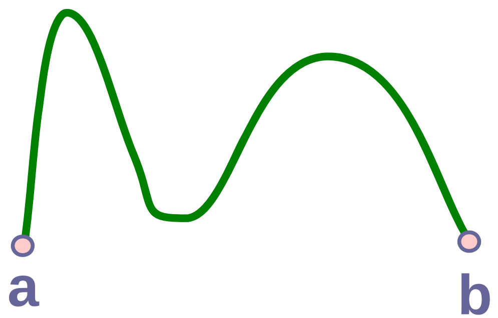
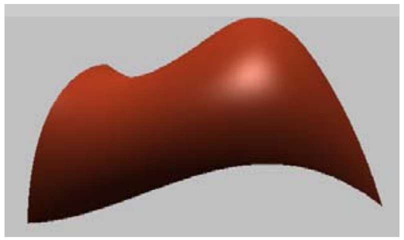
Computer Graphics: Curves and Surfaces

Part 2 – Lecture 14

Today's Outline

- Introduction to Curves and Surfaces
- Bézier Curves
- Bézier Surfaces

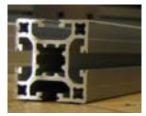




INTRODUCTION TO CURVES AND SURFACES

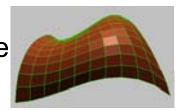
Why Do We Need Curves/Surfaces?

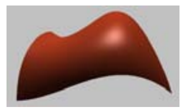
Curves



- Smooth interpolation for computer animation (e.g. motion control paths for objects and camera)
- Profile curves to create revolution or extrusion surfaces
- Control curves to create parametric surfaces

Surfaces



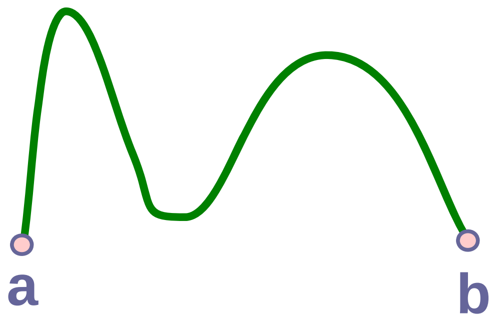


- Polygon mesh does not give us exact surface representation at every point (e.g. necessary for car/airplane design)
- Required level of detail (LOD) varies with size of polygon mesh on screen

Parametric Curves

Goals

- Smoothness
- Compact Representation
- Easy Control
- Easy Computation

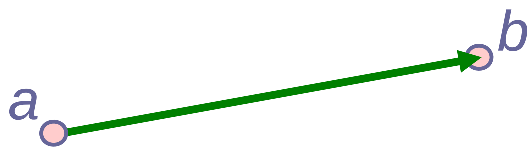


p(0) = a

p(1) = b

p(*t*) = ? when 0.0 <= t <= 1.0

Parametric Lines

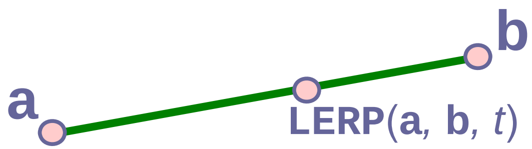


p(*t*) = **a** + (**b**-**a**) *t* — a "curve" that is a polynomial of degree 1, i.e., p(t) = $c_0 + c_1 t^1$

p(*t*) = **a** – *t* **a** + *t* **b** — rewriting in another form

p(*t*) = (1-*t*) **a** + *t* **b** — another rewriting

LERPing (a.k.a. **Tween**ing)



lerp = "Linear intERPolation"
also called "Tween" (in between)

Input:

Two points, a and b and a value, t, between 0 and 1.

Output:

A point a fraction *t* of the way from **a** to **b**.

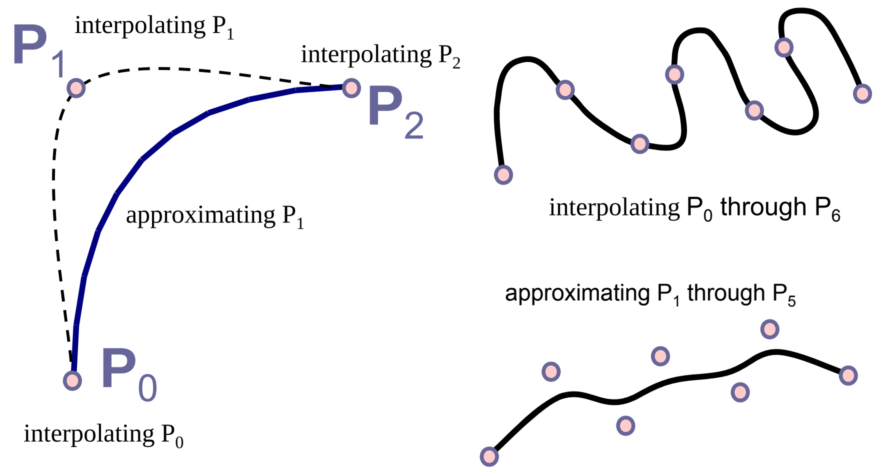
Code:

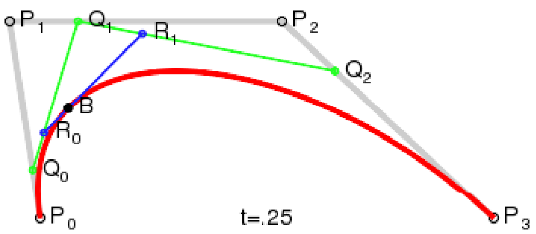
LERP(**a**, **b**, *t*) = (1-*t*)**a** + *t* **b**

Evaluation at NSTEPS points:

(1-*t*)**a** + *t* **b**, for t = 0.0 to 1.0 in steps of 1.0/NSTEPS

Interpolation vs. Approximation





BÉZIER CURVES

The *de Casteljau* Algorithm

Compute a parametric curve $\mathbf{P}(t)$

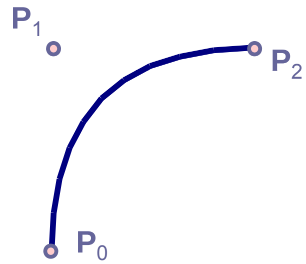
1. Interpolate control points $\mathbf{P_0}$ and $\mathbf{P_2}$
2. Approximate control point $\mathbf{P_1}$

Method:

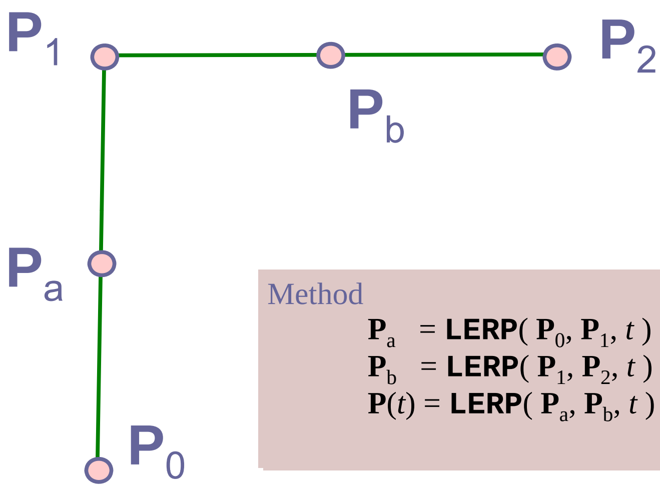
$\mathbf{P}_a = \mathbf{LERP}(\ \mathbf{P}_0, \mathbf{P}_1, t\ )$

$\mathbf{P}_b = \mathbf{LERP}(\ \mathbf{P}_1, \mathbf{P}_2, t\ )$

$\mathbf{P}(t) = \mathbf{LERP}(\ \mathbf{P}_a, \mathbf{P}_b, t\ )$



The *de Casteljau* Algorithm



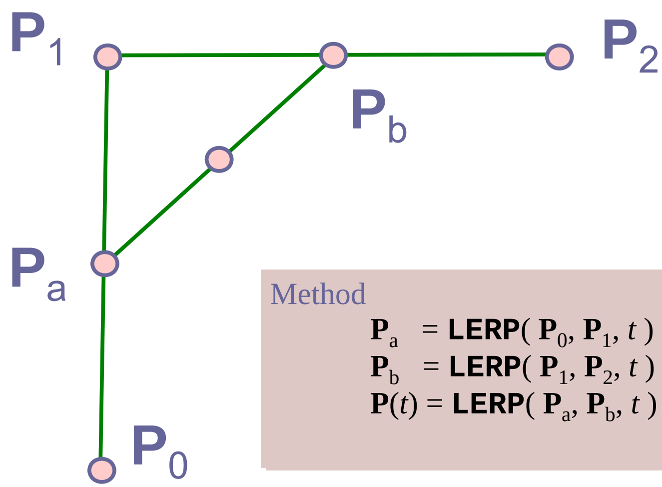
Method

$\mathbf{P}_a = \mathbf{LERP}(\ \mathbf{P}_0, \mathbf{P}_1, t\ )\ |_{t = 0.5}$

$\mathbf{P}_b = \mathbf{LERP}(\ \mathbf{P}_1, \mathbf{P}_2, t\ )\ |_{t = 0.5}$

$\mathbf{P}(t) = \mathbf{LERP}(\ \mathbf{P}_a, \mathbf{P}_b, t\ )$

The *de Casteljau* Algorithm



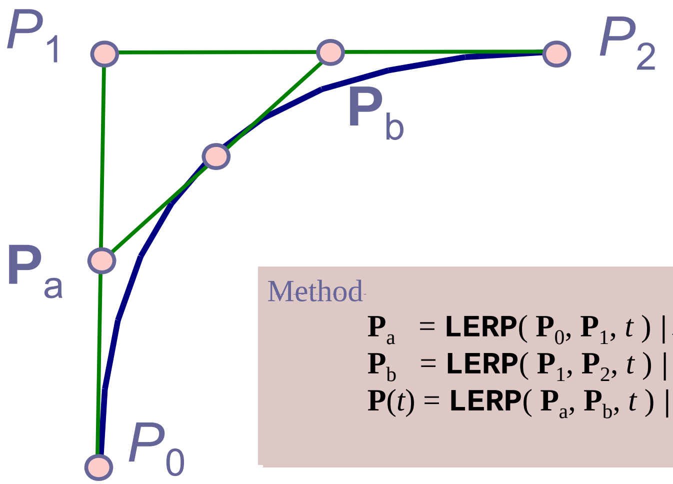
Method

$\mathbf{P}_a = \mathbf{LERP}(\ \mathbf{P}_0, \mathbf{P}_1, t\ )\ |_{t = 0.5}$

$\mathbf{P}_b = \mathbf{LERP}(\ \mathbf{P}_1, \mathbf{P}_2, t\ )\ |_{t = 0.5}$

$\mathbf{P}(t) = \mathbf{LERP}(\ \mathbf{P}_a, \mathbf{P}_b, t\ )\ |_{t = 0.5}$

The *de Casteljau* Algorithm



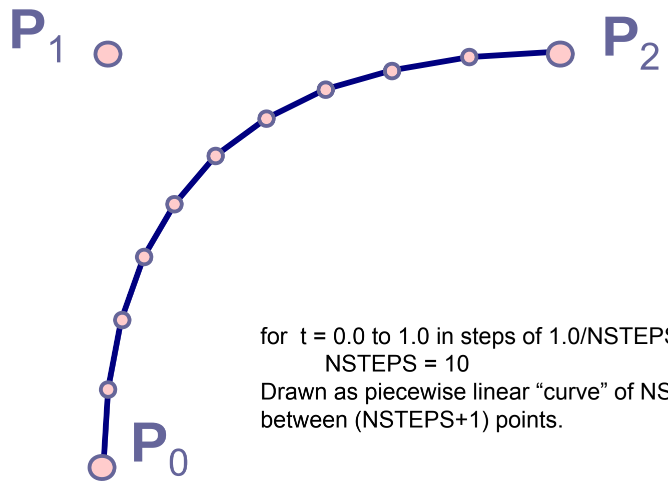
Method

$\mathbf{P}_a$ = **LERP**($\mathbf{P}_0$, $\mathbf{P}_1$, t) $|_{t = \text{all } t \{0,1\}}$

$\mathbf{P}_b$ = **LERP**($\mathbf{P}_1$, $\mathbf{P}_2$, t) $|_{t = \text{all } t \{0,1\}}$

$\mathbf{P}(t)$ = **LERP**($\mathbf{P}_a$, $\mathbf{P}_b$, t) $|_{t = \text{all } t \{0,1\}}$

The *de Casteljau* Algorithm



for t = 0.0 to 1.0 in steps of 1.0/NSTEPS
NSTEPS = 10
Drawn as piecewise linear "curve" of NSTEPS lines between (NSTEPS+1) points.

Quadratic Bézier Curve

Effect of the *de Casteljau* algorithm is:

$$\mathbf{P}(t) = \mathbf{LERP}(\ \mathbf{LERP}(\mathbf{P}_0, \mathbf{P}_1, t),\ \ \mathbf{LERP}(\mathbf{P}_1, \mathbf{P}_2, t),\ \ t)$$

$$\mathbf{P}(t) = (1-t)\ [(1-t)\mathbf{P}_0 + t\ \mathbf{P}_1] + t\ [(1-t)\mathbf{P}_1 + t\ \mathbf{P}_2]$$

$$\boxed{\mathbf{P}(t) = (1-t)^2\ \mathbf{P}_0 + 2t\ (1-t)\mathbf{P}_1 + t^2\ \mathbf{P}_2}$$

Easy to program

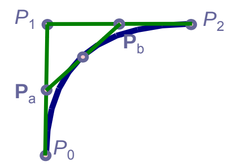
→ Called a **quadratic** Bézier curve

Demo: Bezier applet
http://www.cs.unc.edu/~mantler/research/bezier/index.html

Weighting Functions

- Linear interpolation (2 control points)
  - $\mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\ \mathbf{P}_1$
  - $\mathbf{x}(t) = (1-t)\mathbf{x}_0 + t\ \mathbf{x}_1$
  - $\mathbf{y}(t) = (1-t)\mathbf{y}_0 + t\ \mathbf{y}_1$
  - $\mathbf{z}(t) = (1-t)\mathbf{z}_0 + t\ \mathbf{z}_1$

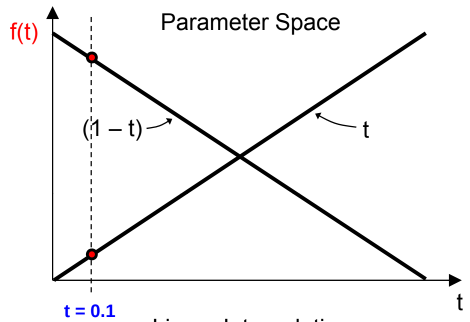
- Quadratic Bézier Curve (3 control points)
  - $\mathbf{P}(t) = (1-t)^2\ \mathbf{P}_0 + 2t\ (1-t)\mathbf{P}_1 + t^2\mathbf{P}_2$
  - $\mathbf{x}(t) = (1-t)^2\ \mathbf{x}_0 + 2t\ (1-t)\mathbf{x}_1 + t^2\mathbf{x}_2$
  - $\mathbf{y}(t) = (1-t)^2\ \mathbf{y}_0 + 2t\ (1-t)\mathbf{y}_1 + t^2\mathbf{y}_2$
  - $\mathbf{z}(t) = (1-t)^2\ \mathbf{z}_0 + 2t\ (1-t)\mathbf{z}_1 + t^2\mathbf{z}_2$

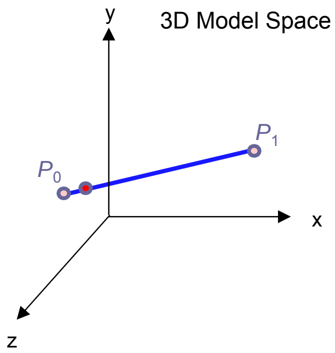


- Points on both curves are weighted sums of control points
- Weights are functions of the parameter *t*
- Weighting functions like attractors or magnets that pull the curve towards the control point

Linear Weighting Functions

Evaluated at values of t, multiplied with control points P_0, P_1, P_2, and summed





Linear Interpolation

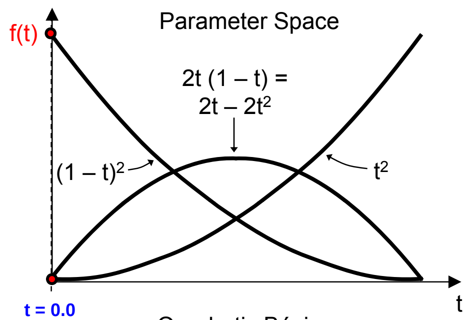
$P(t) = (1 - t)P_0 + t\,P_1$

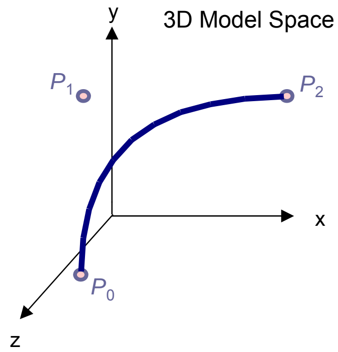
$P(0.1) = (1 - 0.1)P_0 + 0.1\,P_1$

$P(0.1) = 0.9\,P_0 + 0.1\,P_1$

Quadratic Weighting Functions

Evaluated at values of t, multiplied with control points P_0, P_1, P_2, and summed





Quadratic Bézier

$P(t) = (1 - t)^2\,P_0 + 2t\,(1 - t)P_1 + t^2P_2$

$P(0.0) = (1 - 0.0)^2P_0 + 2(0.0)(1\text{-}0.0)\,P_1 + (0.0)^2P_2$

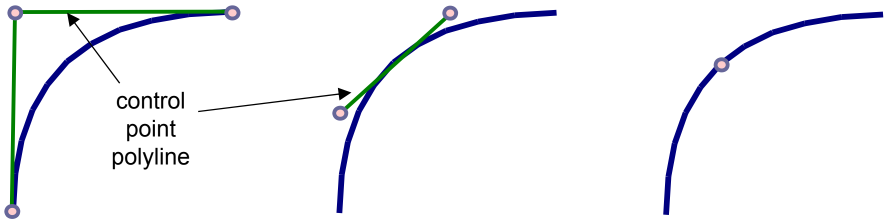
$P(0.0) = 1.0\,P_0 + 0.0\,P_1 + 0.0\,P_2 = P_0$

de Casteljau Algorithm with *n* Points

Given control point polyline with *n* points

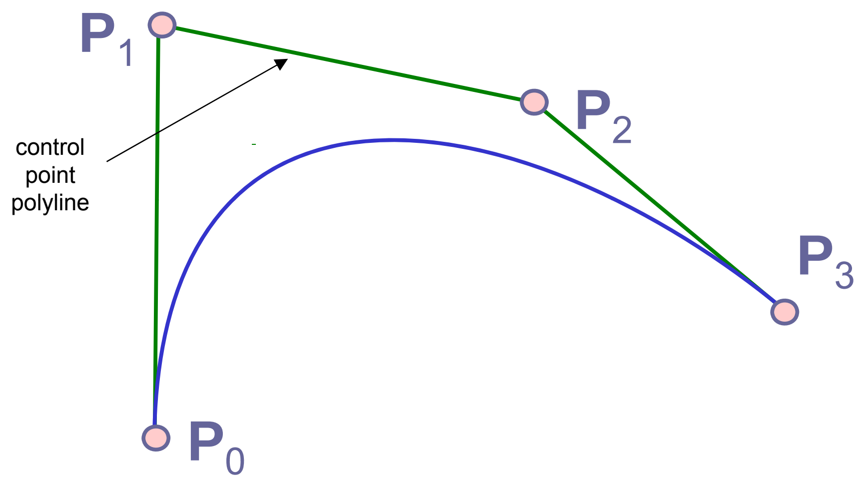
Repeat until control point polyline has 1 point:

Create new control point polyline by **LERP**ing each pair of adjacent control points

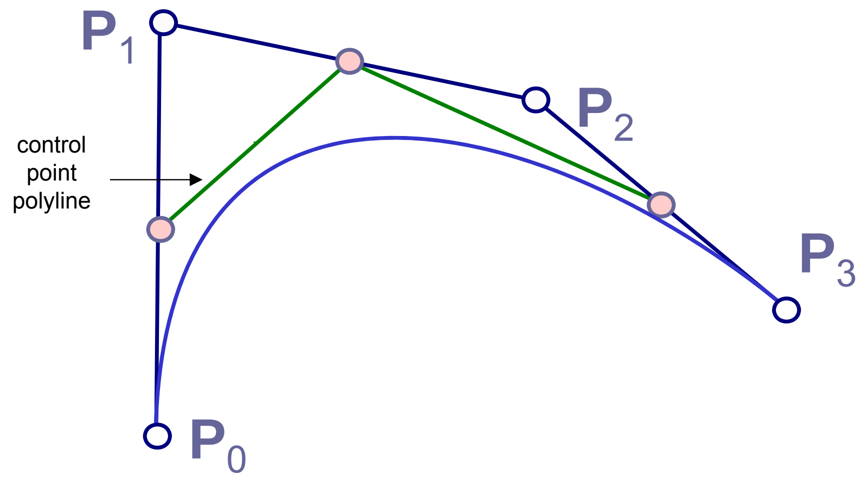


de Casteljau, N=3

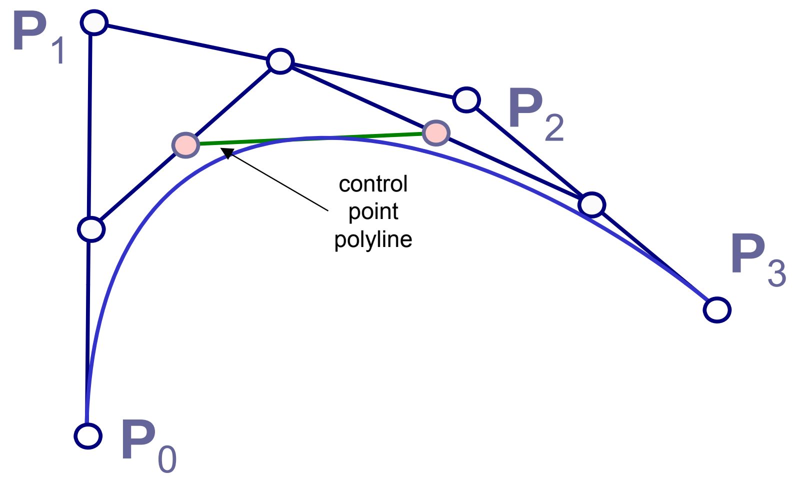
de Casteljau, N=4



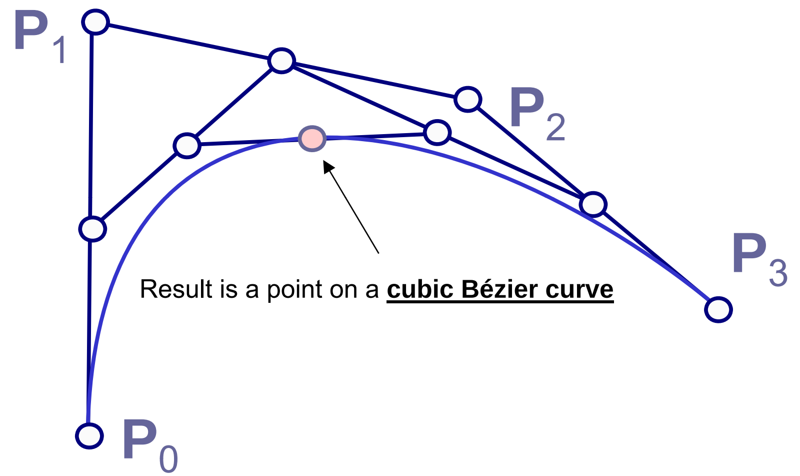
de Casteljau, N=4



de Casteljau, N=4



de Casteljau, N=4



Cubic Bézier Curve

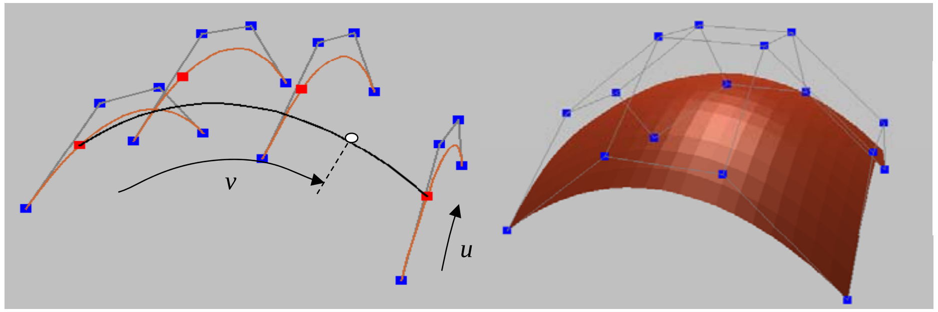
Effect of the N=4 point *de Casteljau* algorithm is:

$$\mathbf{P}(t) = \mathrm{LERP}(\ \mathrm{LERP}(\mathrm{LERP}(\mathbf{P_0}, \mathbf{P_1}, t), \mathrm{LERP}(\mathbf{P_1}, \mathbf{P_2}, t), t),$$
$$\mathrm{LERP}(\mathrm{LERP}(\mathbf{P_1}, \mathbf{P_2}, t), \mathrm{LERP}(\mathbf{P_2}, \mathbf{P_3}, t), t),\ \ t)$$
$$\mathbf{P}(t) = (1-t)\,[(1-t)\,[(1-t)\mathbf{P_0} + t\,\mathbf{P_1}] + t\,[(1-t)\mathbf{P_1} + t\,\mathbf{P_2}]] +$$
$$t\,[(1-t)\,[(1-t)\mathbf{P_1} + t\,\mathbf{P_2}] + t\,[(1-t)\mathbf{P_2} + t\,\mathbf{P_3}]]$$
$$\mathbf{P}(t) = (1-t)^3\,\mathbf{P_0} + 3t\,(1-t)^2\,\mathbf{P_1} + 3t^2\,(1-t)\,\mathbf{P_2} + t^3\,\mathbf{P_3}$$

→ Called a **cubic** Bézier curve

Demo: Bézier applet again

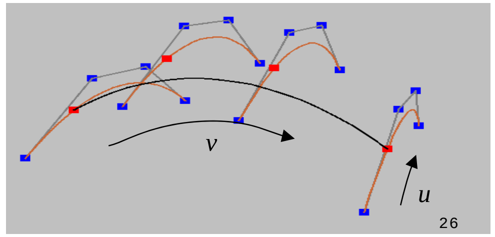
http://www.cs.unc.edu/~mantler/research/bezier/index.html



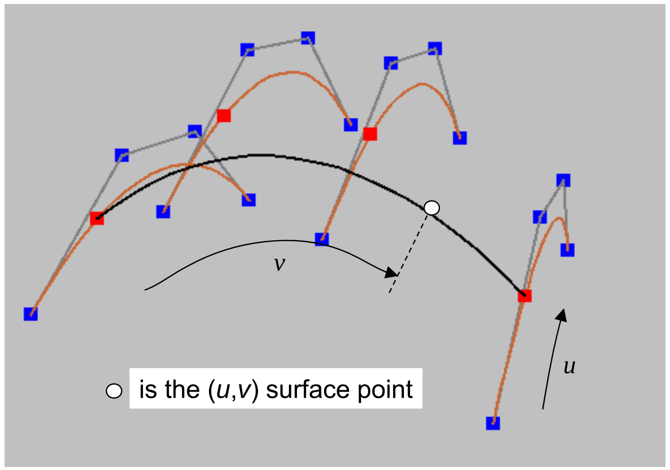
BÉZIER SURFACES

Bézier Surfaces (Patches)

- Surface P(u, v) swept out by a moving Bezier curve V(v)
- Describe the trajectory of V's 4 control points with 4 cubic Bézier curves: $U_1(u)$, $U_2(u)$, $U_3(u)$, $U_4(u)$
- To calculate P(u, v):
  1. Calculate $U_1(u)$, $U_2(u)$, $U_3(u)$, $U_4(u)$
  2. Calculate V(v) using $U_1(u)$, $U_2(u)$, $U_3(u)$, $U_4(u)$ as V's control points
- Patch has 16 control points in total



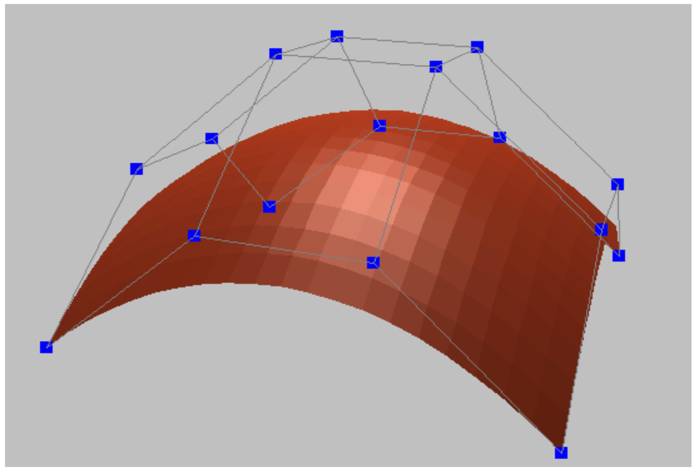
Bézier Patches



16 control points; 4 cubic Bezier curves; (u, v) defines a point on the patch

Demo: http://www.cs.auckland.ac.nz/compsci372s2c/christofLectures/BezierPatchApplet/

Bézier Patches



Control points are a 4 x 4 mesh

SUMMARY

Summary

- Bézier Curves
  1. **LERP**ing between control points to get new control points
  2. **LERP**ing again between the new control points
  3. Until there is only one point
- Bézier Patches
  - Surface P(u, v) swept out by a moving Bezier curve V(v)
  - Describe V's control points with 4 cubic Bézier curves: $U_1(u)$, $U_2(u)$, $U_3(u)$, $U_4(u)$

Old ray tracing assignment images
http://www.cs.auckland.ac.nz/GG/weeklyimages/2006.php

References:

- Curves: Hill, Chapter 10.3
- Bézier Curves: Hill, Chapter 10.4
- Bézier Patches: Hill, Chapter 10.11.3

Quiz

1. What is the difference between interpolation and approximation?
2. How do you construct a quadratic Bézier curve with the *de Casteljau* algorithm given 3 control points?
3. How do you construct a cubic Bézier curve with the *de Casteljau* algorithm given 4 control points?
4. How do you construct a Bézier surface patch given 16 control points?