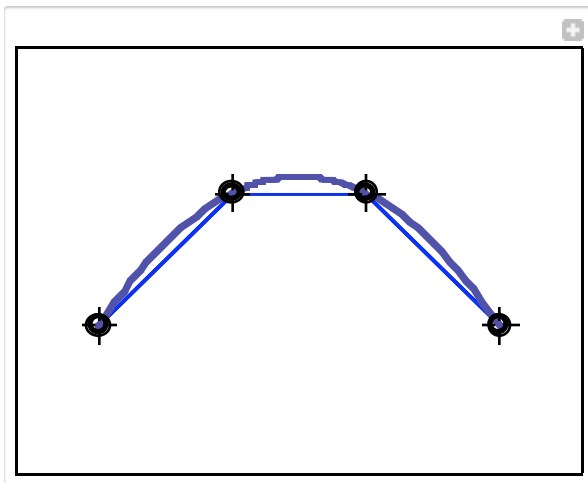
### *Mathematical Roots*

- Approximation Theory and Numerical Analysis
- Algebraic Geometry and Differential Geometry

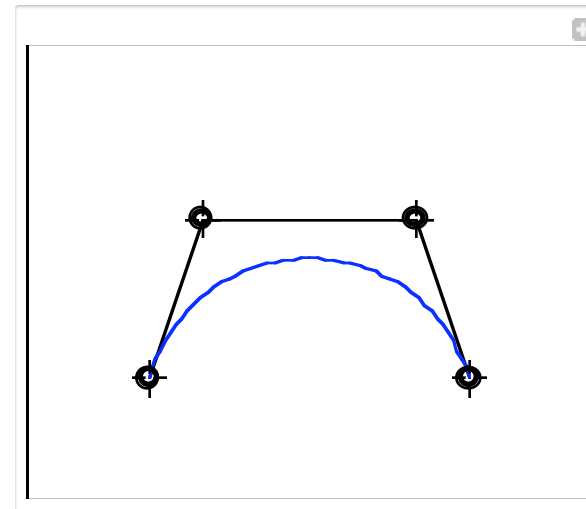
**Part I:**  
**Approximation Theory**

## Tension between Mathematics and Engineering

1. How do Mathematicians and Computer Scientist actually represent curves and surfaces?
  - Algebra -- Formulas and Algorithms  
--  $y = x^2$        $x^2 + y^2 = 1$
2. How do Scientists and Engineers want to represent curves and surfaces?
  - Geometry -- Interpolation and Approximation



*Interpolation*

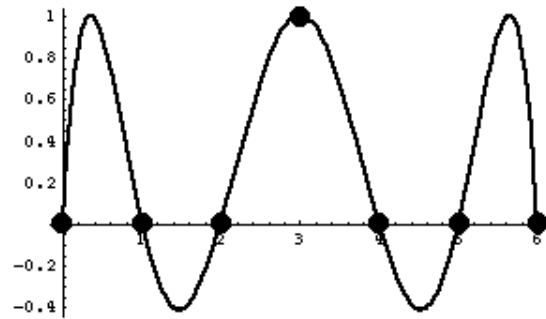


*Approximation*

## Tension between Science and Engineering

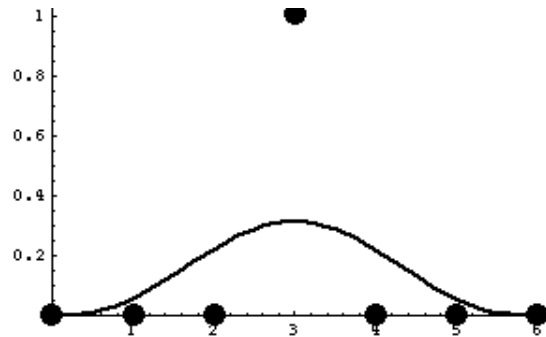
1. How do Scientists need to represent curves and surfaces?

- Interpolate the Data



2. How do Engineers need to represent curves and surfaces?

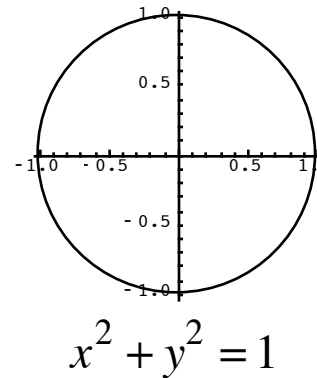
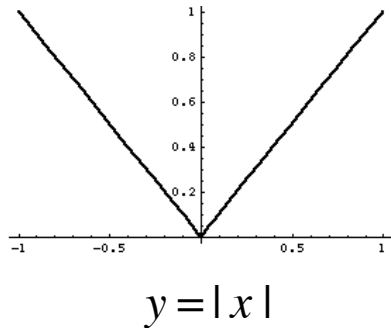
- Approximate the Shape



## Tension between Mathematics and Computer Science

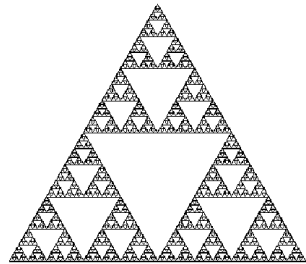
1. How do Mathematicians typically represent curves and surfaces?

- Formulas and Equations

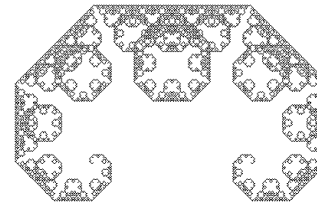


2. How do Computer Scientists naturally represent curves and surfaces?

- Algorithms and Procedures



Sierpinski Gasket



C-Curve

## Themes

1. Algebra vs. Geometry
2. Interpolation vs. Approximation
3. Formulas vs. Algorithms

**Part IA:**  
**Lagrange Interpolation**

## Parametric Representations

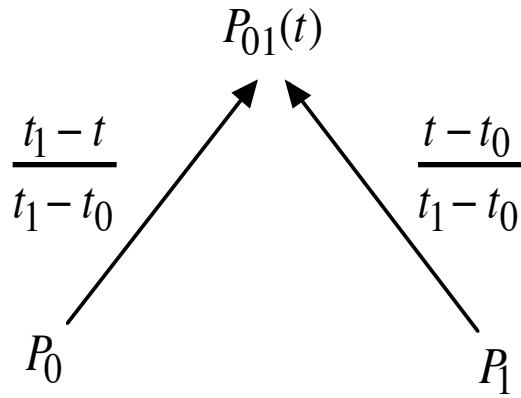
### *Parametric Curves in the Plane*

- $C : I \subset \mathbf{R} \rightarrow \mathbf{R}^2$ 
  - $C(t) = (x(t), y(t))$
  - $C(t) = (\cos(t), \sin(t))$

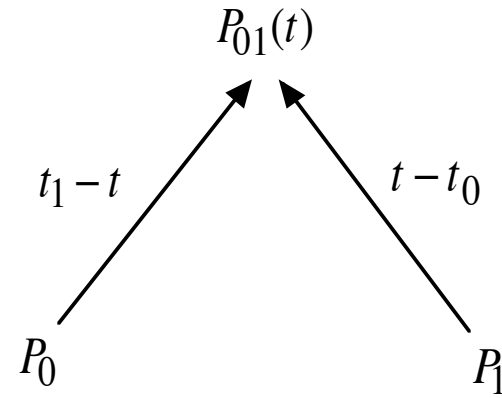
### *Control Points and Blending Functions*

- $C(t) = \sum_{k=0}^n B_k(t)P_k = \left( \sum_{k=0}^n B_k(t)x_k, \sum_{k=0}^n B_k(t)y_k \right)$ 
  - $B_k(t) = \text{Blending Functions}$
  - $P_k = (x_k, y_k) = \text{Control Points}$

## Linear Interpolation



Normalized

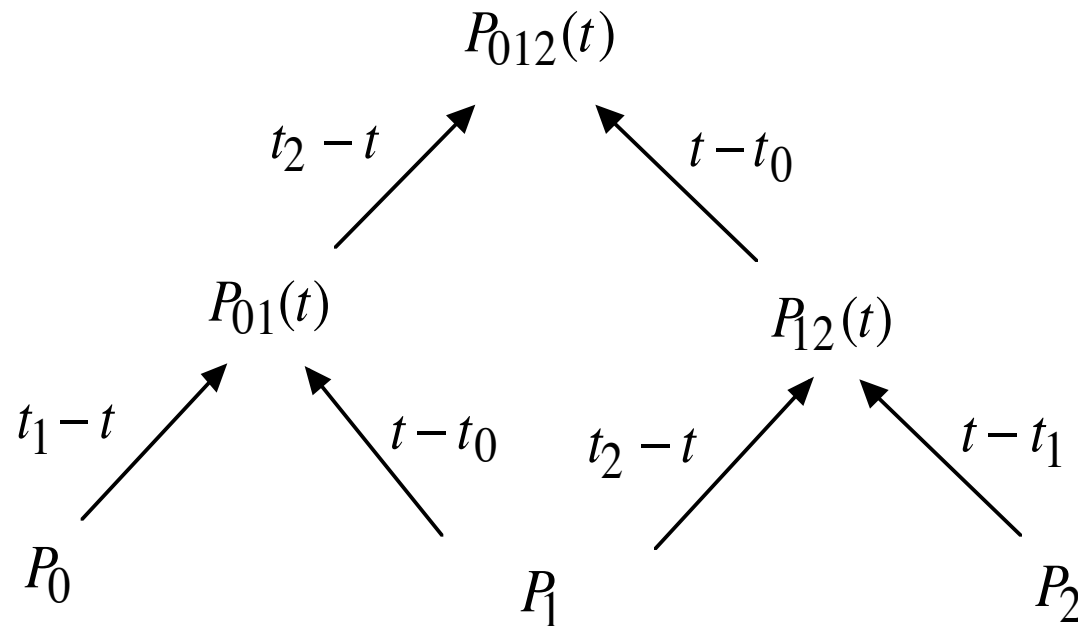


Unnormalized

$$P_{01}(t) = \frac{t_1 - t}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1$$

$$P_{01}(t_0) = P_0 \quad P_{01}(t_1) = P_1$$

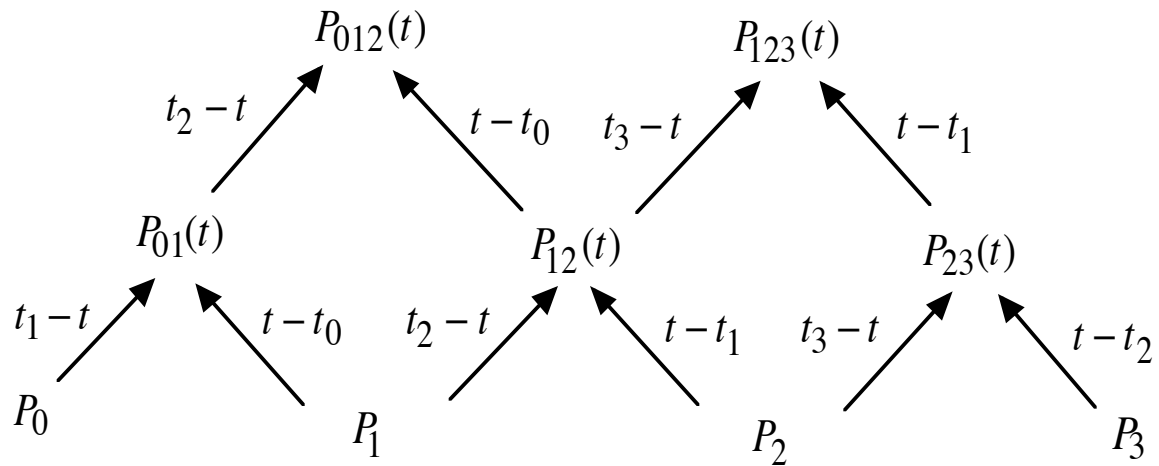
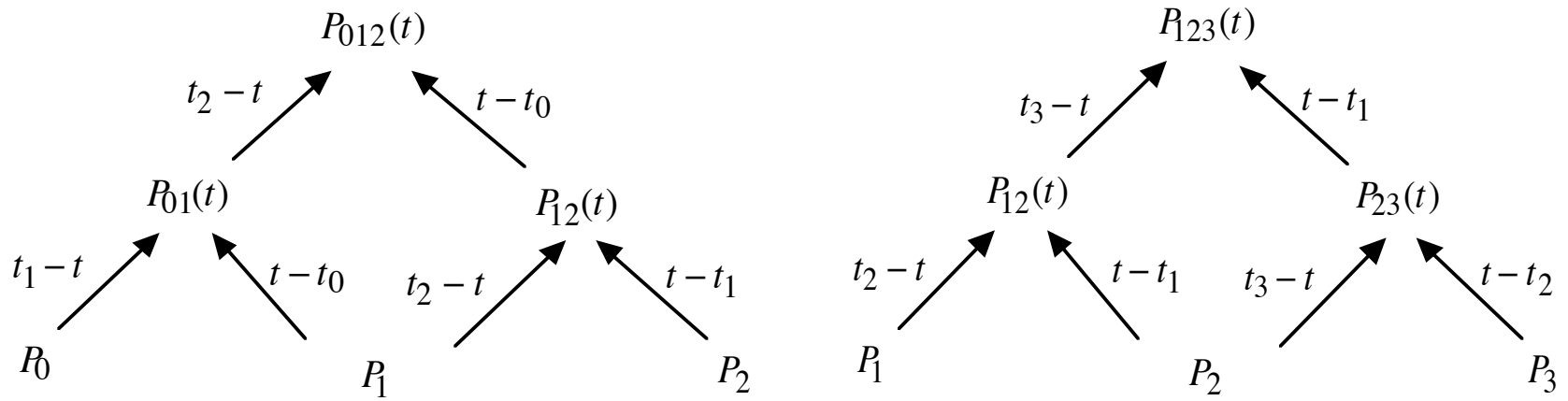
## Neville's Algorithm for Quadratic Interpolation



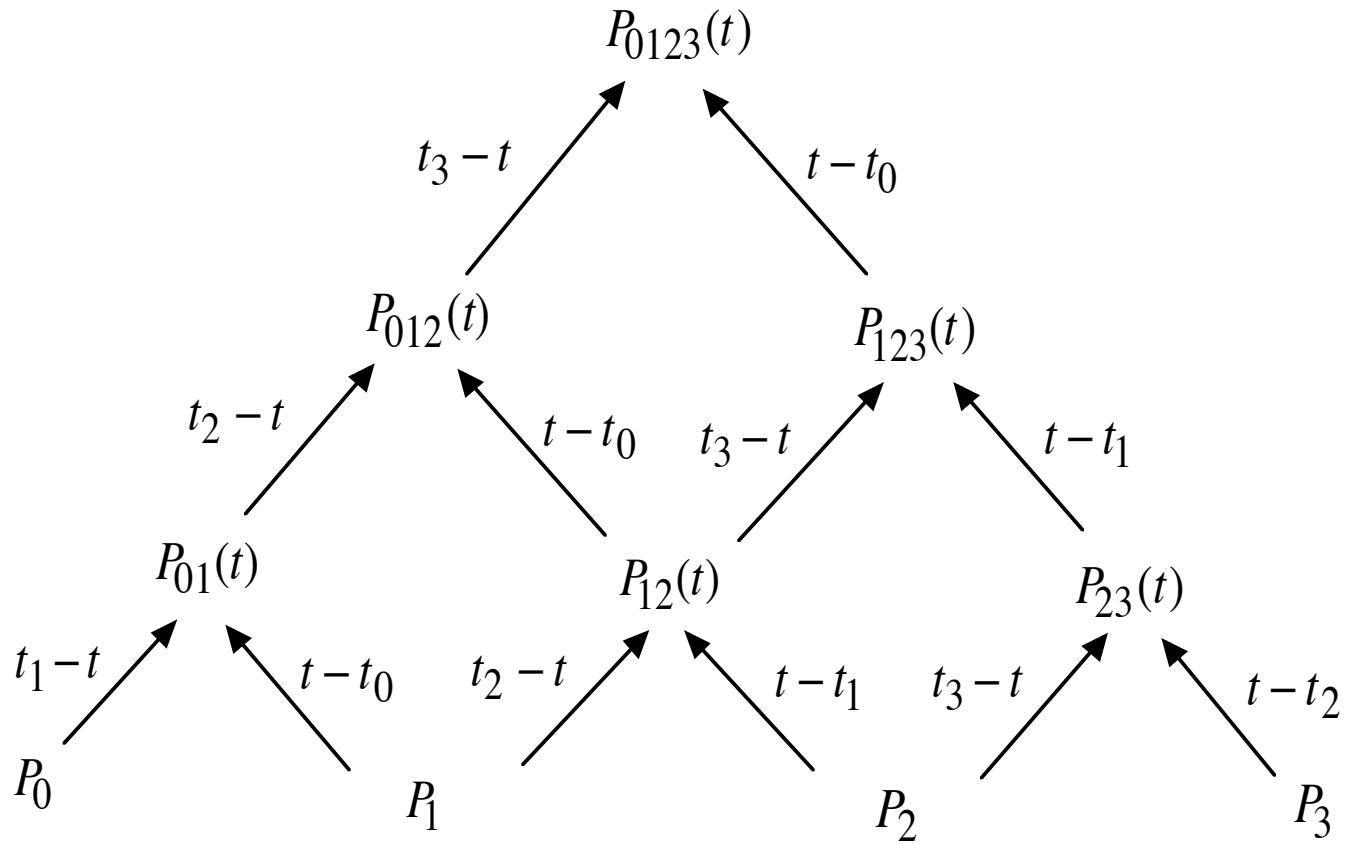
$$P_{012}(t) = \frac{t_2 - t}{t_2 - t_0} P_{01}(t) + \frac{t - t_0}{t_2 - t_0} P_{12}(t)$$

$$P_{012}(t_k) = P_k$$

## Neville's Algorithm for Two Quadratic Curves



## Neville's Algorithm for Cubic Curves



$$P_{0123}(t) = \frac{t_3-t}{t_3-t_0}P_{012}(t) + \frac{t-t_0}{t_3-t_0}P_{123}(t)$$

$$P_{0123}(t_k) = P_k$$

## Advantages of Neville's Algorithm

### 1. *Numerically Stable*

- Uses the Given Data Directly
- No Need to Represent the Polynomial in the Basis  $1, t, t^2, \dots$

### 2. *Fast*

- Dynamic Programming
- $O(n^2)$  vs.  $O(2^n)$

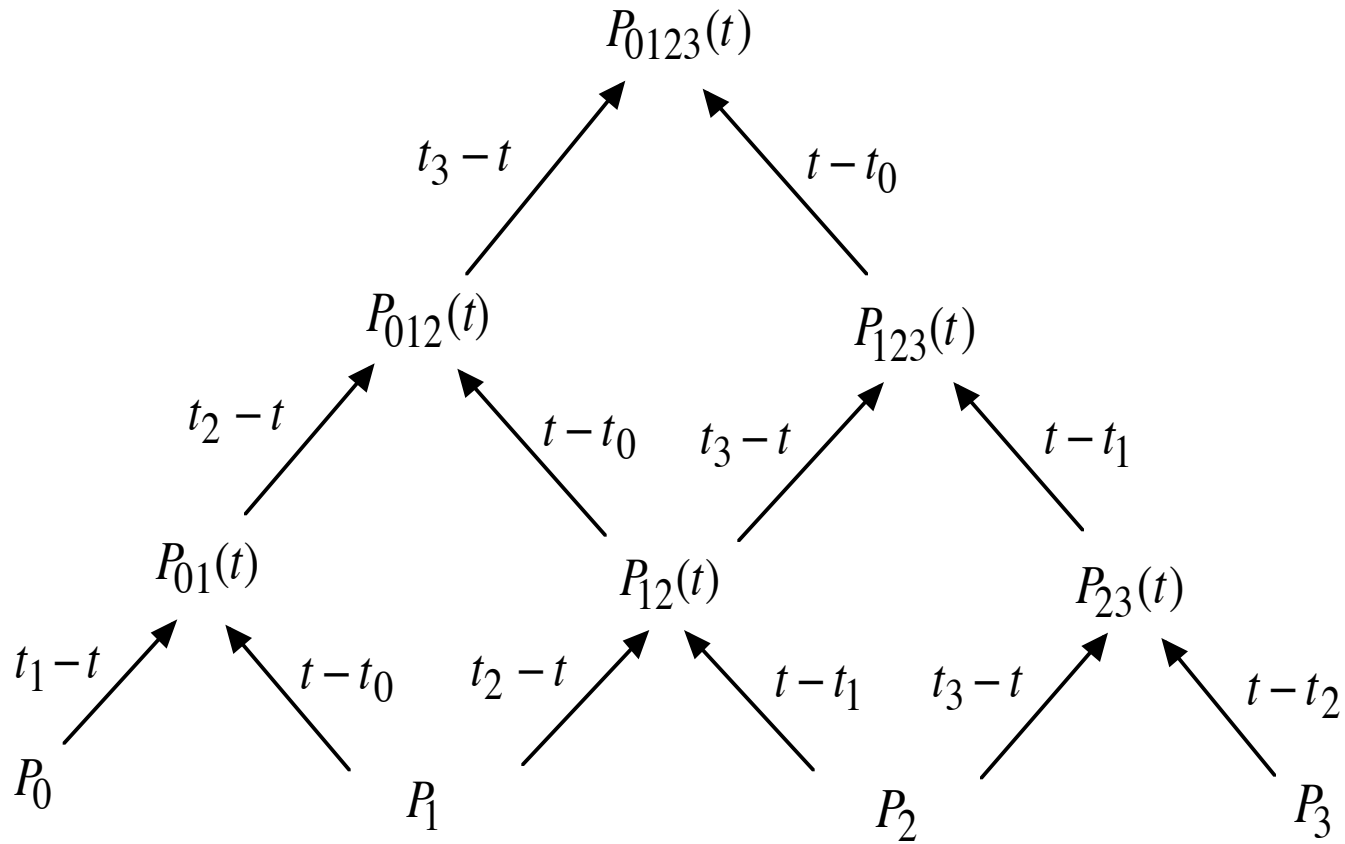
### 3. *Simple Structure*

- Parallel Property
- Strip off First and Last Indices

### 4. *Easy to Update*

- Add a Computation of  $O(n)$  Instead of Redoing Work of  $O(n^2)$ .

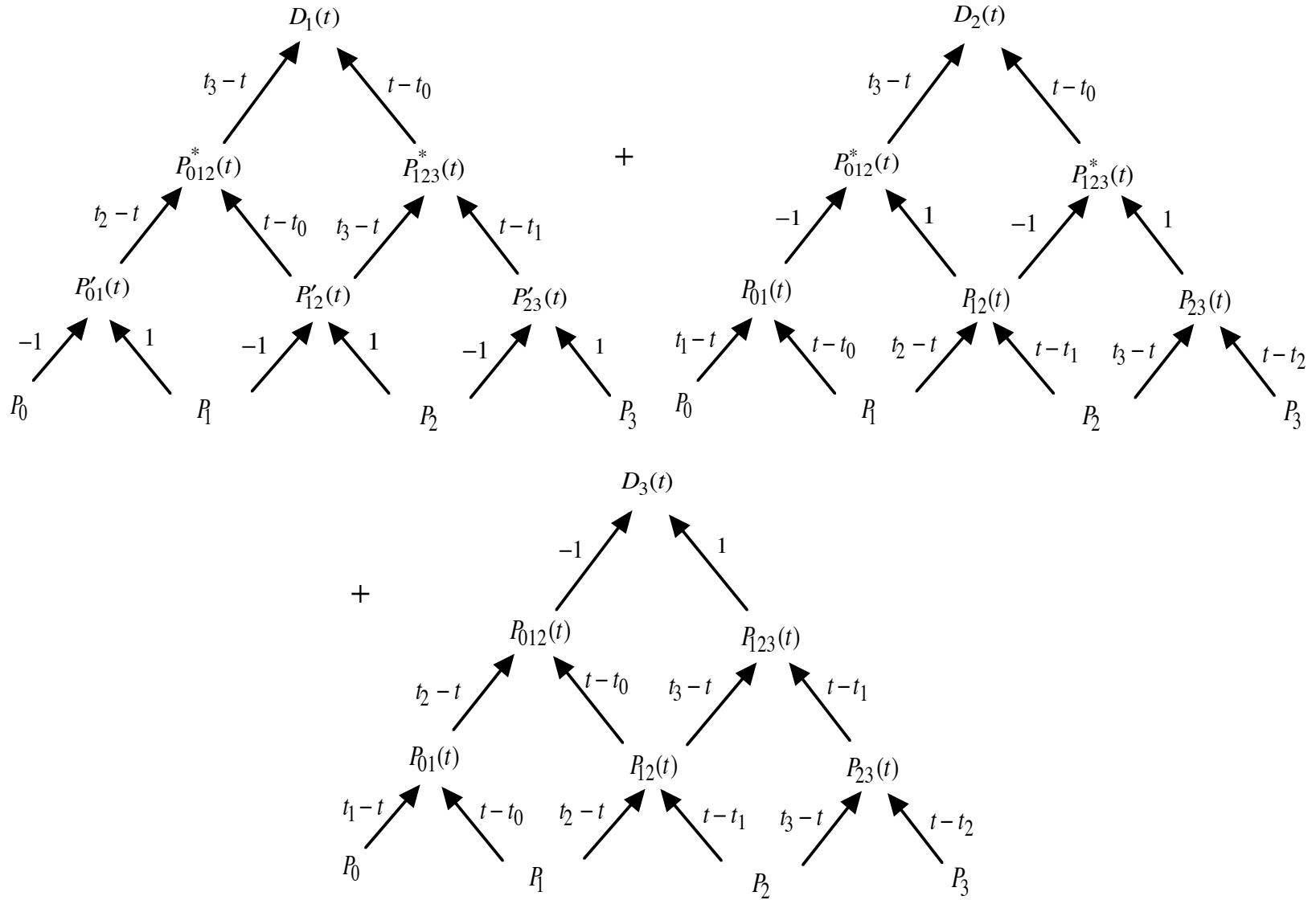
## Neville's Algorithm for Cubic Curves



$$P_{0123}(t) = \frac{t_3 - t}{t_3 - t_0} P_{012}(t) + \frac{t - t_0}{t_2 - t_0} P_{123}(t)$$

$$P_{0123}(t_k) = P_k$$

## Differentiating Neville's Algorithm for Cubic Curves



## Lagrange Interpolation

### *Explicit Interpolation Formula*

- $$P_{0\dots n}(t) = \sum_{k=0}^n P_k L_k^n(t | t_0, \dots, t_n)$$

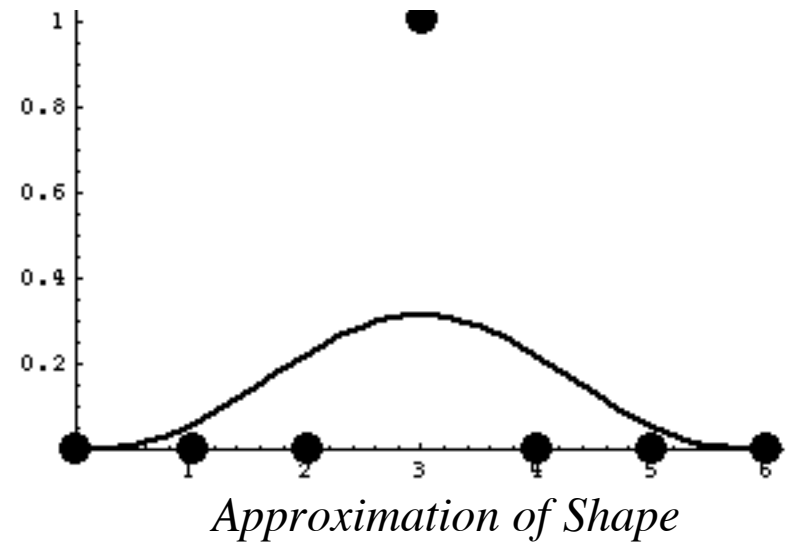
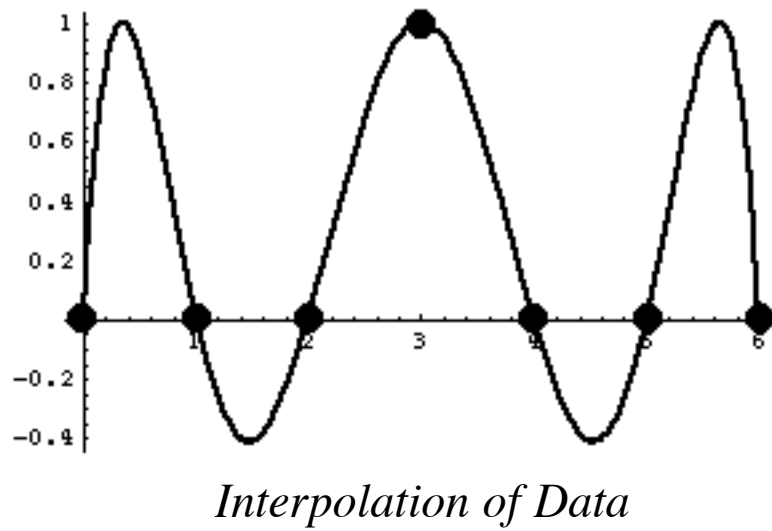
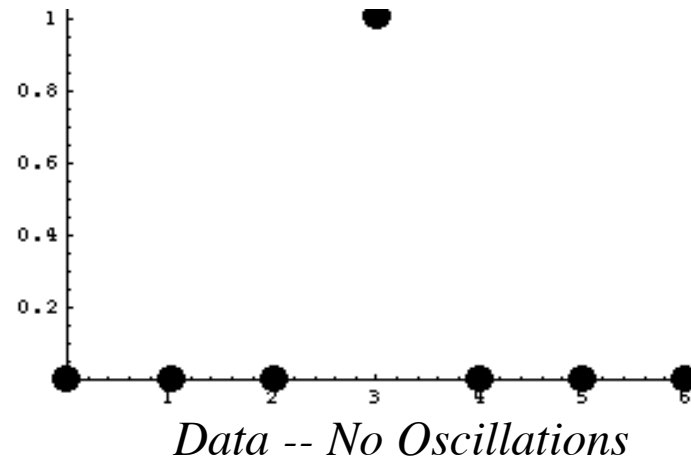
### *Properties of Lagrange Basis Functions*

- $$L_k^n(t | t_0, \dots, t_n) = \frac{\prod_{j \neq k} (t - t_j)}{\prod_{j \neq k} (t_k - t_j)} \quad (\text{Definition})$$

- $$\begin{aligned} L_k^n(t_j | t_0, \dots, t_n) &= 0 & j \neq k \\ &= 1 & j = k. \end{aligned} \quad (\text{Cardinal Conditions})$$

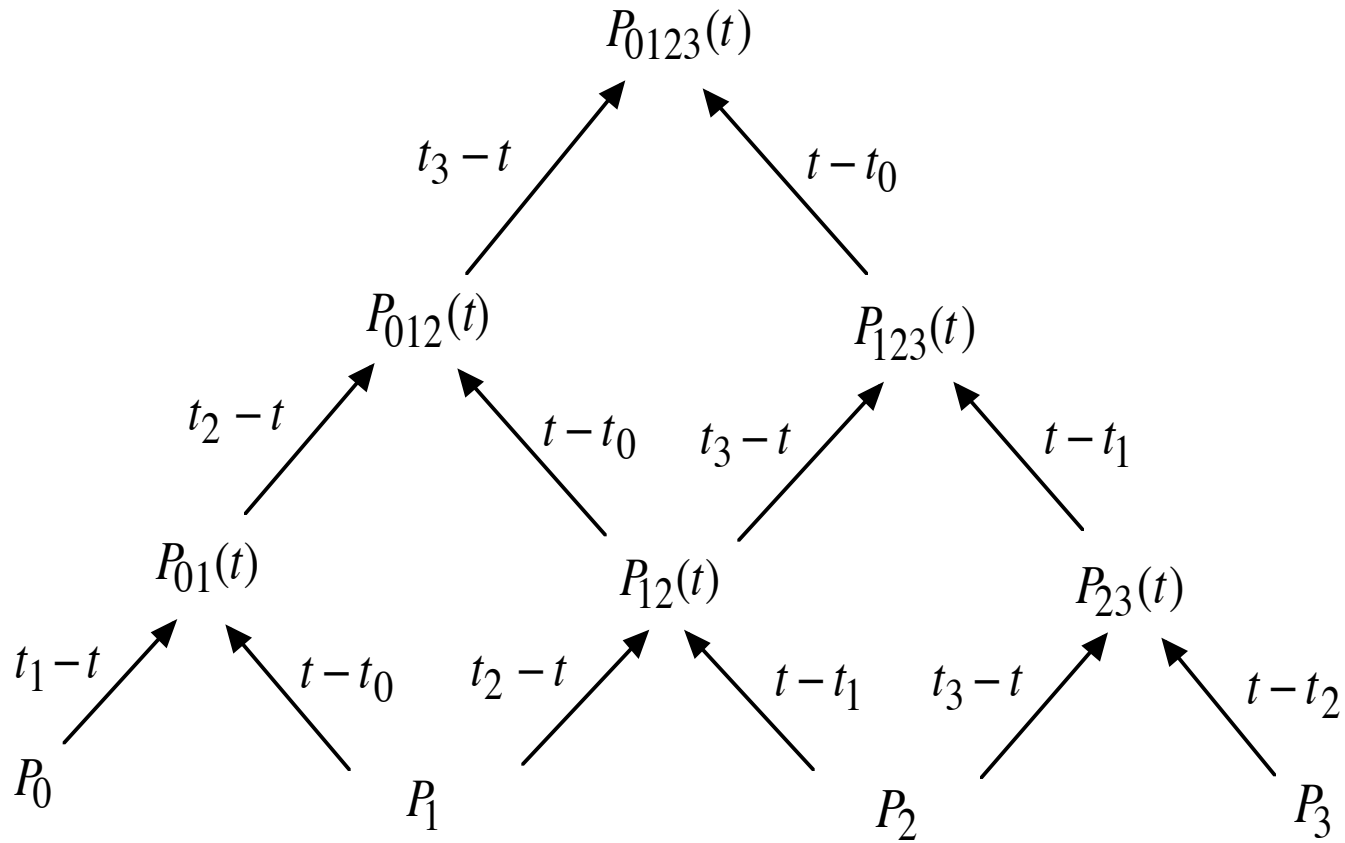
- $$\sum_{k=0}^n L_k^n(t | t_0, \dots, t_n) \equiv 1 \quad (\text{Partition of Unity})$$

## Interpolation vs. Approximation



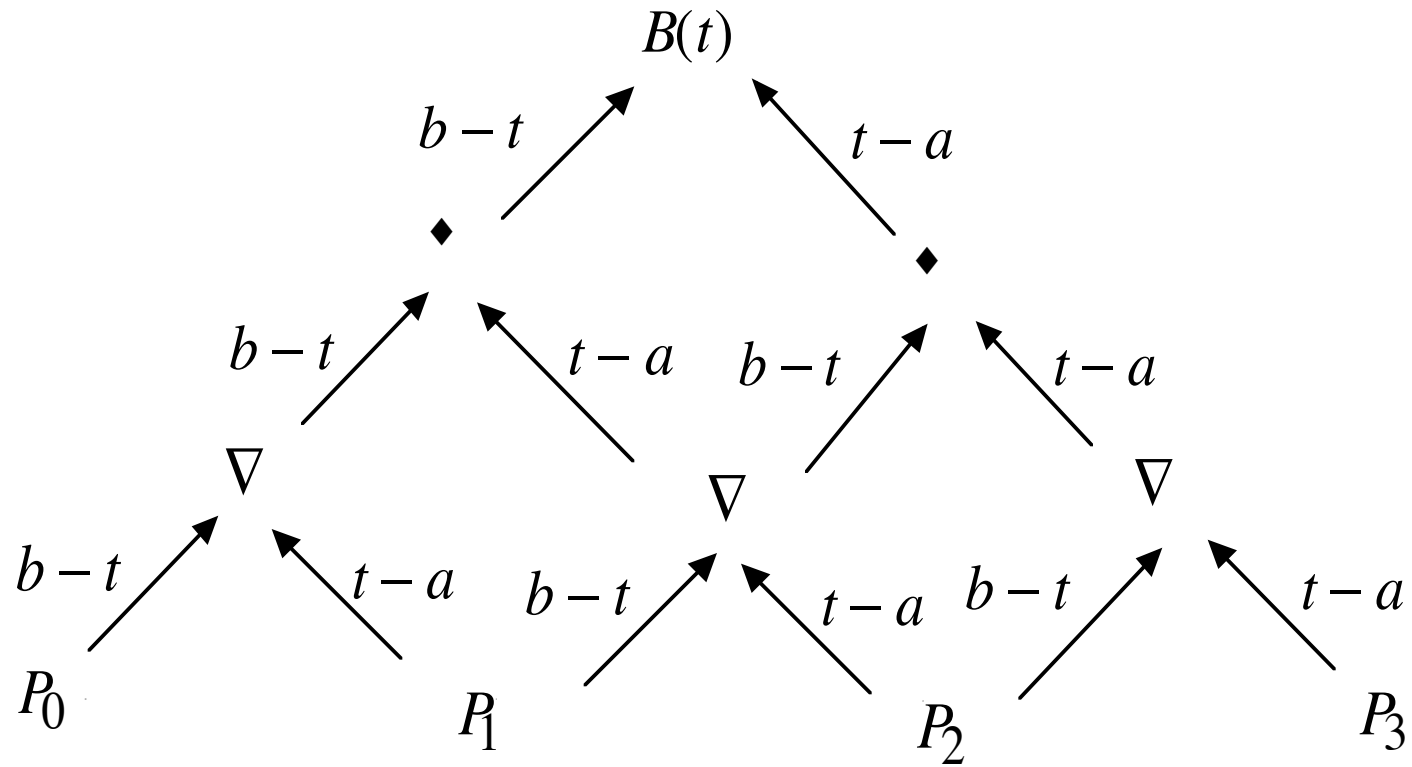
**Part IB:**  
**Bezier Approximation**

## Neville's Algorithm -- Interpolation



$$P_{0123}(t_k) = P_k$$

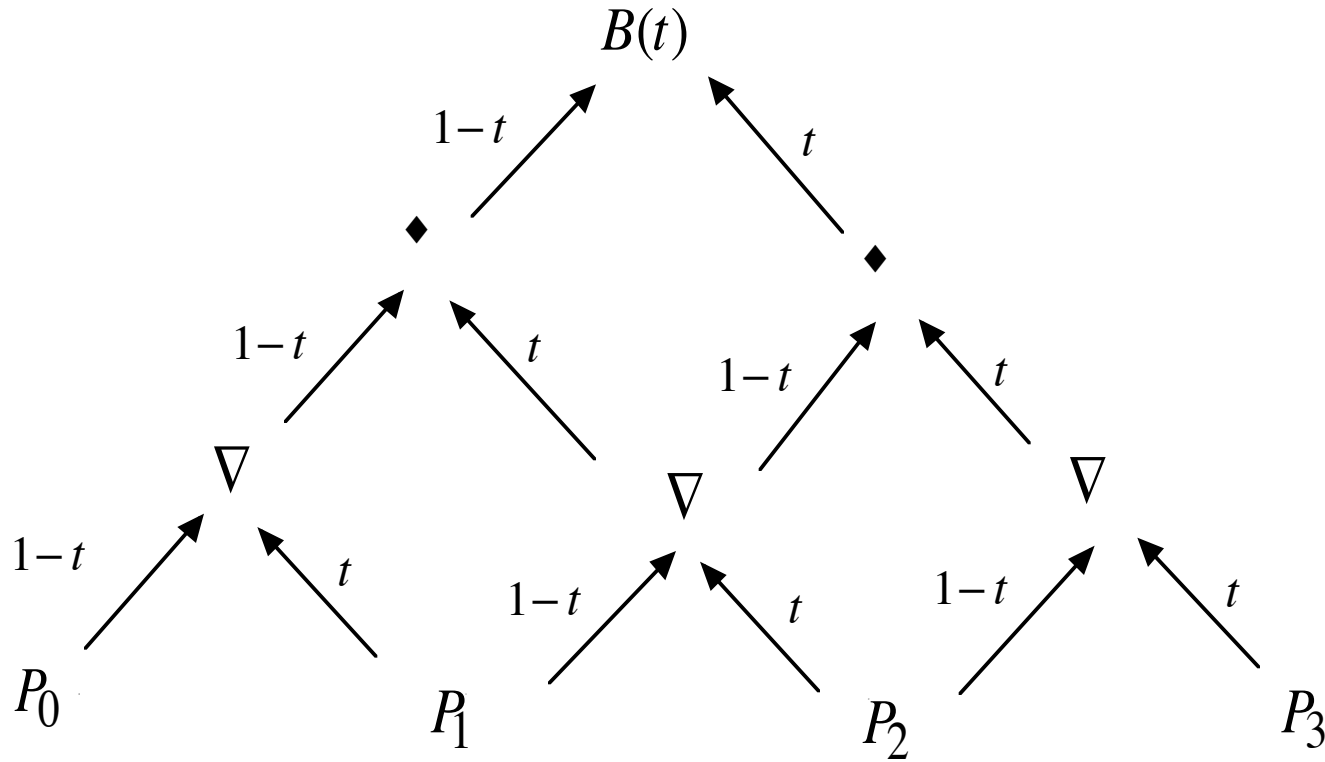
## De Casteljau's Algorithm -- Approximation



$B(t) = \text{Bezier Curve } a \leq t \leq b$

$P_0, \dots, P_n = \text{Control Points}$

## De Casteljau's Algorithm



$B(t) = \text{Bezier Curve } 0 \leq t \leq 1$

$P_0, \dots, P_n = \text{Control Points}$

## Properties and Algorithms for Bezier Curves

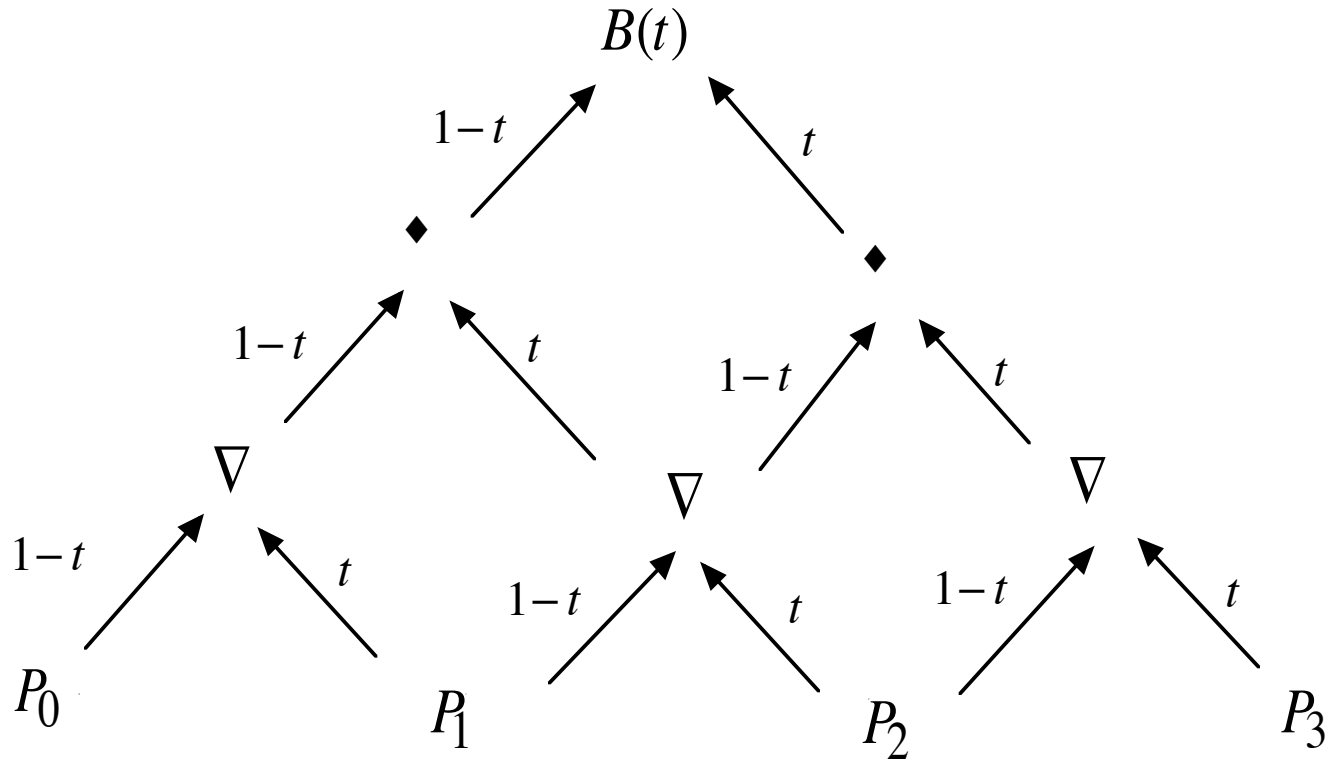
### *Properties*

1. Polynomial
2. Translation Invariant -- Affine Invariant
3. Lie in Convex Hull of their Control Points
4. Symmetry
5. Interpolation of End Points
6. Non-Degenerate
7. Variation Diminishing

### *Algorithms*

1. Evaluation
2. Differentiation
3. Subdivision
4. Blossoming
5. Degree Elevation

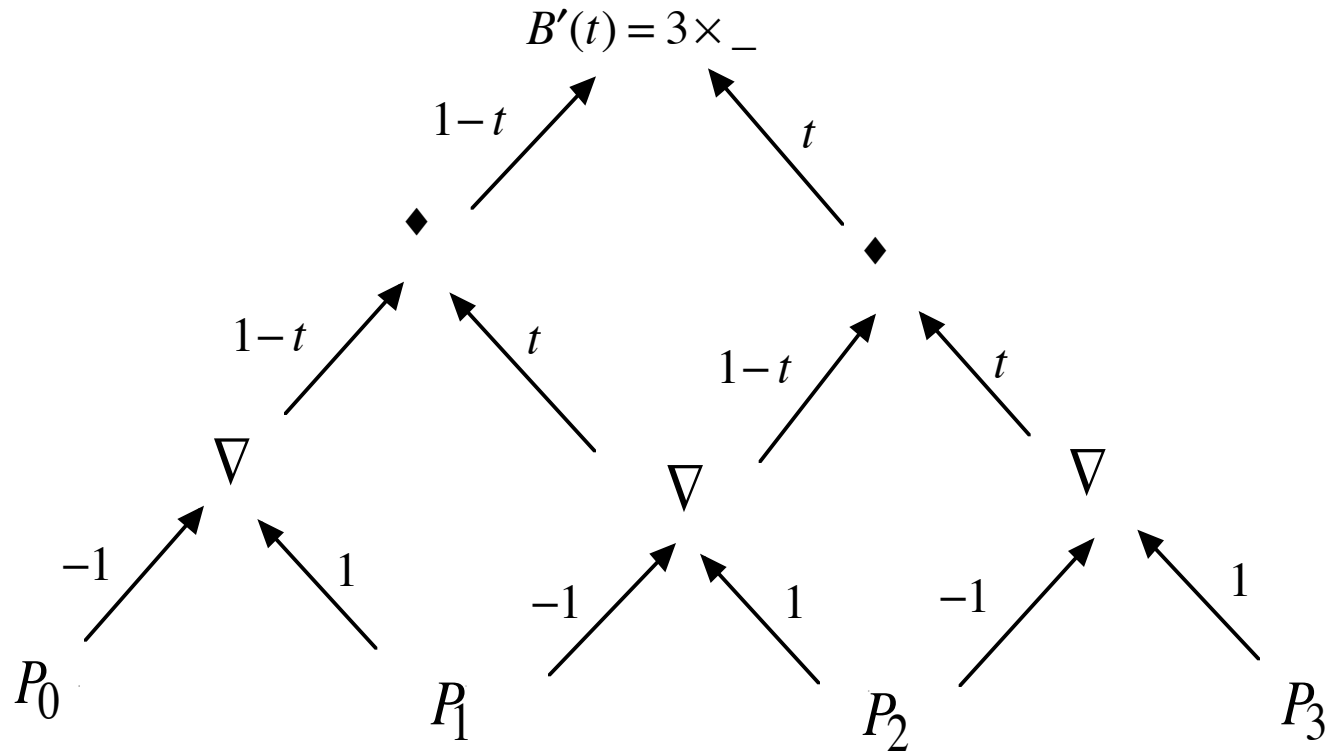
## De Casteljau's Algorithm



$B(t) = \text{Bezier Curve } 0 \leq t \leq 1$

$P_0, \dots, P_n = \text{Control Points}$

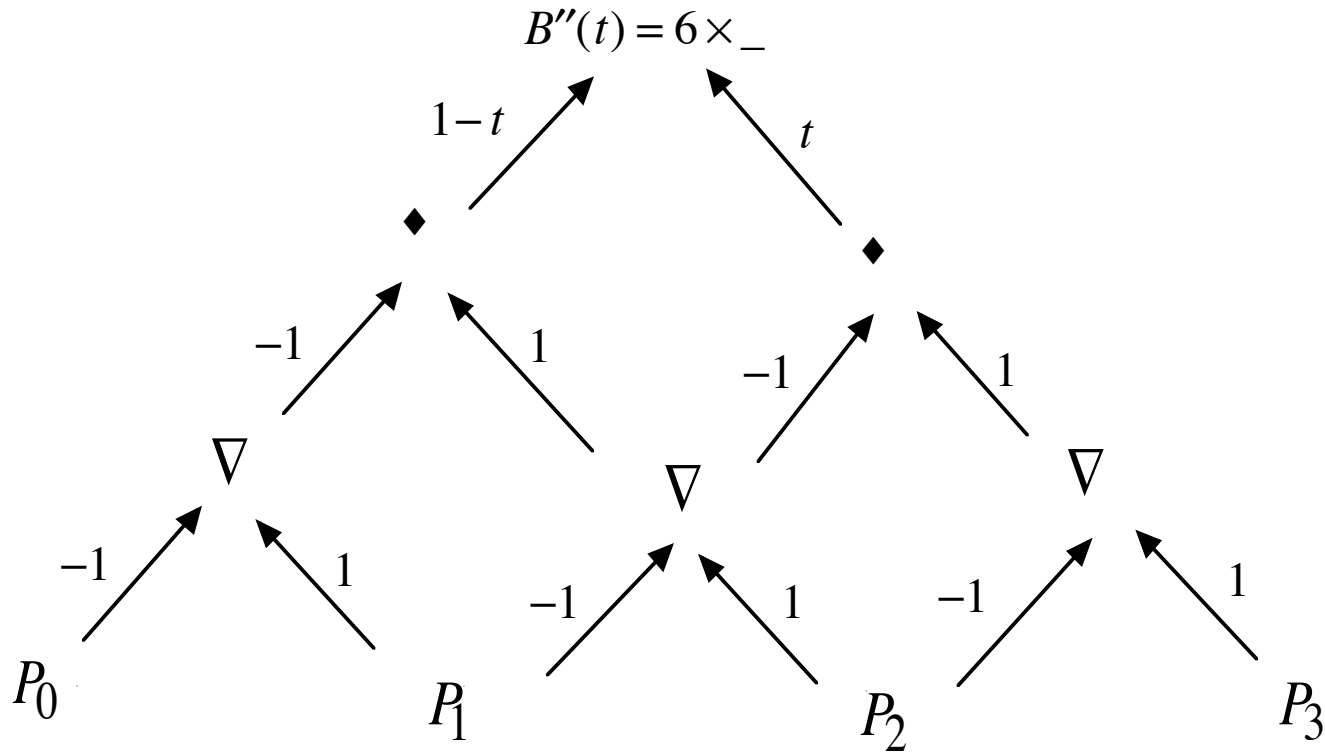
## Differentiating de Casteljau's Algorithm -- First Derivative



$B(t) = \text{Bezier Curve } 0 \leq t \leq 1$

$P_0, \dots, P_n = \text{Control Points}$

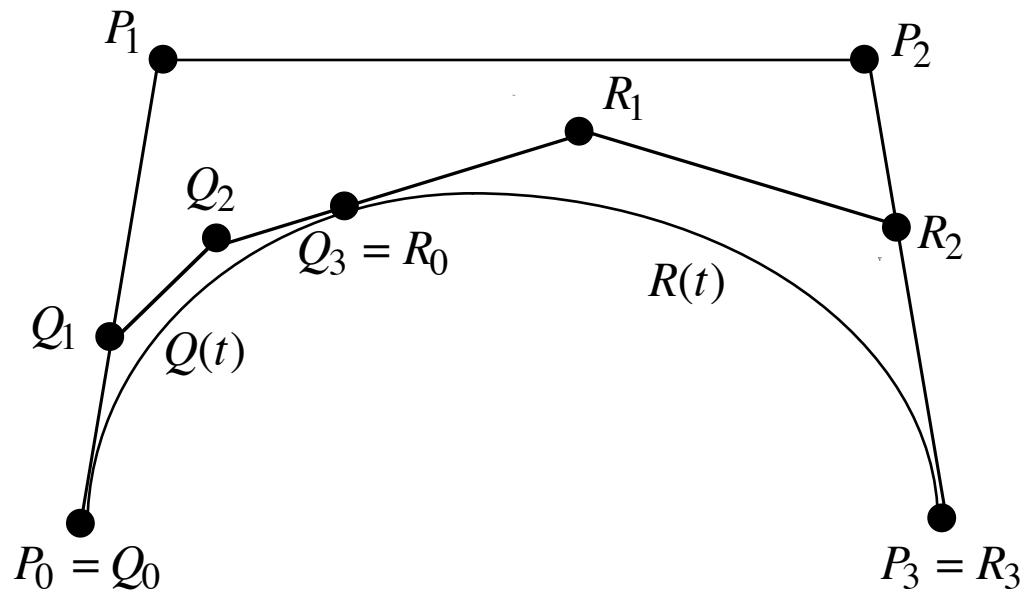
## Differentiating de Casteljau's Algorithm -- Second Derivative



$B(t) = \text{Bezier Curve } 0 \leq t \leq 1$

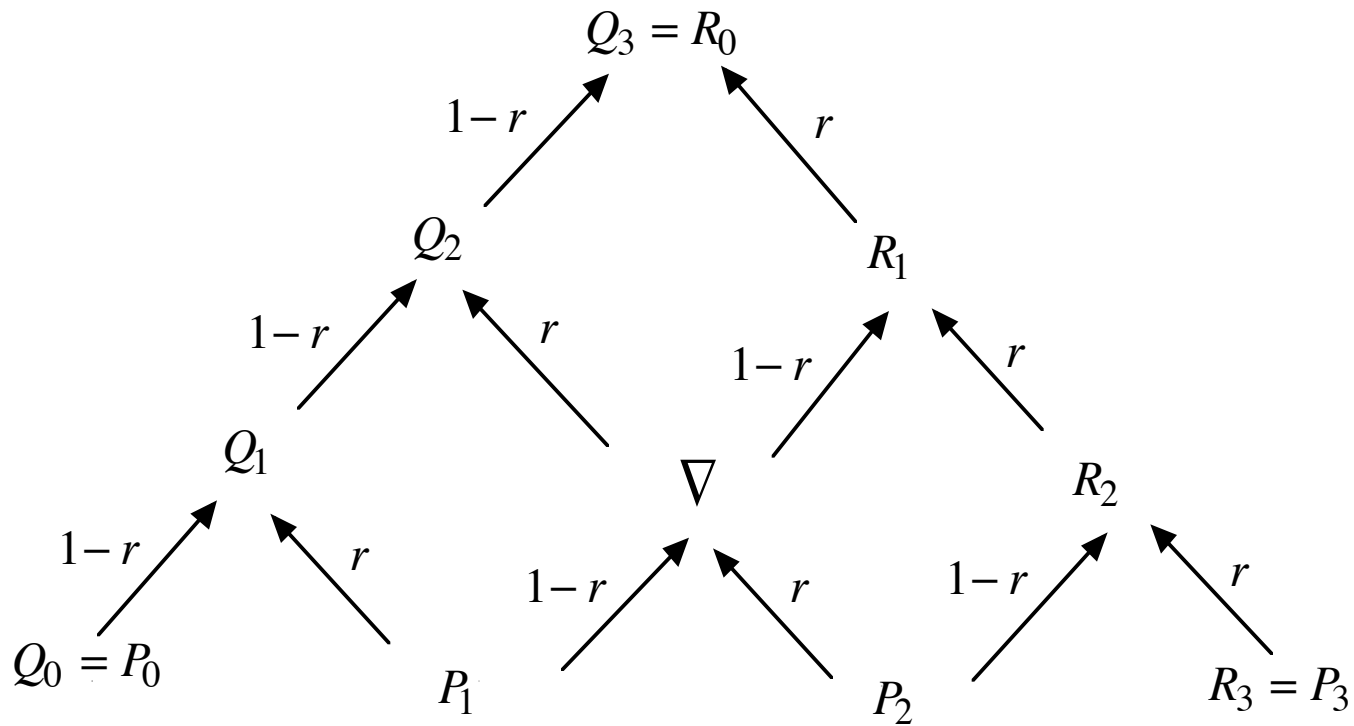
$P_0, \dots, P_n = \text{Control Points}$

## Bezier Subdivision



*Problem:* Given  $P_0, \dots, P_n$ , find  $Q_0, \dots, Q_n$  and  $R_0, \dots, R_n$ .

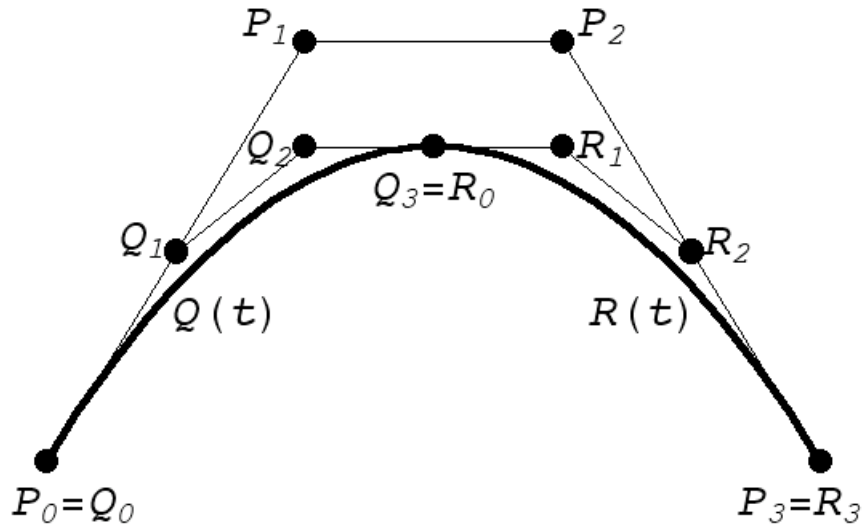
## De Casteljau's Subdivision Algorithm



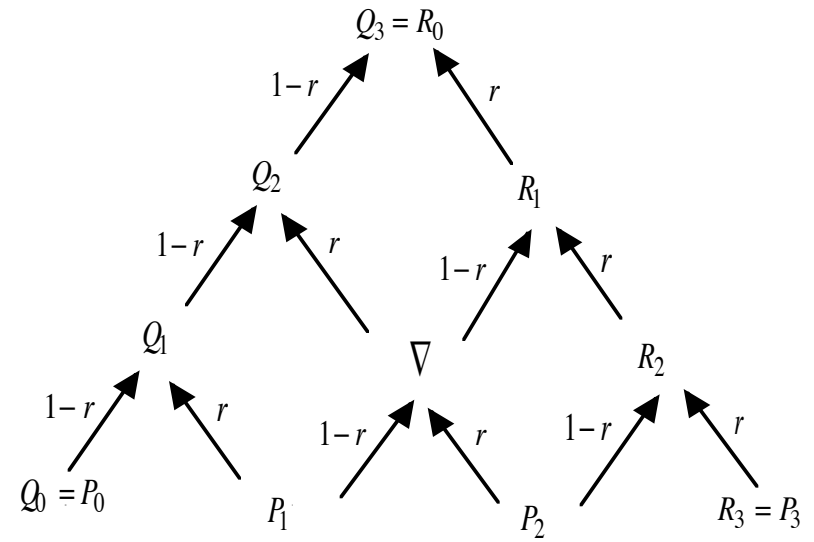
$B(t) = \text{Bezier Curve } 0 \leq t \leq 1$

$P_0, \dots, P_n = \text{Original Control Points}$

## De Casteljau Subdivision



*Geometry*



*Algorithm*

## Algorithms for Bezier Curves

### *Rendering Algorithm*

- If the Bezier curve can be approximated to within tolerance by the straight line joining its first and last control points,  
    then draw either this line segment or the control polygon.
- Otherwise *subdivide* the curve (at  $r = 1/2$ ) and render the segments recursively.

### *Intersection Algorithm (Root Finding)*

- If the convex hulls of the control points of two Bezier curves fail to intersect,  
    then the curves themselves do not intersect.
- Otherwise if each Bezier curve can be approximated by the straight line joining its first and last control points,  
    then intersect these line segments.
- Otherwise *subdivide* the two curves and intersect the segments recursively.

## Bernstein Approximation

### *Explicit Formula*

- $$B(t) = \sum_{k=0}^n P_k B_k^n(t)$$

### *Properties of Bernstein Basis Functions*

- $$B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

*(Definition)*

- $$B_k^n(t) = (1-t)B_k^{n-1}(t) + tB_{k-1}^{n-1}(t)$$

*(Recursion)*

- $$\frac{dB_k^n}{dt} = n \left\{ B_{k-1}^{n-1}(t) - B_k^{n-1}(t) \right\}$$

*(Differentiation)*

- $$B_i^n(rt) = \sum_{k=i}^n B_i^k(r) B_k^n(t)$$

*(Subdivision)*

## Bernstein Approximation -- Tensor Product Surfaces

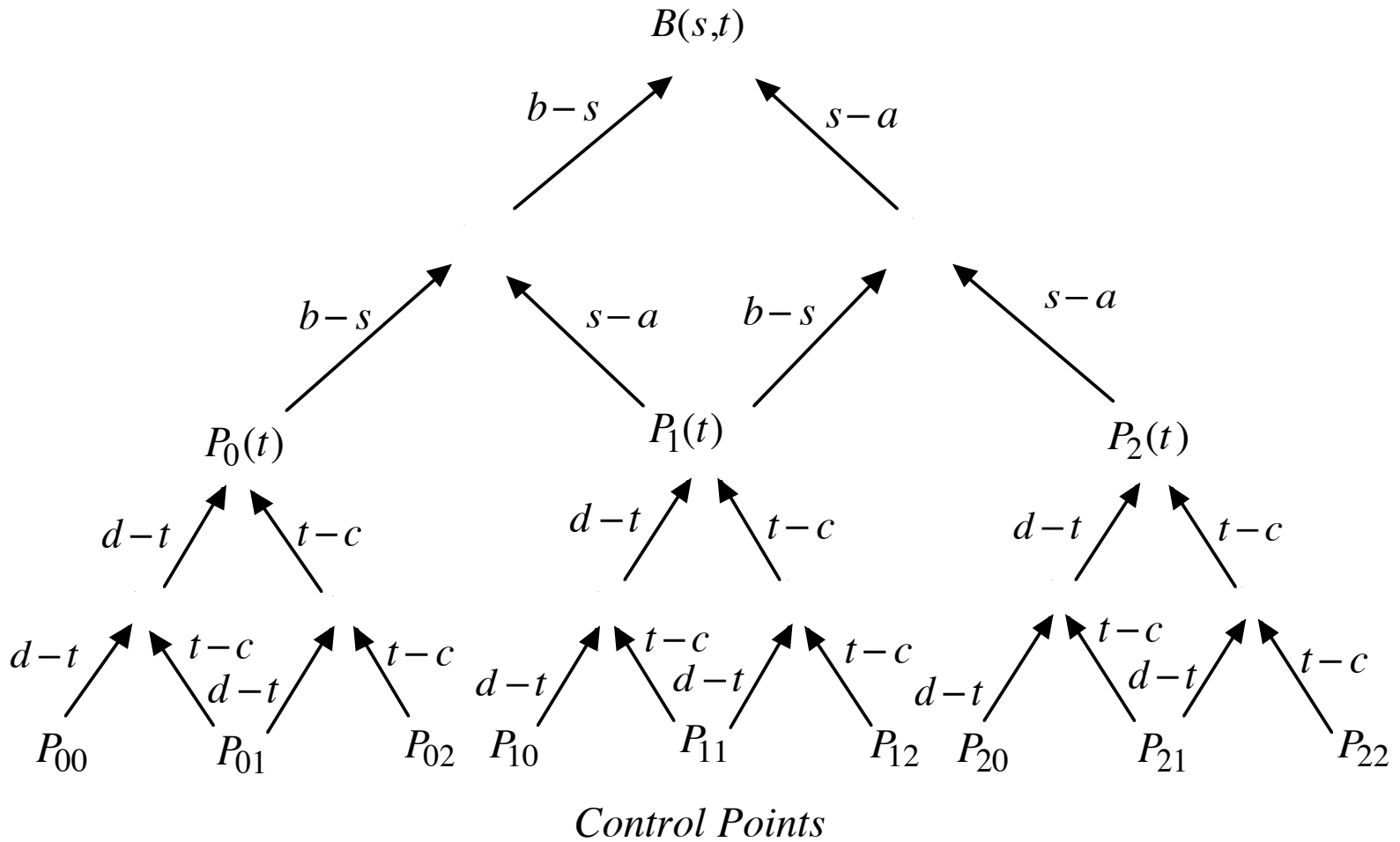
### *Explicit Formulas*

- $$B(s,t) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} B_i^m(s) B_j^n(t)$$

-- 
$$B(s,t) = \sum_{i=0}^m B_i^m(s) \sum_{j=0}^n P_{ij} B_j^n(t) \quad \Rightarrow \quad B(s,t) = \sum_{i=0}^m B_i^m(s) P_i(t)$$

-- 
$$B(s,t) = \sum_{j=0}^n B_j^n(t) \sum_{i=0}^m P_{ij} B_i^m(s) \quad \Rightarrow \quad B(s,t) = \sum_{j=0}^n B_j^n(t) P_j(s)$$

## De Casteljau Algorithm -- Tensor Product Bezier Patches



## Properties and Algorithms for Bezier Surfaces

### *Properties*

1. Polynomial
2. Translation Invariant -- Affine Invariant
3. Lie in Convex Hull of their Control Points
4. Symmetry
5. Interpolation of Boundary Curves
6. Non-Degenerate

### *Algorithms*

1. Evaluation
2. Differentiation
3. Subdivision
4. Blossoming
5. Degree Elevation

## Weaknesses of Bernstein/Bezier Approximation

- High Degree Polynomials
- No Local Control

**Part IC:**  
**B-Spline Approximation**

## B-Spline Curves and Surfaces

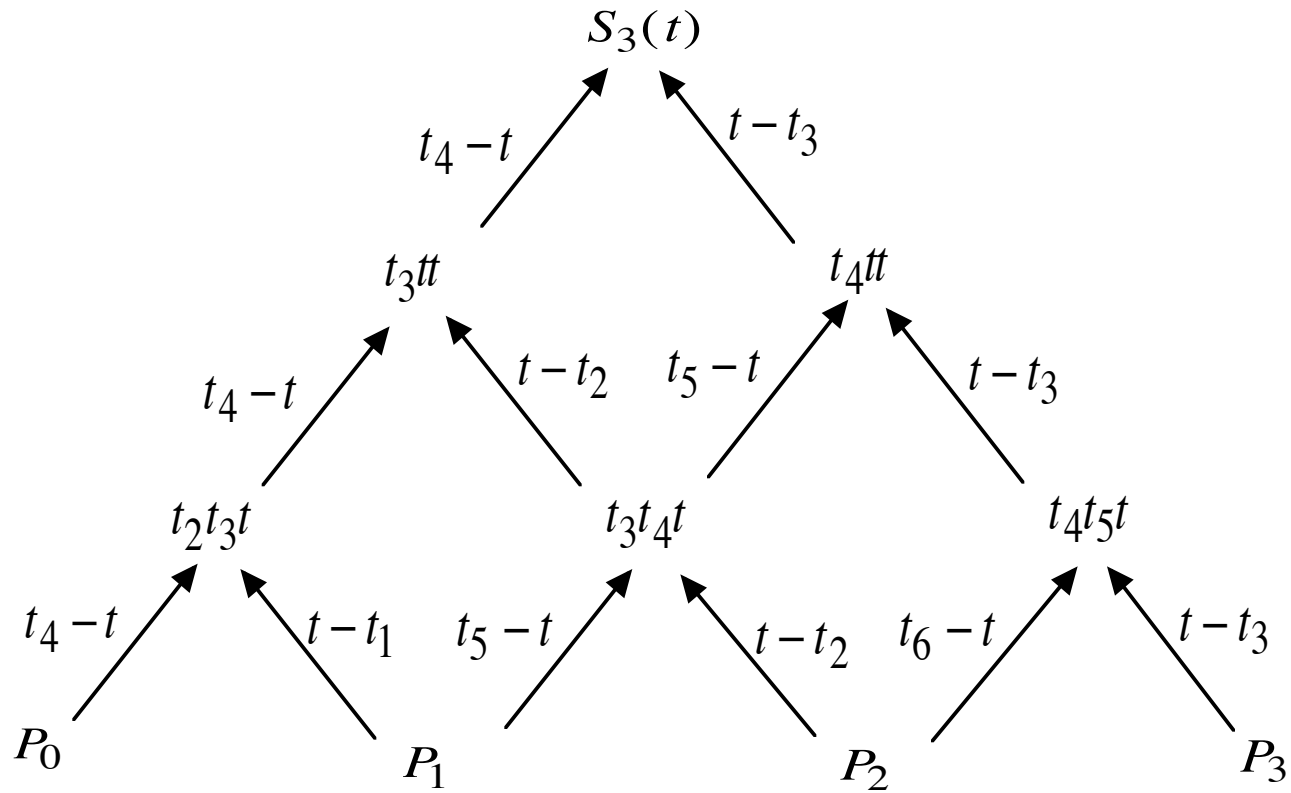
### *Properties*

1. Piecewise Polynomial -- Low Degree
2. Smooth --  $n - \mu$  continuous derivatives at knots of multiplicity  $\mu$
3. Translation Invariant -- Affine Invariant
4. Lie in Convex Hull of their Control Points
5. Interpolation at Points Corresponding to Multiple Knots
6. Non-Degenerate
7. Variation Diminishing (Curves Only)
8. Local Control

### *Algorithms*

1. Evaluation -- De Boor Algorithm
2. Differentiation
3. Knot Insertion
4. Blossoming

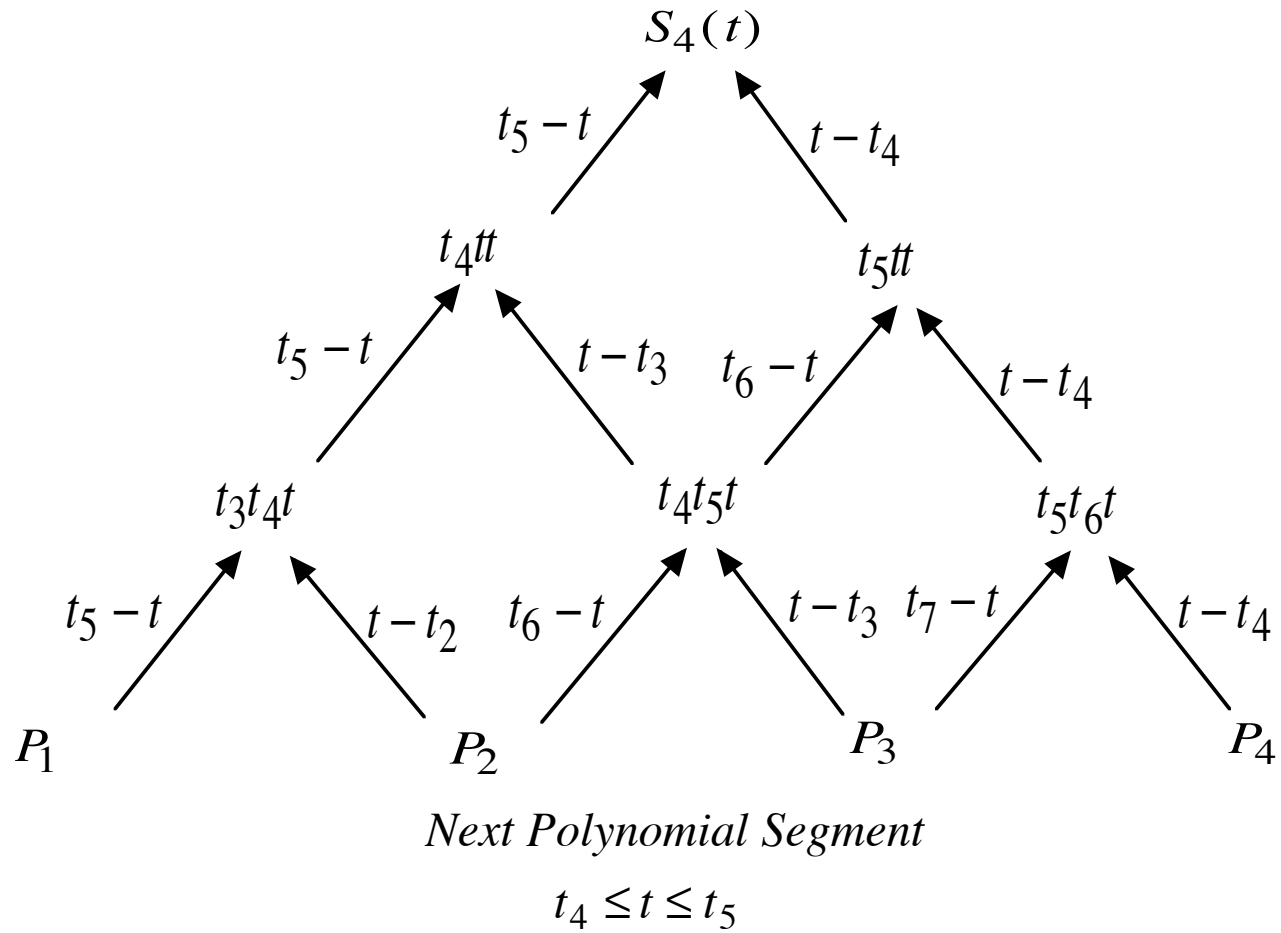
## De Boor Algorithm



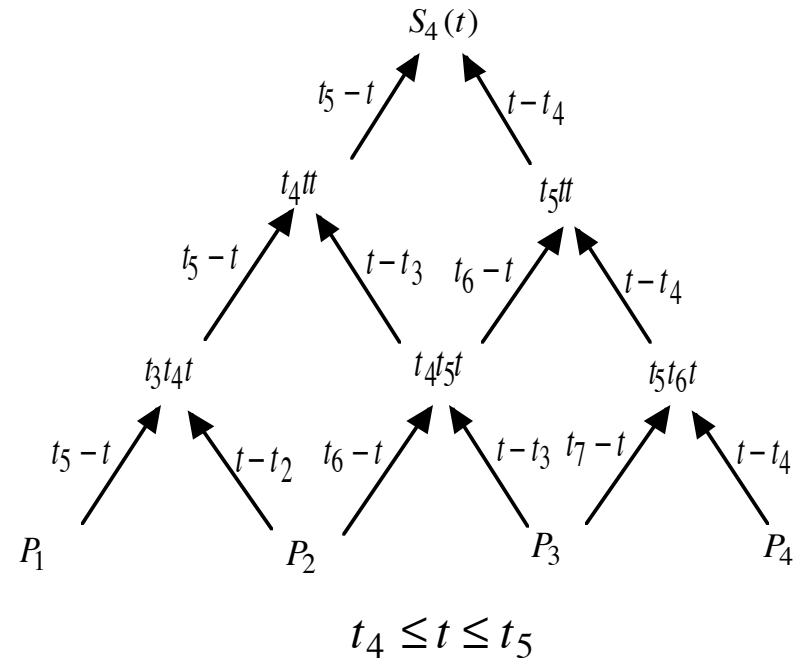
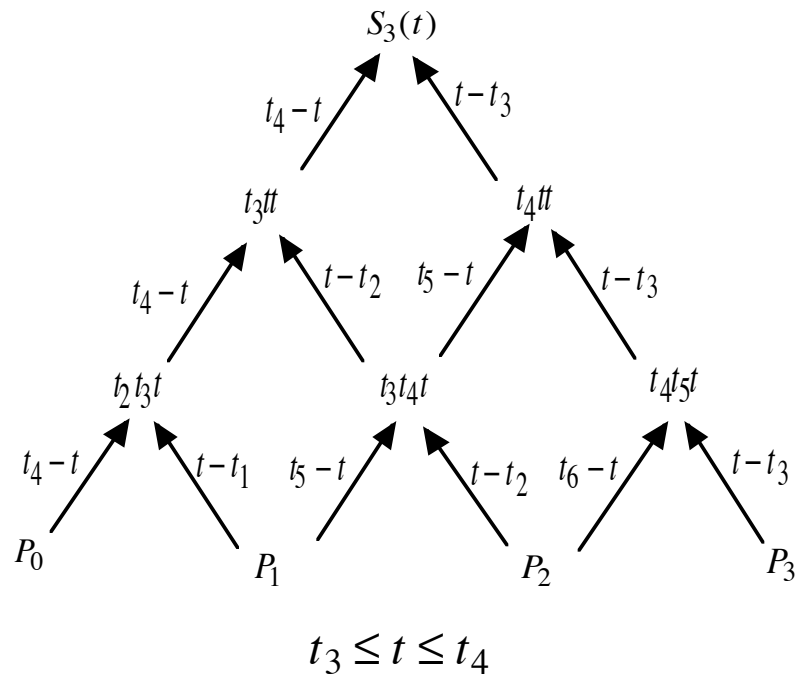
*One Polynomial Segment*

$$t_3 \leq t \leq t_4$$

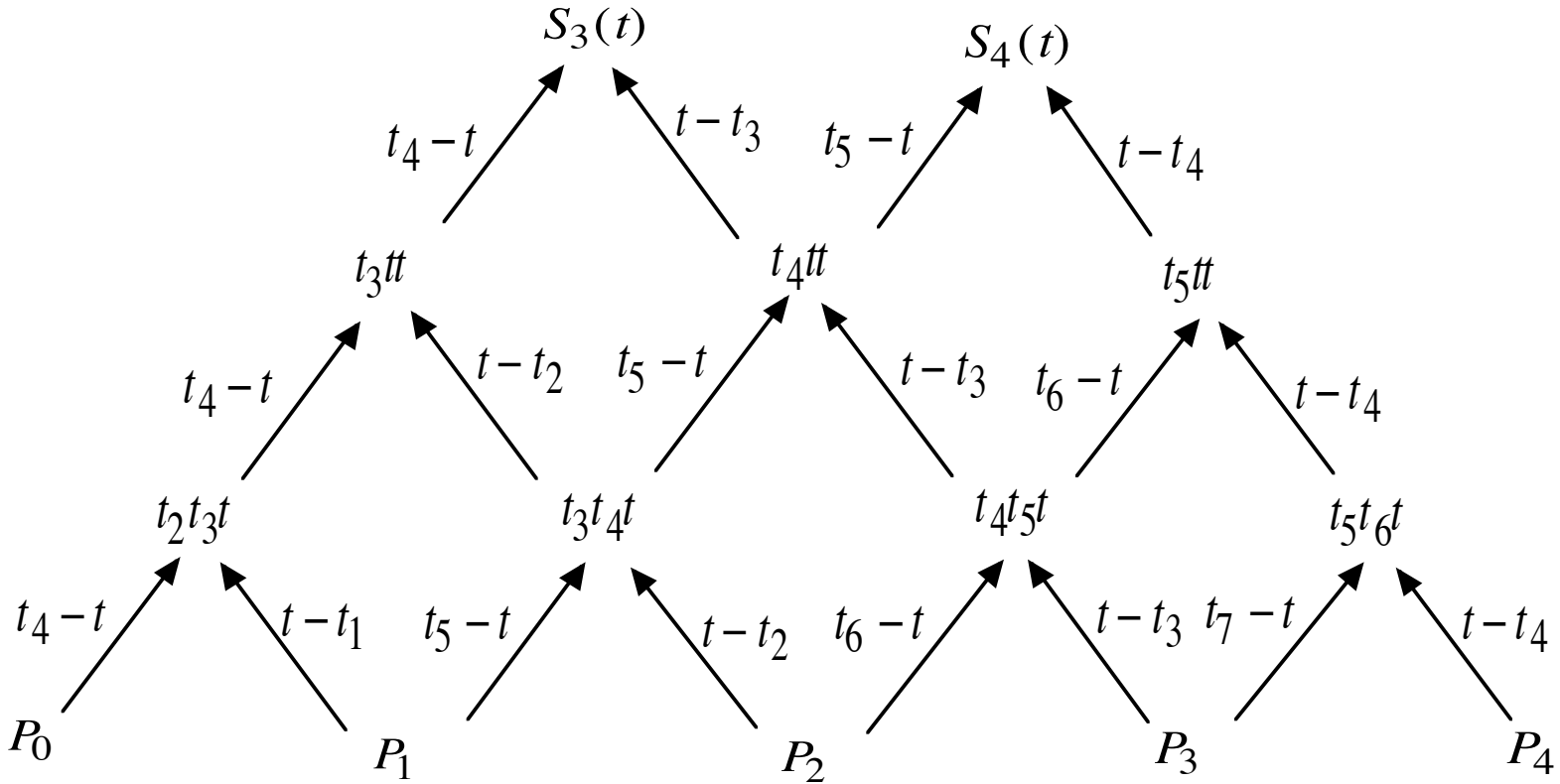
## De Boor Algorithm



## De Boor Algorithm -- Two Segments



**De Boor Algorithm**



*Two Polynomial Segments*

$$t_3 \leq t \leq t_5$$

## Advantages of the de Boor Algorithm

### 1. *Numerically Stable*

- Uses the Given Data Directly
- No Need to Represent the Polynomial Segments in the Basis  $1, t, t^2, \dots$

### 2. *Fast*

- Dynamic Programming
- $O(n^2)$  vs.  $O(2^n)$

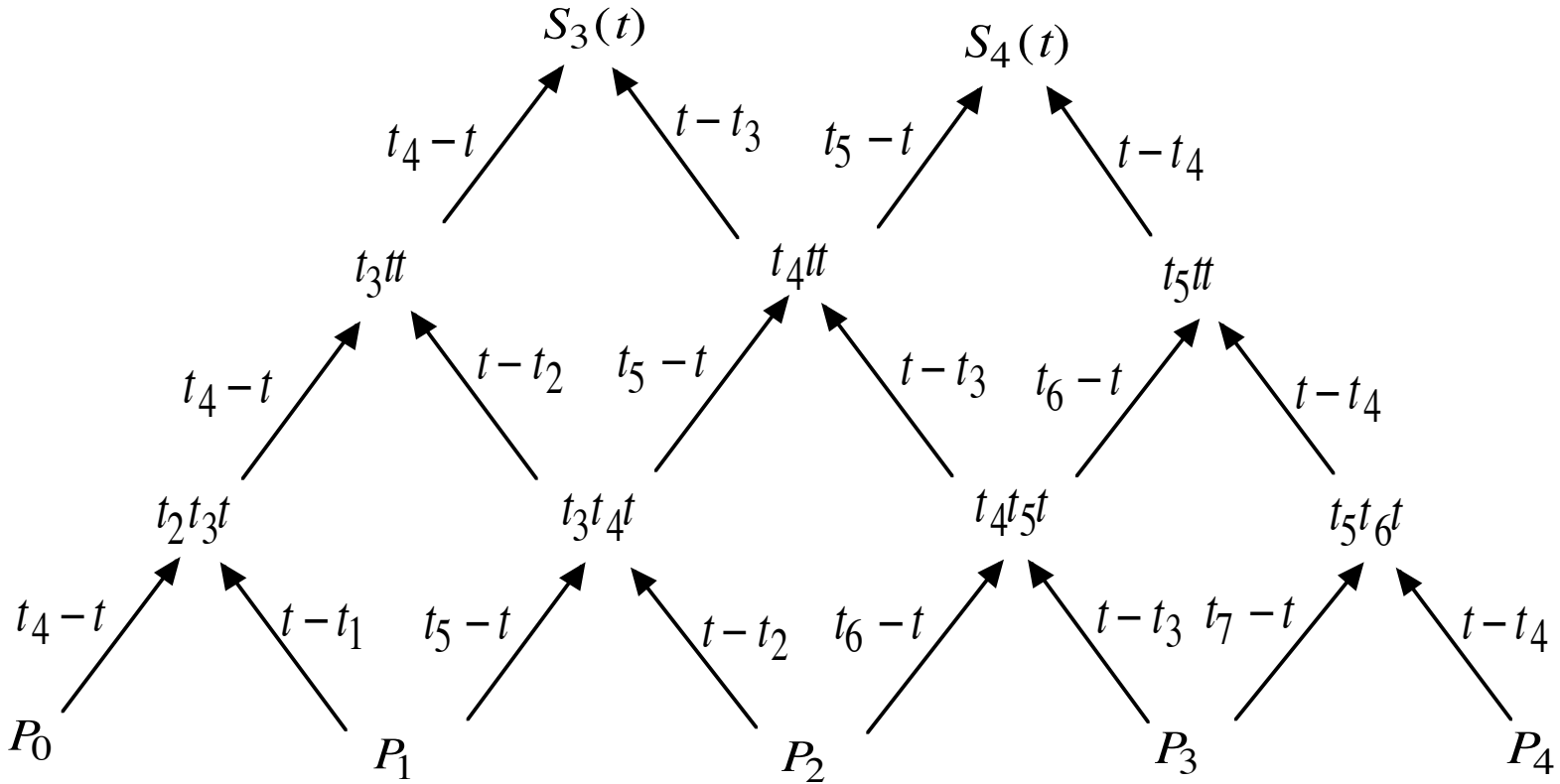
### 3. *Simple Structure*

- In-Out Property
- Blossoming Notation

### 4. *Blossoming*

- Easy Proofs

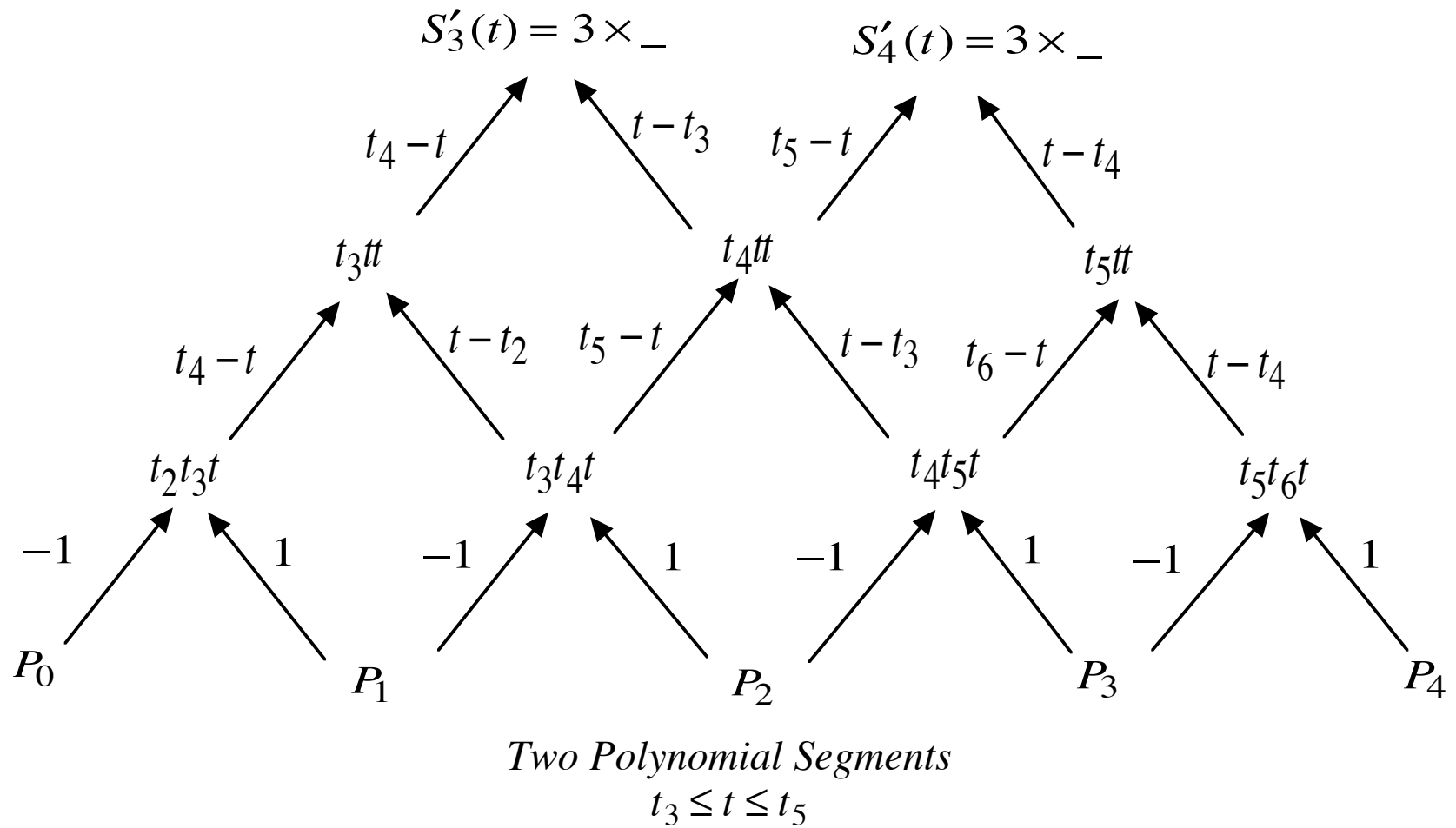
**De Boor Algorithm**



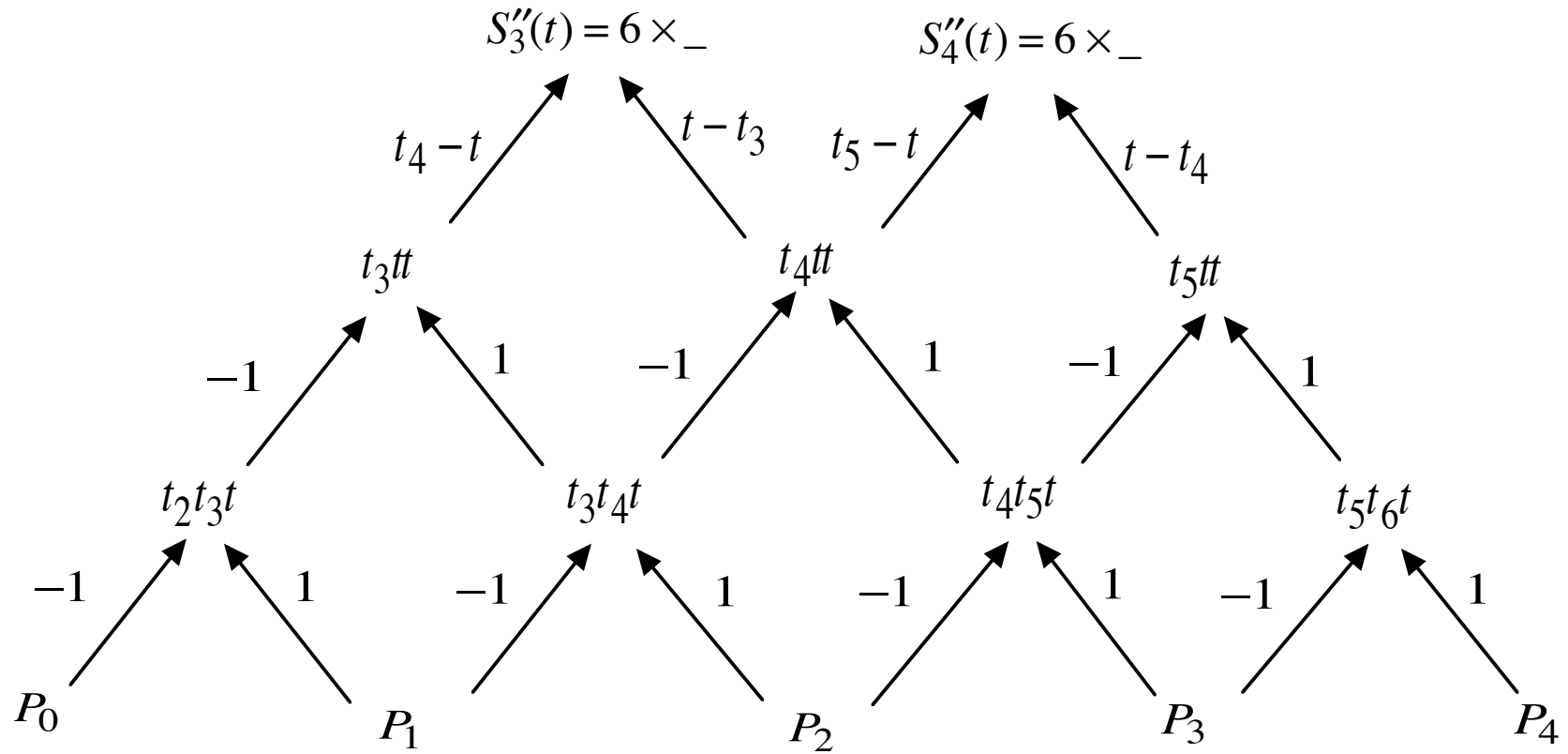
*Two Polynomial Segments*

$$t_3 \leq t \leq t_5$$

## Differentiating the de Boor Algorithm -- First Derivative



## Differentiating the de Boor Algorithm -- Second Derivative



*Two Polynomial Segments*  
 $t_3 \leq t \leq t_5$

## Knot Insertion Algorithm

### *Input*

- $T = \{t_1, \dots, t_{\nu+n}\}$  -- knot sequence
- $P = \{P_0, \dots, P_\nu\}$  -- control points
- $\Gamma = \{\tau_1, \dots, \tau_{\mu+n}\}$  -- new knot sequence --  $\Gamma \supset T$

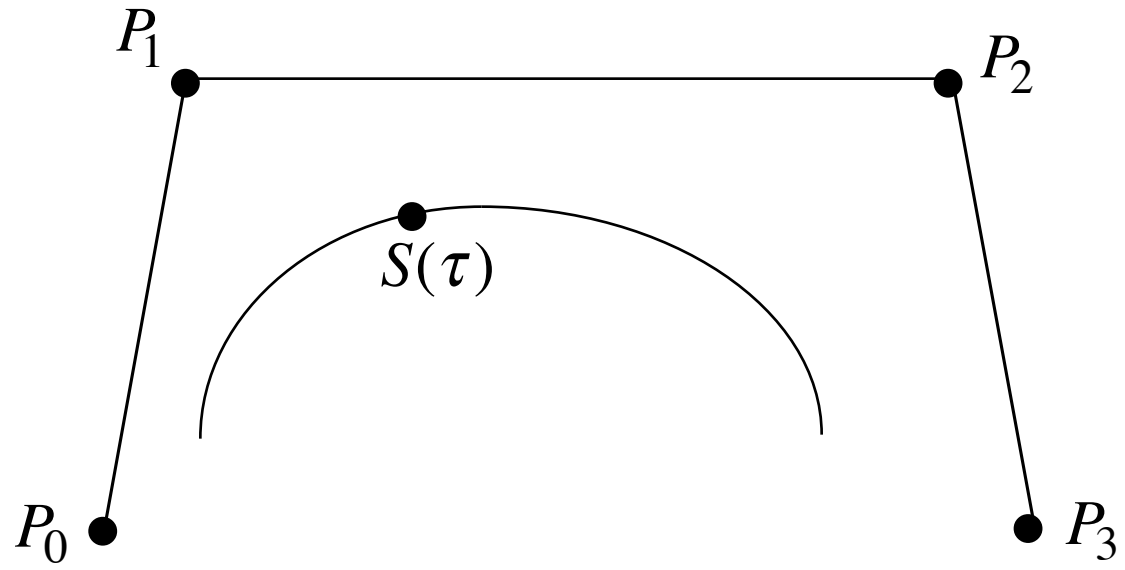
### *Output*

- $Q = \{Q_0, \dots, Q_\mu\}$  -- new control points

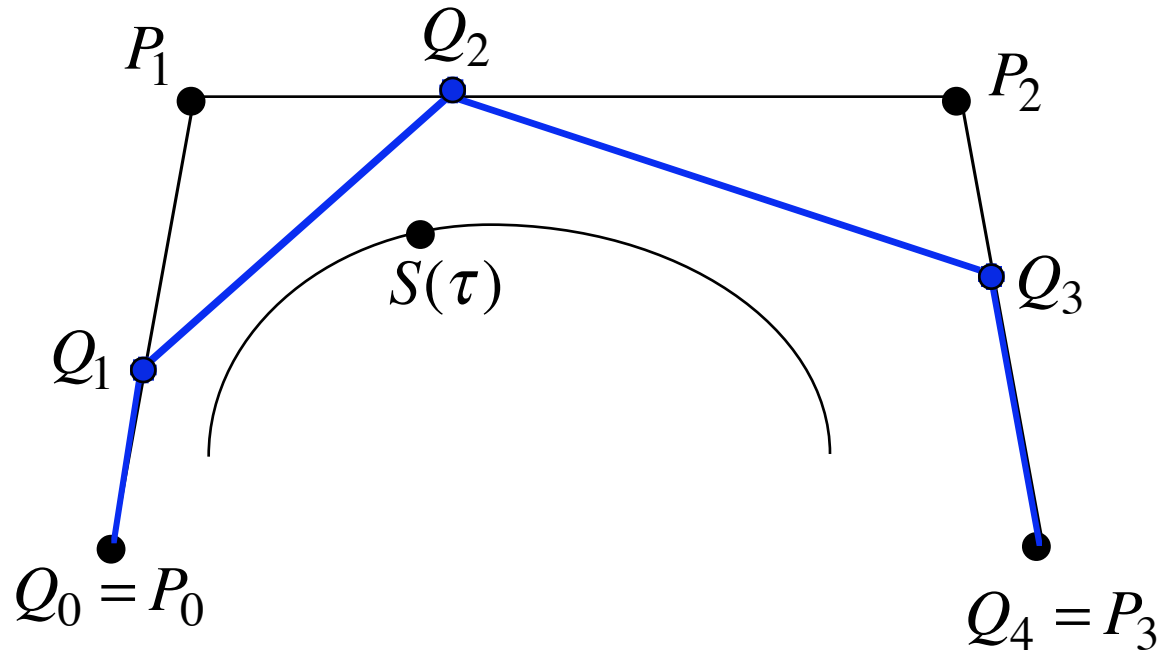
### *Constraint*

- $S(t | T, P) = S(t | \Gamma, Q)$

**B-Spline Curve**



# Knot Insertion



## Knot Insertion Algorithms

### *Local Algorithms*

- Boehm's Algorithm
- Oslo Algorithm

$$\text{-- } T = \{t_1, \dots, t_{v+n}\}$$

$$\text{-- } \Gamma = \{t_1, \dots, t_k, u_{k,1}, \dots, u_{k,d_k}, t_{k+1}, \dots, t_{v+n}\}$$

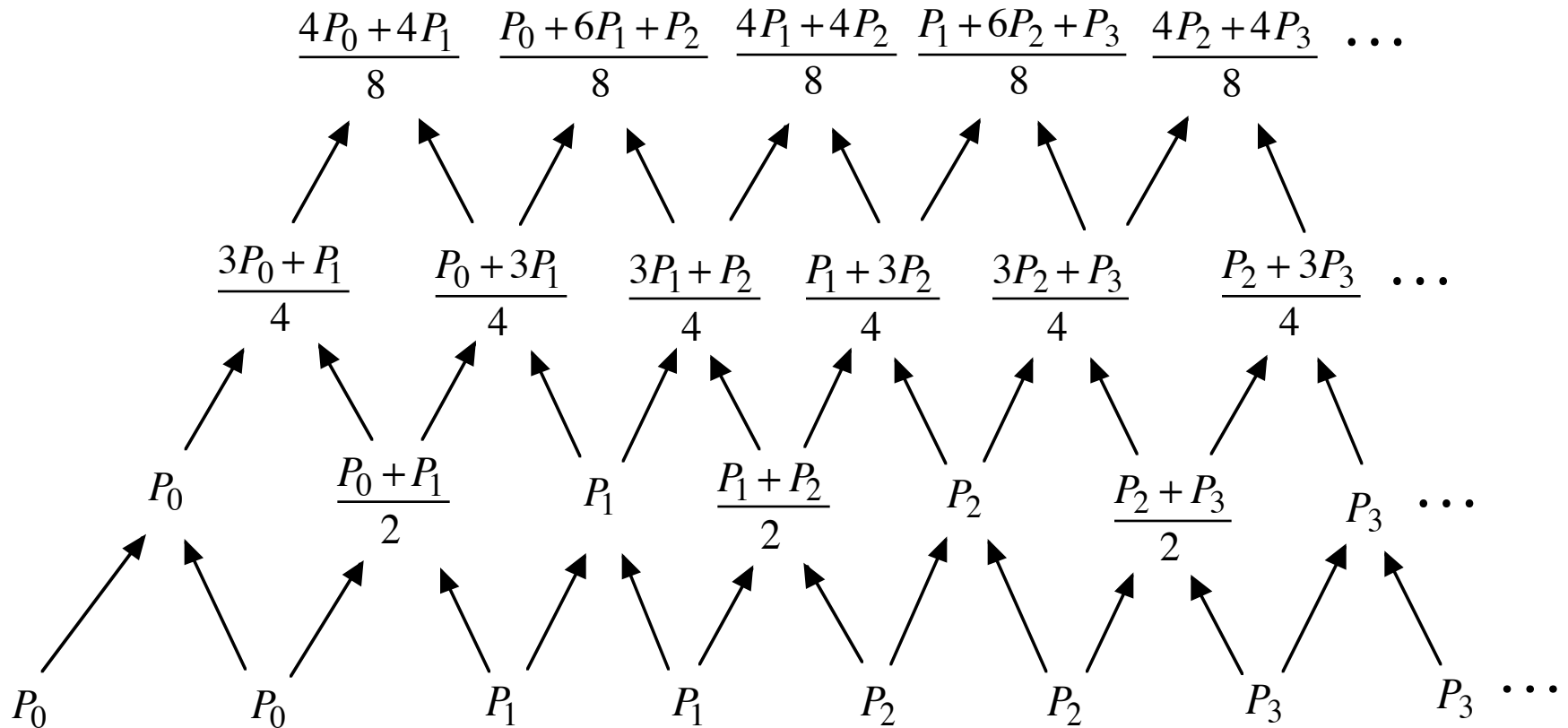
### *Global Algorithms*

- Lane-Riesenfeld Algorithm (Uniform Knots)
- Schaefer's Algorithm (Arbitrary Knots)

$$\text{-- } T = \{t_1, \dots, t_{v+n}\}$$

$$\text{-- } \Gamma = \{t_1, u_{1,1}, \dots, u_{1,d_1}, t_2, \dots, t_{v+n-1}, u_{v+n-1,1}, \dots, u_{v+n-1,d_{v+n-1}}, t_{v+n}\}$$

## Lane-Riesenfeld Knot Insertion Algorithm



*Original Control Points*

*Split and Average*

## Algorithms for B-Spline Curves

### *Rendering Algorithm -- B-Spline Curves*

- Apply a knot insertion to generate a control polygon that approximates the B-spline curve to within the given tolerance.
- Render the control polygon.

### *Intersection Algorithm -- B-Spline Curves*

- If the convex hulls of the control points of two B-spline curves fail to intersect, then the curves themselves do not intersect.
- Otherwise if each B-spline curve can be approximated by the straight line segments joining the first and last points of each polynomial segment, then intersect these line segments.
- Otherwise apply knot insertion to insert one new knot midway between each consecutive pair of the original knots and intersect the B-spline curves recursively.

## B-Spline Approximation

### *Explicit Formula*

- $$S(t) = \sum_{k=0}^v P_k N_{k,n}(t)$$

### *Properties of B-Spline Basis Functions*

- $$N_{k,0}(t) = 1 \quad t_k \leq t < t_{k+1}$$

$$N_{k,n}(t) = \frac{t - t_k}{t_{k+n} - t_k} N_{k,n-1}(t) + \frac{t_{k+n+1} - t}{t_{k+n+1} - t_{k+1}} N_{k+1,n-1}(t) \quad (\text{Recursive Definition})$$

- $$\frac{dN_{k,n}(t)}{dt} = n \left( \frac{N_{k,n-1}(t)}{t_{k+n} - t_k} - \frac{N_{k+1,n-1}(t)}{t_{k+n+1} - t_{k+1}} \right) \quad (\text{Differentiation})$$

- $$S(t) = \sum_i s(t_{i+1}, \dots, t_{i+n}) N_{in}(t) \quad (\text{Blossoming})$$

## Tensor Product B-Spline Surfaces

### *Explicit Formulas*

- $$S(s,t) = \sum_{i=0}^{\mu} \sum_{j=0}^{\nu} P_{ij} N_{i,m}(s) N_{j,n}(t)$$

-- 
$$S(s,t) = \sum_{i=0}^{\mu} N_{i,m}(s) \sum_{j=0}^{\nu} P_{ij} N_{j,n}(t) \quad \Rightarrow \quad S(s,t) = \sum_{i=0}^{\mu} N_{i,m}(s) P_i(t)$$

-- 
$$S(s,t) = \sum_{j=0}^{\nu} N_{j,n}(t) \sum_{i=0}^{\mu} P_{ij} N_{i,m}(s) \quad \Rightarrow \quad S(s,t) = \sum_{j=0}^{\nu} N_{j,n}(t) P_j(s)$$

## Non-Uniform Rational B-Splines -- NURBS

### *Problem -- Rational Functions*

- B-splines cannot represent exactly many common rational curves and surfaces

-- Circle:  $x(t) = \frac{1-t^2}{1+t^2}$        $y(t) = \frac{2t}{1+t^2}$

-- Sphere:  $x(s,t) = \frac{1-s^2}{1+s^2} \frac{1-t^2}{1+t^2}$        $y(s,t) = \frac{1-s^2}{1+s^2} \frac{2t}{1+t^2}$        $z(s,t) = \frac{2s}{1+s^2}$

### *Solution -- Rational B-Splines*

- Represent the denominator as an additional coordinate

--  $R(t) = \left( \sum_{k=0}^v m_k P_k N_{k,n}(t), \sum_{k=0}^v m_k N_{k,n}(t) \right) = \sum_{k=0}^v (m_k P_k, m_k) N_{k,n}(t)$

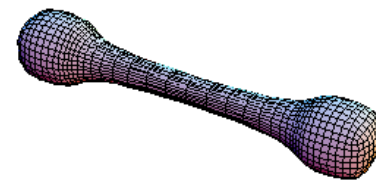
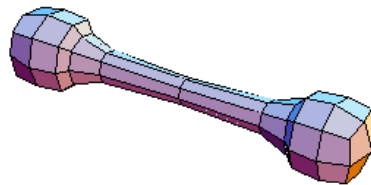
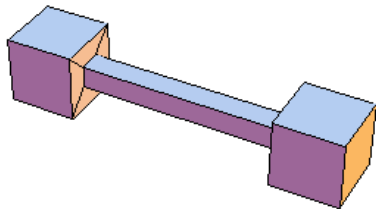
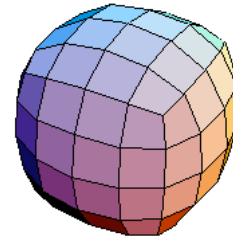
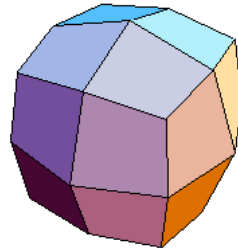
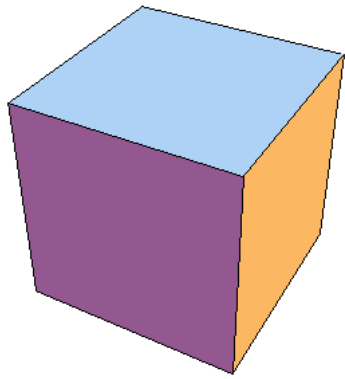
- Use the same algorithms with one additional coordinate
- Divide by the last coordinate

## Weaknesses of B-Spline Approximation for Surfaces

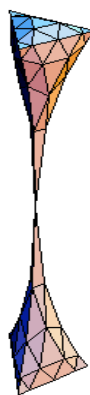
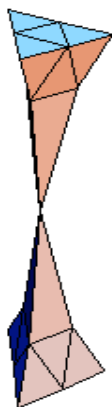
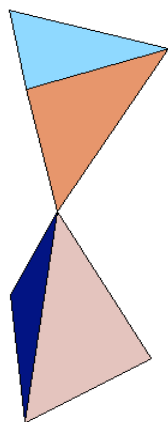
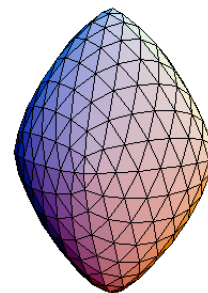
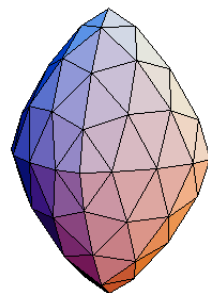
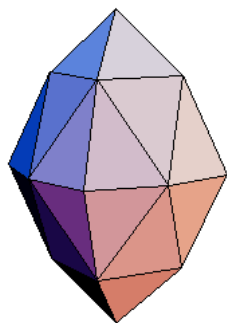
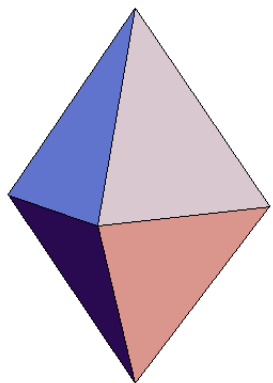
- Only for Regular Quadrilateral Meshes
  - Limited Topology: Planes, Cylinders, Tori
- No Irregular Quadrilateral Meshes
  - General Topology: Spheres, Higher Order Genuses
- No Triangular Meshes

**Part ID:**  
**Subdivision Surfaces**

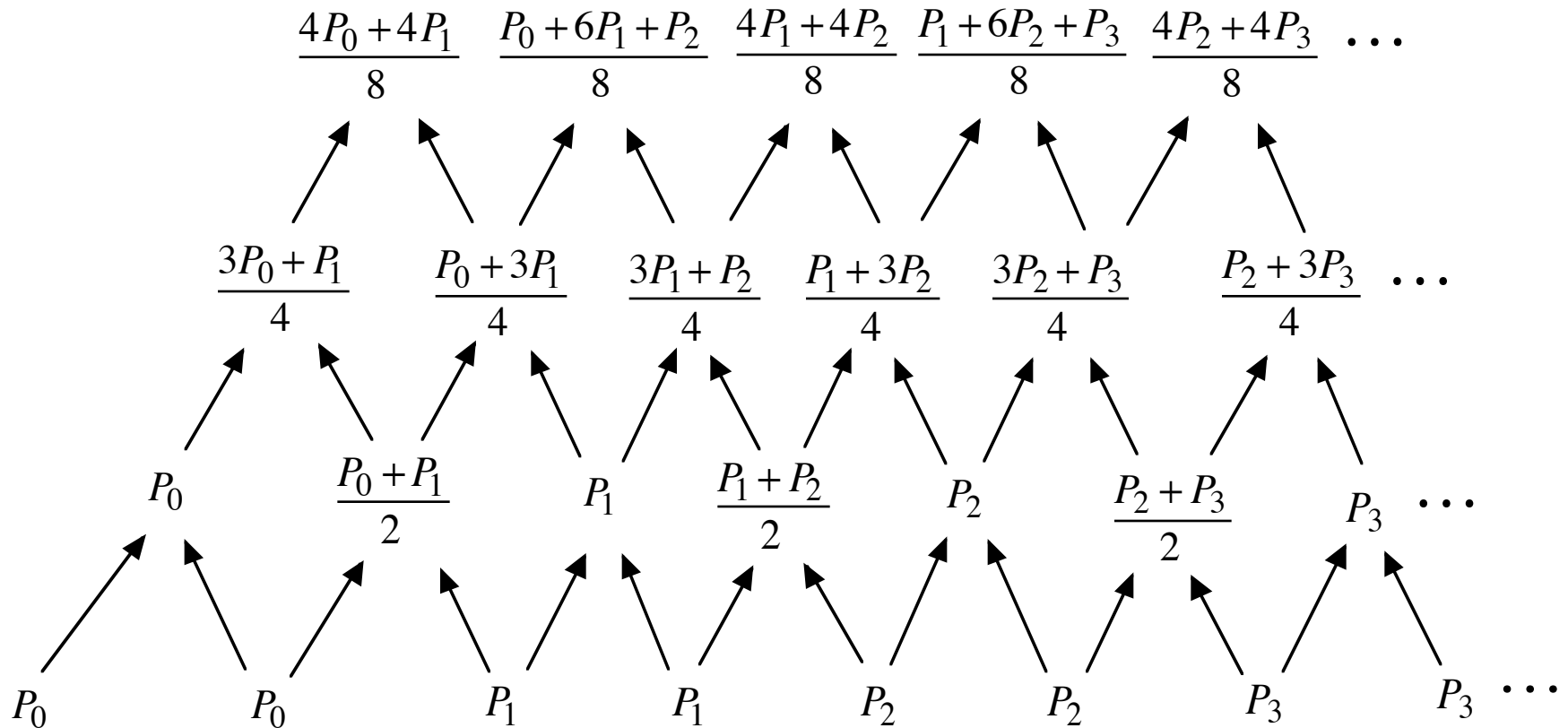
## Quadrilateral Meshes



## Triangular Meshes



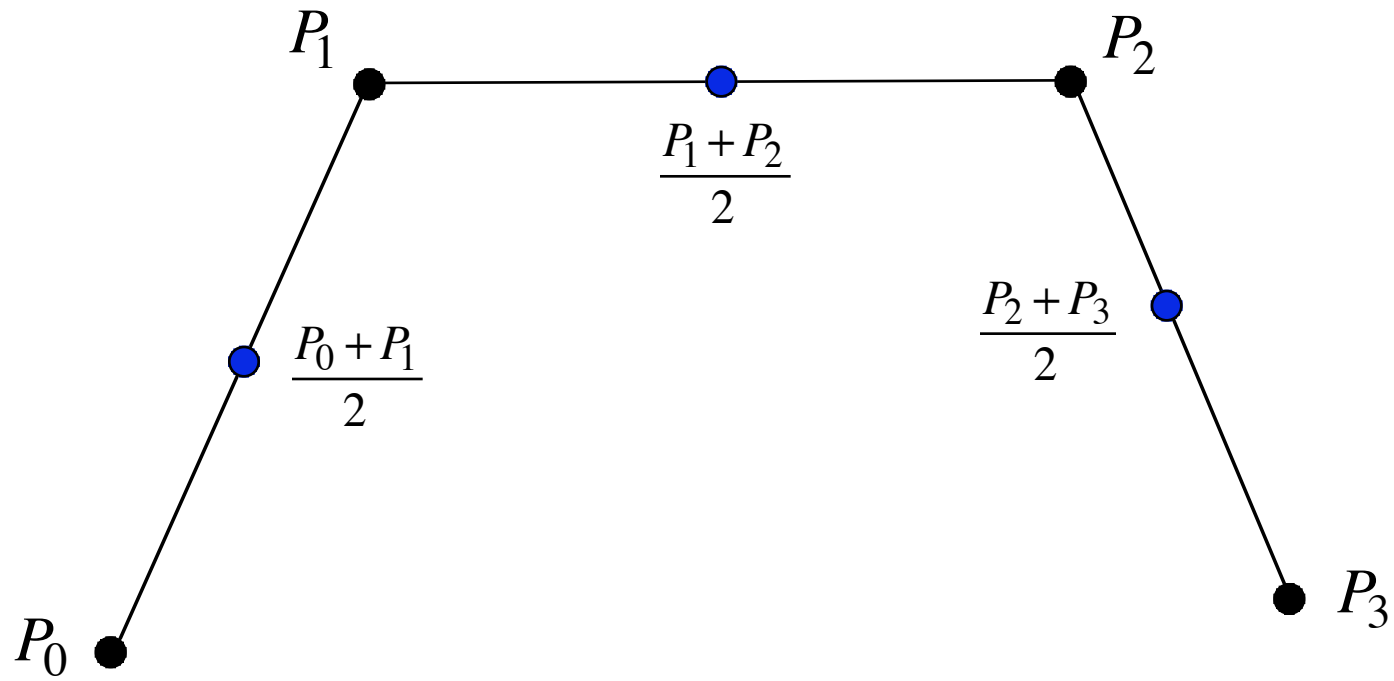
## Lane-Riesenfeld Knot Insertion Algorithm -- Cubic B-Splines



*Original Control Points*

*Split and Average*

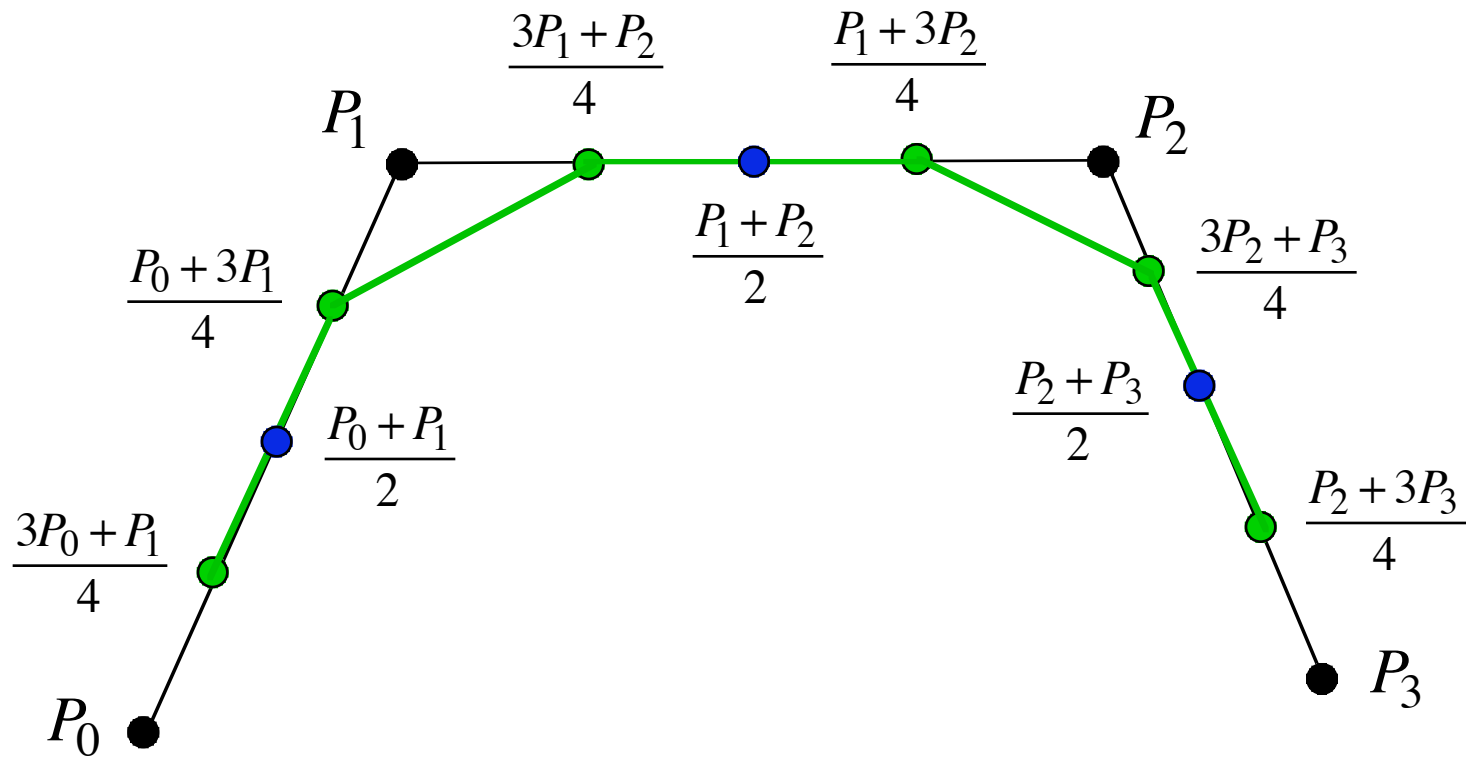
## Lane-Riesenfeld Knot Insertion Algorithm -- Cubic B-Spline Curves



*Step 1: Change Topology*

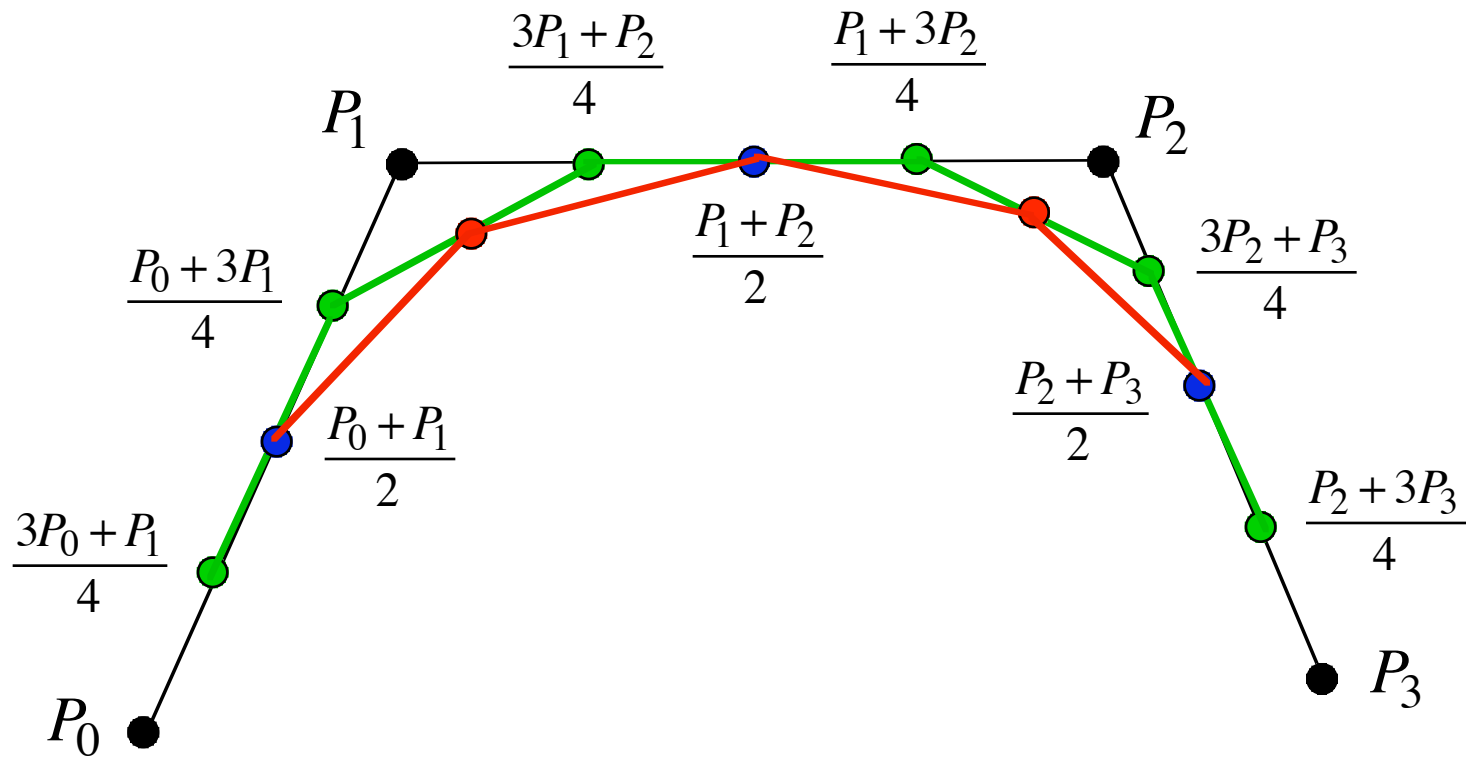
*Insert New Vertices at Centroids of Edges*

## Lane-Riesenfeld Knot Insertion Algorithm -- Cubic B-Spline Curves



*Step 2: Compute Centroids of Edges*

## Lane-Riesenfeld Knot Insertion Algorithm -- Cubic B-Spline Curves



*Step 3: Change Geometry*

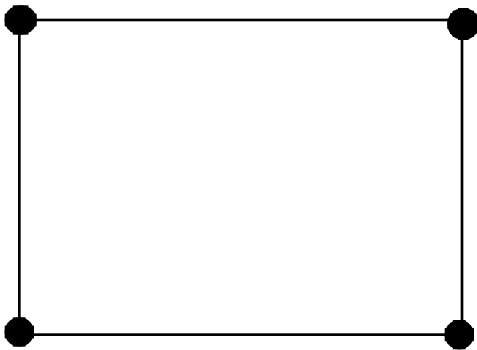
*Move Vertices to Centroids of Centroids*

## Lane-Riesenfeld Subdivision for Bicubic B-Spline Surfaces

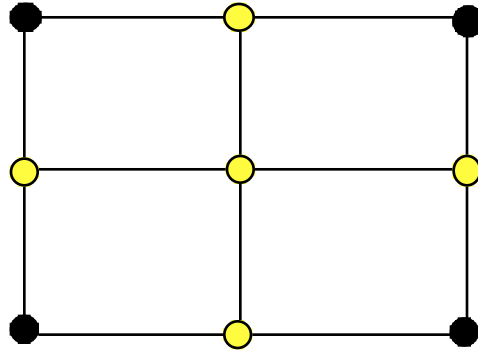
*Centroid Averaging*

- $Q \rightarrow \frac{C_1 + C_2 + C_3 + C_4}{4}$

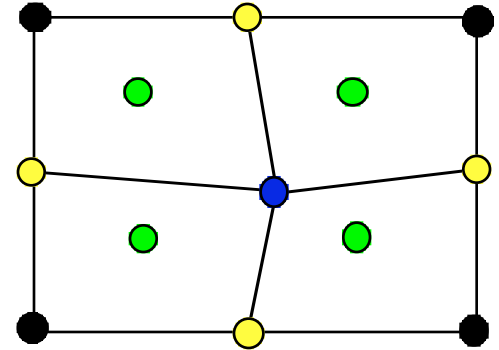
--  $C_1, C_2, C_3, C_4$  are the centroids of the faces adjacent to  $Q$



Original Quad



New Topology  
*Insert New Vertices  
at Centroids of Edges  
and Faces*



New Geometry  
*Move Vertices to the  
Centroids of Centroids  
of Surrounding Faces*

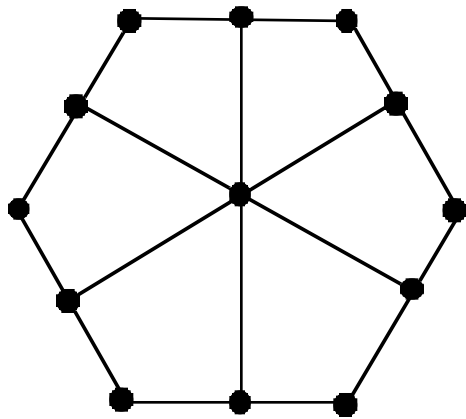
## Subdivision Surfaces -- Quadrilateral Meshes

*Centroid Averaging (Mimics Bicubic B-Splines)*

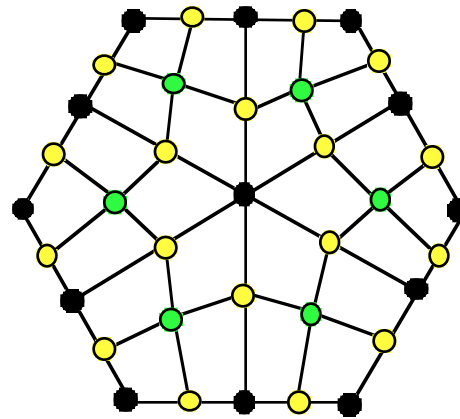
- $Q \rightarrow \frac{1}{n} \sum_{k=1}^n C_k$

--  $C_1, \dots, C_n$  are the centroids of the faces adjacent to  $Q$

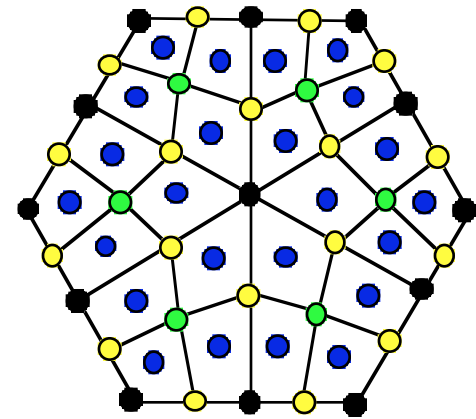
--  $n$  is the valence of  $Q$



*Extraordinary Vertex*



*New Topology*



*New Geometry*

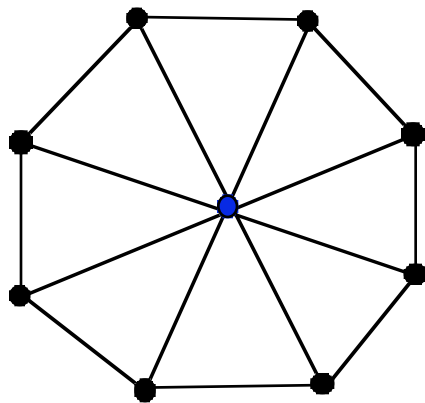
## Subdivision Surfaces -- Triangular Meshes

*Centroid Averaging (Mimics Three Direction Quartic Box Splines)*

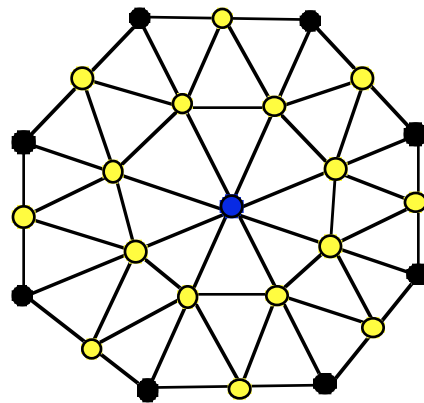
- $Q \rightarrow \frac{1}{n} \sum_{k=1}^n C_k$

- $C_1, \dots, C_n$  are the weighted centroids of the faces adjacent to  $Q$

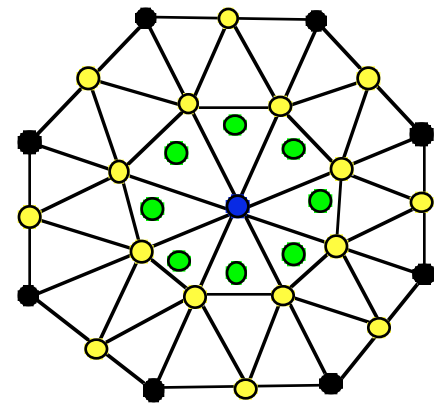
- $n$  is the valence of  $Q$



*Extraordinary Vertex*



*New Topology*



*New Geometry*

## Geri's Game -- Disney/Pixar



## Weaknesses of Subdivision Procedures

- No Explicit Formulas
- Difficult to Analyze Smoothness
- Difficult to Construct Good Subdivision Algorithms
- Difficult to Generate  $C^2$  Surfaces for Meshes with Extraordinary Points

## Applications

### *B-Splines*

- Manufacturing -- Automotive, Aerospace, Ship Building, . . .

### *Subdivision Surfaces*

- Entertainment -- Video Games, Cartoons, Movies, . . .

## **Themes**

1. Algebra vs. Geometry
2. Interpolation vs. Approximation
3. Formulas vs. Algorithms

**Part II:**  
**Algebraic Geometry**

## Algebraic Geometry and Geometric Modeling

*Algebraic geometry and geometric modeling both deal with curves and surfaces generated by polynomial equations. Algebraic geometry investigates the theoretical properties of polynomial curves and surfaces; geometric modeling uses polynomial, piecewise polynomial, and rational curves and surfaces to build computer models of mechanical components and assemblies for industrial design and manufacture.*

[Preface to *Contemporary Mathematics*, Vol. 334]

## Algebraic Geometry in Geometric Modeling

*Problems -- Analysis of Polynomial Curves and Surfaces*

- Implicitization  $P(t) = (x(t), y(t)) \rightarrow F(x, y) = 0$
- Intersection
- Singularities and Singular Loci
- Base Points
- Improper Parametrizations

*Solutions -- Elimination Theory (Solving Systems of Polynomial Equations)*

- Classical Resultants and Sparse Resultants (Standard)
- Grobner Bases (Slow)
- Syzygies and Mu-Bases -- Moving Curves and Surfaces (New)

## Outline

### 1. Polynomial Curves and Surfaces

- Examples and Pictures
- Analysis
  - Intersection
  - Implicitization
  - Inversion
  - Singularities

### 2. Elimination Theory

- Resultants
- Grobner Bases
- Mu-Bases