

# Computer Vision

## Week 5

Segmentation: boundary based  
techniques, continued  
Model based approaches in boundary  
detection

### Model-based Approaches:

- Curve fitting: Lines, circles, splines
- Curve approximation: Linear and nonlinear regression
- Robust algorithm
- Some other curves
- Regularization theory- inspired: GED (Gokmen and Jain)

## Curve fitting

- General approach: Segment arcs into simple segments
- Then, model these with simple curves such as:
  - Line segments
  - Circular arcs
  - Cubic polynomial curves
- Curve fitting vs. curve approximation

## Curve Representation

- 3 types of representation for curves:
  - Explicit
  - Implicit
  - Parametric
- Simplest curve: Line
  - Explicit:  $y=mx+B$
  - Implicit:  $ax+by+c=0$
  - Parametric:  $x=x_1+(x_2-x_1)t$ ;  
 $y=y_1+(y_2-y_1)t$

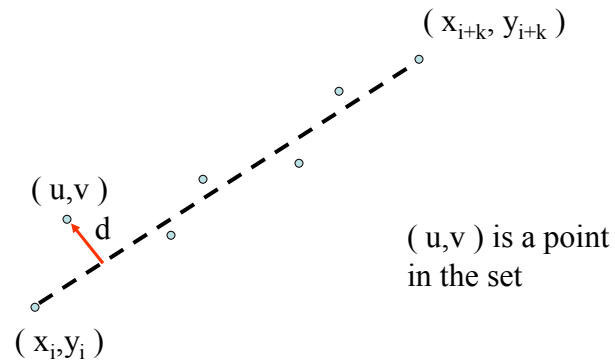
## Polyline Representation

- Sequence of line segments joined end to end
- Fit edge list with a sequence of line segments
- Polyline interpolates a selected subset of edge points, BUT ends of segments are members of original edge list
- Vertices: where end points join

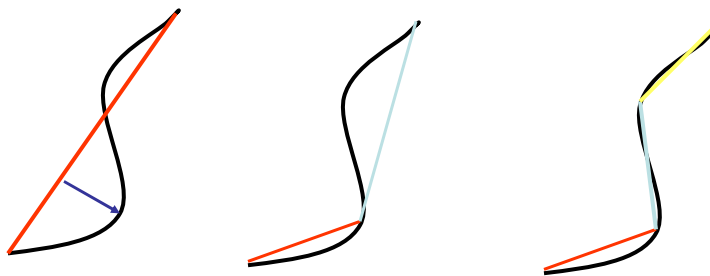
## Polyline Algorithm

1. Start with ordered list of edge points as input,  $\{ (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \}$
2. Join point  $(x_i, y_i)$  and  $(x_{i+k}, y_{i+k})$ , where  $k > 1$ , with a straight line segment
3. Determine which point contained in the subset between  $(x_i, y_i)$  and  $(x_{i+k}, y_{i+k})$  is maximum distance,  $d$ , from the constructed line segment
4. If  $d$  is less than the accepted maximum error, increase  $k$  to  $k+1$  and repeat steps 2 and 3. If  $d$  is greater than the acceptable error, reduce  $k$  by 1. Recalculate line segment. This is the polyline segment for the subset of points.
5. Use point  $i+k$  as the starting point for the next polyline and continue the process until all points in the set have been joined by polylines

How do you determine error,  $d$ ?



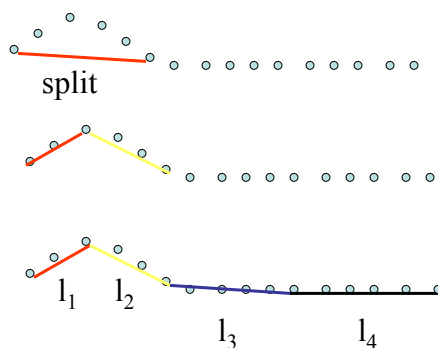
Polyline Split Method Example



## Polyline: Hop Along Algorithm

- 1. Start with first k edges
- 2. Fit a line segment between first and last edges on sublist
- 3. If normalized maximum error too large, shorten sublist at maximum error, go to step 2.
- 4. If line fit succeeds, compare orientation with previous line segment, if similar, replace two line segments with one segment
- Continue to next k edges, go to step 2

## Hop Along Algorithm Example



Compare orientation of  $l_3$  and  $l_4$  and combine into one segment

## Curve approximation: Least Squares Fit

Consider fitting a straight line through a set of data points

- ease requirement of polyline approximation
- first and last point of the function approximation do not have to be edge points

## Least Squares

In function **interpolation** we match the original data exactly, in **approximation** we relax this requirement

The approach taken for 'best fit' is to minimize the square of the differences between the **data value** and the value of the **approximation function**

# Linear Least Squares

Approximate data by a straight line of the form  $f(x) = ax + b$

We need to determine the coefficients  $a$  and  $b$

Minimize sum of squares of the distance between given value  $y_i$  ( original data ) and the computed function value:

$$f_i = a \cdot x_i + b$$

Example: ( 1, 2.1 ), ( 2, 2.9 ), ( 5, 6.1 ), ( 7, 8.3 )

Problem is to find the coefficients  $a$  and  $b$

The optimal coefficients of a straight line fit minimize the total squared error:

$$E = ( f(x_1) - y_1 )^2 + ( f(x_2) - y_2 )^2 + ( f(x_3) - y_3 )^2 + ( f(x_4) - y_4 )^2$$

$$E = [ ax_1 + b - y_1 ]^2 + [ ax_2 + b - y_2 ]^2 + [ ax_3 + b - y_3 ]^2 + [ ax_4 + b - y_4 ]^2$$

Minimize  $E$  with respect to two variables,  $a$  and  $b$

Independent of the number of data points we want to fit a straight line that minimizes error in y-coordinate. We have two equations with two unknowns:

**For  $i = 1:n$**

$$a \sum x_i^2 + b \sum x_i = \sum x_i * y_i$$

$$a \sum x_i + b \sum 1 = \sum y_i \quad \text{note: } b \sum 1 = b * n$$

Need to calculate four different summations. In statistics literature:

$$S_{xx} = \sum x_i^2 \quad S_x = \sum x_i$$

$$S_{xy} = \sum x_i * y_i \quad S_y = \sum y_i$$

Therefore:

$$aS_{xx} + bS_x = S_{xy}$$

$$aS_x + bn = S_y$$

We can also put the equations in matrix form. Starting with:

$$aS_{xx} + bS_x = S_{xy}$$

$$aS_x + bn = S_y$$

$$\begin{pmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & \sum 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum x_i * y_i \\ \sum y_i \end{pmatrix}$$

## Least squares vs. Total least squares

- Least squares does not minimize distance of point to line but minimizes error in the y coordinate
- Total least squares minimizes total error:

$$d=ax+by+c \quad (a^2+b^2=1)$$

Minimize  $\Sigma d^2$

## Who came from which line?

- Assume we know how many lines there are - but which lines are they?
  - easy, if we know who came from which line
- Three strategies
  - Incremental line fitting
  - K-means
  - RANSAC

**Algorithm 15.1:** Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large

```
Put all points on curve list, in order along the curve
Empty the line point list
Empty the line list
Until there are too few points on the curve
  Transfer first few points on the curve to the line point list
  Fit line to line point list
  While fitted line is good enough
    Transfer the next point on the curve
      to the line point list and refit the line
  end
  Transfer last point(s) back to curve
  Refit line
  Attach line to line list
end
```

**Algorithm 15.2:** K-means line fitting by allocating points to the closest line and then refitting.

```
Hypothesize  $k$  lines (perhaps uniformly at random)
or
Hypothesize an assignment of lines to points
  and then fit lines using this assignment

Until convergence
  Allocate each point to the closest line
  Refit lines
end
```

# RANSAC

- Choose a small subset uniformly at random
- Fit to that
- Anything that is close to result is signal; all others are noise
- Refit
- Do this many times and choose the best
- Issues
  - How many times?  
Often enough that we are likely to have a good line
  - How big a subset?  
Smallest possible
  - What does close mean?  
Depends on the problem
  - What is a good line?  
One where the number of nearby points is so big it is unlikely to be all outliers

## Algorithm 15.4: RANSAC: fitting lines using random sample consensus

```
Determine:
   $n$  — the smallest number of points required
   $k$  — the number of iterations required
   $t$  — the threshold used to identify a point that fits well
   $d$  — the number of nearby points required
        to assert a model fits well
Until  $k$  iterations have occurred
  Draw a sample of  $n$  points from the data
  uniformly and at random
  Fit to that set of  $n$  points
  For each data point outside the sample
    Test the distance from the point to the line
    against  $t$ ; if the distance from the point to the line
    is less than  $t$ , the point is close
  end
  If there are  $d$  or more points close to the line
  then there is a good fit. Refit the line using all
  these points.
end
Use the best fit from this collection, using the
fitting error as a criterion
```

## Other curves

- Implicit curves:  $f(x,y)=0$ 
  - Example: parabola:  $x^2-y=0$
  - $x^2+y^2-R^2=0$
- Parametric curves:
  - Polynomials; cubic polynomials are popular

## Cubic Splines

- Third degree polynomial curves that can be controlled via “control points”
  - B-splines,
  - Uniform B-splines,
  - non-uniform B-splines
  - NURBS

## Polynomials

- Polynomials are curved naturally and easy to work with

–  $x = c_{x0} + c_{x1}t + c_{x2}t^2 + \dots + c_{xn}t^n$

– In general:

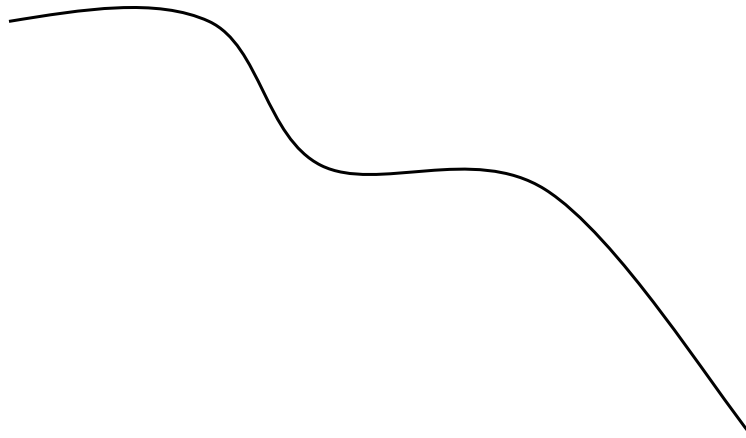
$$q(t) = \sum_{k=0}^n t^k c_k$$

– where:

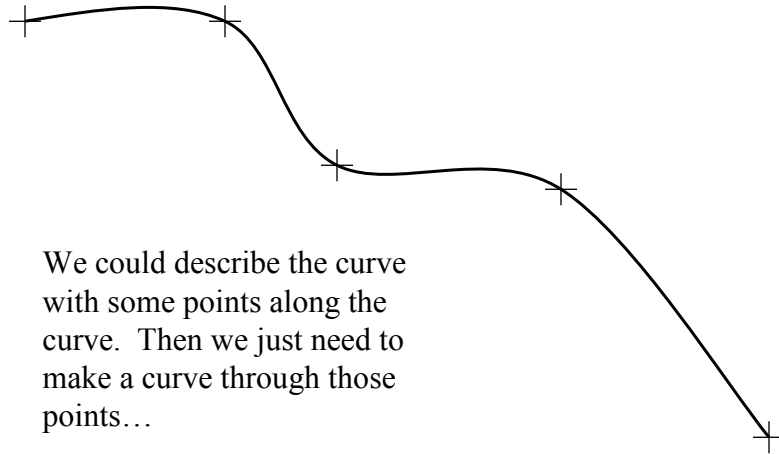
“n + 1 degrees of freedom” or “order n”

$$c_k = \begin{bmatrix} c_{xk} \\ c_{yk} \\ c_{zk} \end{bmatrix}$$

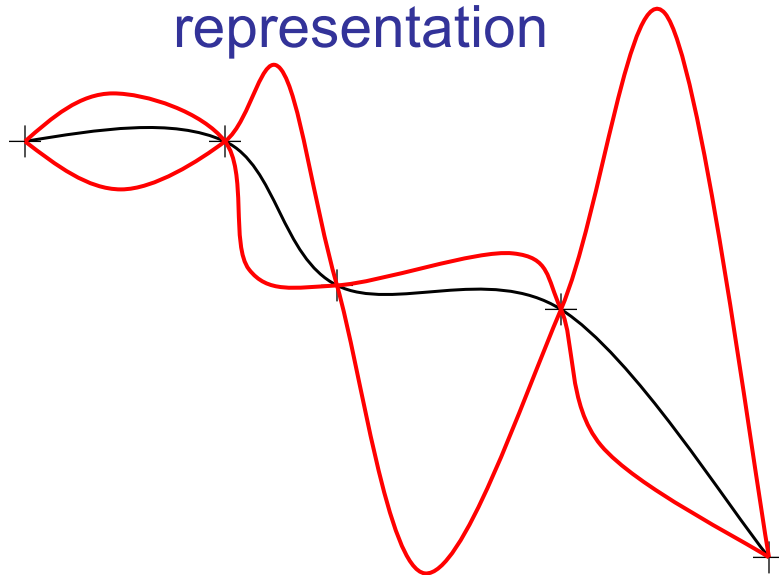
Now, how to describe this curve?



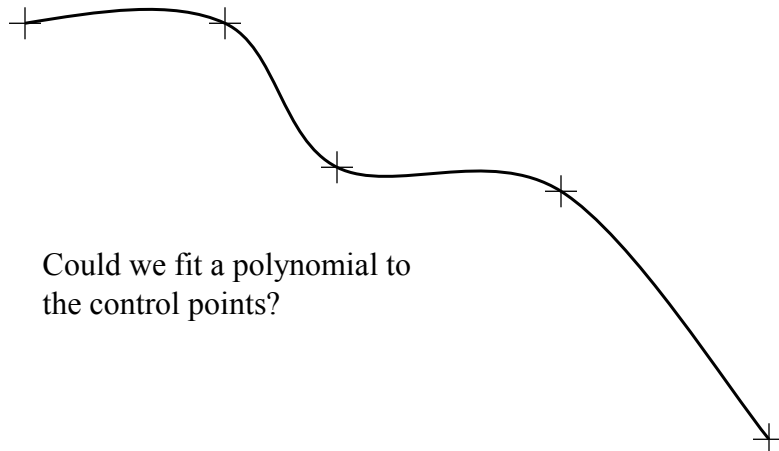
## Control Points



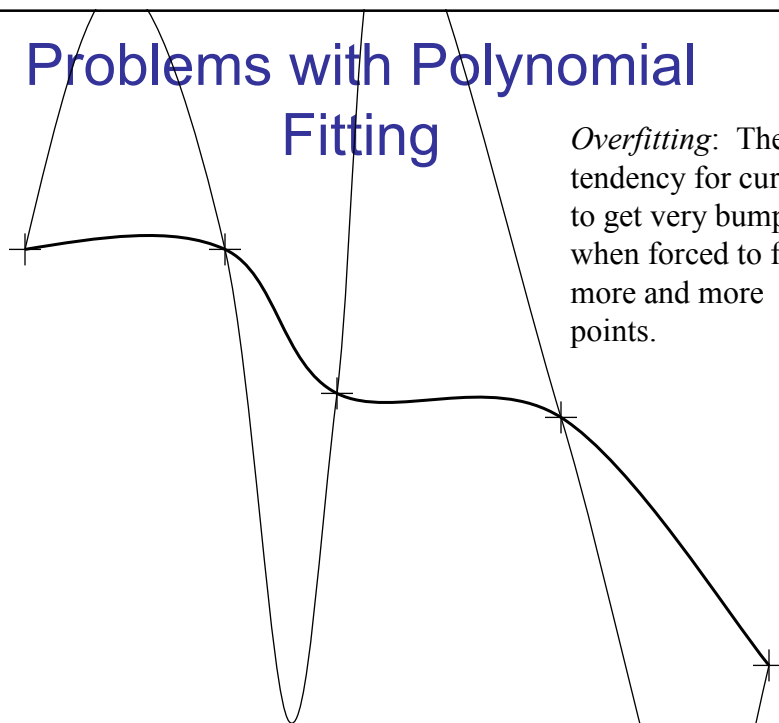
## Control Points are not a unique representation



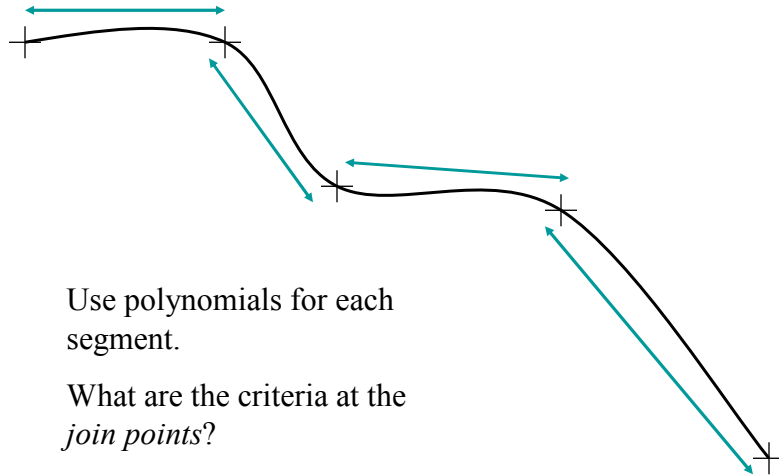
## Polynomial Fitting



## Problems with Polynomial Fitting



## Answer: Do the curve in segments



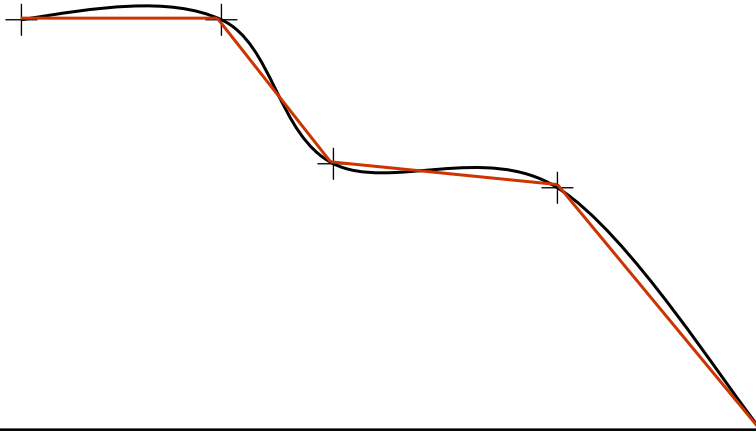
## Connections at join points...

- At the join points:
  - $G^0$  continuity – The ends touch (necessary)
  - $G^1$  continuity – The *slope* at the joint is the same (necessary)
  - $C^1$  continuity – The *first derivative* at the joint is the same (not necessary)



## Which degree polynomial?

- Linear, quadric, cubic..?
- Linear polynomials are line segments:



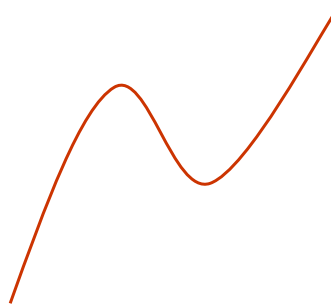
## We love cubics

- We'll use cubic polynomials for curve segments

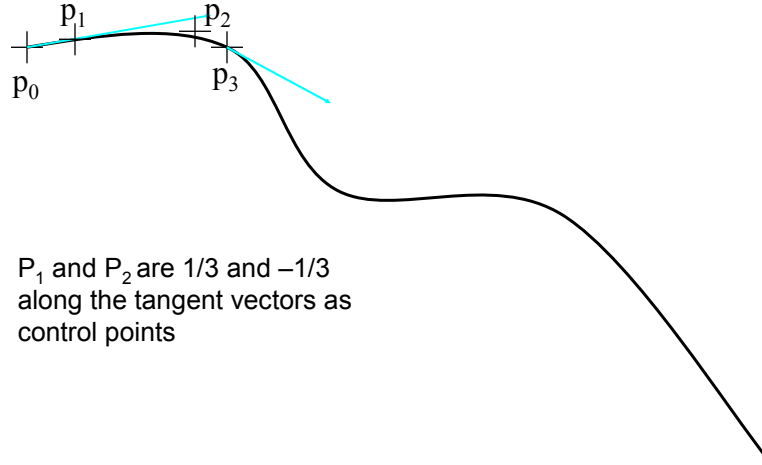
–  $x = c_{x0} + c_{x1}t + c_{x2}t^2 + c_{x3}t^3$

– Can be expressed as:

$$q(t) = \sum_{k=0}^3 t^k c_k \quad c_k = \begin{bmatrix} c_{kx} \\ c_{ky} \\ c_{kz} \end{bmatrix}$$



## A Bézier Curve



## Blending function form for Bezier polynomials

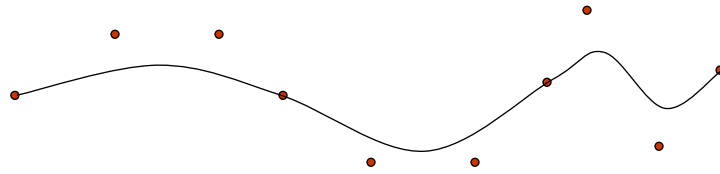
$$Q(t) = \sum_{i=0}^3 c_i t^i = \sum_{i=0}^3 P_i B_i(t)$$

$$Q(t) = \underbrace{(1-t)^3}_{B_0(t)} P_0 - \underbrace{3t(1-t)^2}_{B_1(t)} P_1 + \underbrace{3t^2(1-t)}_{B_2(t)} P_2 + \underbrace{t^3}_{B_3(t)} P_3$$

Bezier Blending polynomials: Bernstein polynomials

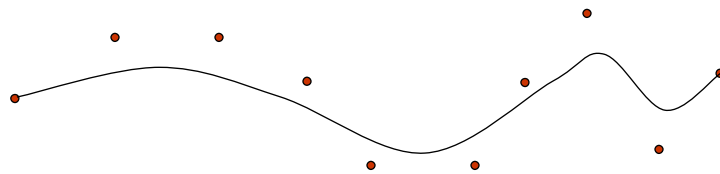
## Interpolating Curves

- An interpolating curve will hit certain points on the curve.
  - Bezier curves hit the end points
  - What if we have a lot of segments?



## What's bad about that?

- Specifying points on the curve (surface) is often difficult
  - We like to specify end points, but not other points
- What we would really like:



## What else do we want?

- We want local control
  - A control point effects a limited range of the curve
- We want more control points
  - Why only 4?

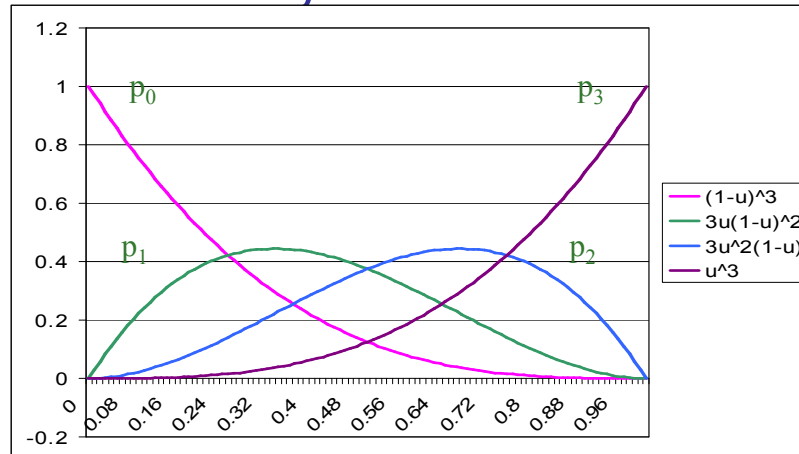
## Some ideas

- Remember that a Bézier curve can be defined as:

$$p(t) = \sum_{k=0}^3 b_k(t) p_k$$

- This uses a function  $b_k(t)$  that controls the influence of the points.

## The Bézier Blending Polynomials

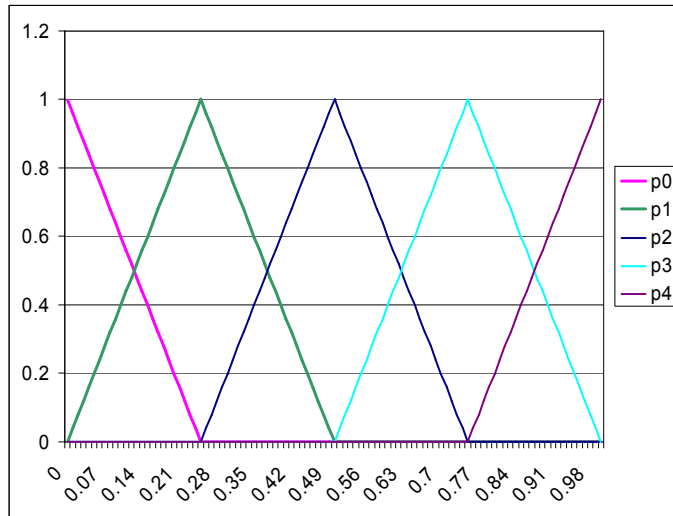


$$p(t) = \sum_{k=0}^3 b_k(t) p_k$$

## Can we extend this?

- Suppose we have 5 points?
  - Can we create a blending function of some kind?

## One (dumb) idea...



What would this draw?

## Can we do better?

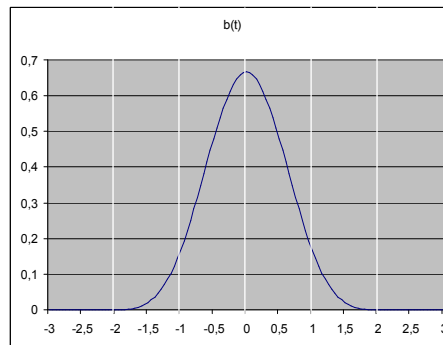
- What do we want?
  - Smooth curve segments
    - Cubics seem to work pretty well
  - Local control
    - Limit the range of influence of a point
  - Any number of control points
  - Lots of flexibility

## Cubic B-Splines

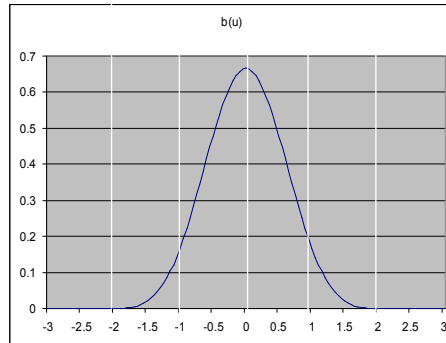
- Let's lay down some ground rules
  - Cubic (nice compromise between curves, control, and complexity)
  - Local control
    - Point effects a limited range of the curve
  - We'll not worry about the end points for now...

## Cubic B-Spline Function

$$b(t) = \begin{cases} 0 & \text{if } t \leq -2 \\ \frac{1}{6}(2+t)^3 & \text{if } -2 < t \leq -1 \\ \frac{1}{6}(2+t)^3 - \frac{2}{3}(1+t)^3 & \text{if } -1 < t \leq 0 \\ -\frac{2}{3}(1-t)^3 + \frac{1}{6}(2-t)^3 & \text{if } 0 < t \leq 1 \\ \frac{1}{6}(2-t)^3 & \text{if } 1 < t \leq 2 \\ 0 & \text{if } 2 < t \end{cases}$$



## Some Observations



The points where we tie one function to the next are called “knots”

The knots are at integral values.

$t$  values no longer 0 to 1

Clear “transition points” in the equation (but continuous at these points)

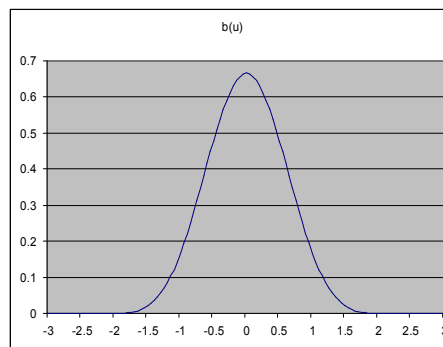
## Making this the appropriate function for a control point

- We'll have  $n+1$  control points ( $p_0, \dots, p_n$ ) corresponding to  $t$  values of 0 to  $n$ .

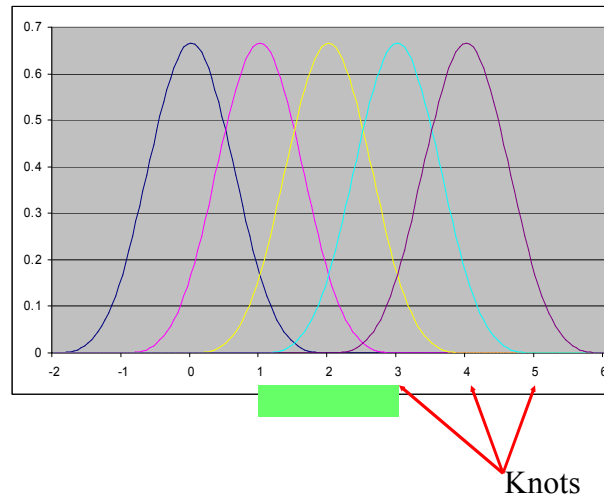
$$b_i(t) = b(t - i)$$

$$p(t) = \sum_{i=0}^n b_i(t) p_i$$

$u$  is valid in the range  
1 to  $n-1$

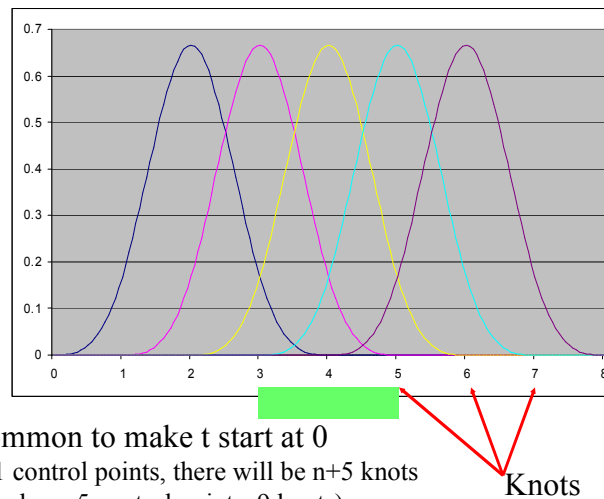


## Cubic B-Spline for 5 Point Curve



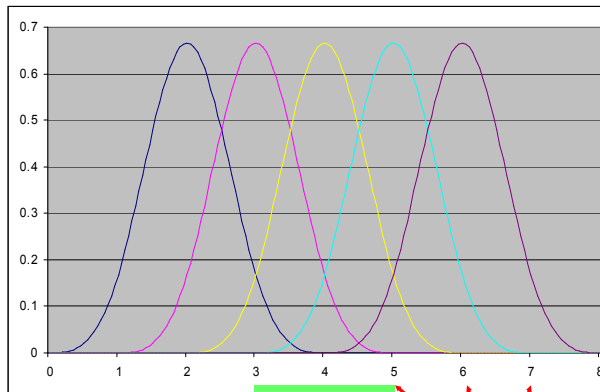
## Modifying the t value

t is valid in the range  
3 to n+1



It's common to make t start at 0  
For n+1 control points, there will be n+5 knots  
(here we have 5 control points, 9 knots)

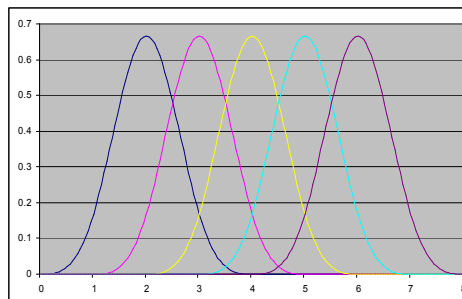
## Uniform B-Splines



Knots are evenly spaced

## Formal definition

- Parameters
  - $(p_0, \dots, p_n) \rightarrow n + 1$  control points
  - $(t_0, \dots, t_{n+4}) \rightarrow n + 5$  knot values
  - $d \rightarrow$  Degree (we'll always use 3)



## Cox-deBoor recursion

- This function defines a spline curve
  - Not limited to degree 3, either...
  - With degree 3, gives the curves we used before

$$B_{k,0} = \begin{cases} 1, & \text{if } u_k \leq u < u_{k+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{k,d} = \frac{u - u_k}{u_{k+d} - u_k} B_{k,d-1}(u) + \frac{u_{k+d+1} - u}{u_{k+d+1} - u_{k+1}} B_{k+1,d-1}(u)$$

$$p(u) = \sum_{i=0}^n B_{i,d}(u) p_i$$

## Nonuniform B-Splines

- When we move to Cox-deBoor, it is no longer mandatory that the knots be evenly spaced!
  - If we move knots closer together, we increase the influence of some control points
  - Forcing duplicate knots can cause points to interpolate
  - Most common example:
    - Knots = {0, 0, 0, 0, 1, 1, 1, 1}
    - This is equivalent to a Bezier curve

## Non-uniform B-Spline common usages

- $n=3$ , knots = (0, 0, 0, 0, 1, 1, 1, 1)
  - Bezier curve
- Arbitrary  $n$ ,
  - knots = (0, 0, 0, 0, 1, 2, ...,  $n-3$ ,  $n-2$ ,  $n-2$ ,  $n-2$ ,  $n-2$ )
  - Example: (0, 0, 0, 0, 1, 2, 3, 3, 3, 3)
    - $n = 5$ , 6 control points....
  - These are called “open splines”
    - They interpolate the end points

## Non-uniform B-Spline advantages

- Very powerful
  - Can make not only polynomial curves, but also circles, ellipses, etc.
- Easily extended to surfaces

$$p(u, v) = \sum_{i=0}^n \sum_{j=0}^m B_{i,d}(u) B_{j,d}(v) p_{i,j}$$

# NURBS

- Non-Uniform Rational B-Splines
- Basically, Non-Uniform B-Splines, with one additional twist
  - If we consider  $p_i$  as  $[x, y, z, 1]$ , the result is invariant under perspective projection

$$p(u) = \frac{\sum_{i=0}^n B_{i,d}(u) p_i}{\sum_{i=0}^n B_{i,d}(u)}$$

← After projection

Note the division.  
This is the  
“rational” part.

## Another idea: Model surface, not edge

- View image intensity as a surface in 3D
- Model it with a planar surface
- Decide that there is an edge when slope of planar surface is large.

## Regularization

- Approximate intensity function by a smooth function
- Smooth function is nice because it is differentiable

## GED: Generalized Edge Detector Gokmen and Jain

- Approximate an image by a smooth function  $f$  by minimizing:

$$E_m(f) = \iint (f(x, y) - d(x, y))^2 dx dy + \lambda \underbrace{\iint (f_x^2 + f_y^2) dx dy}_{\text{smoothness term}}$$

Membrane solution

$$E_p(f) = \iint (f(x, y) - d(x, y))^2 dx dy + \lambda \iint (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2) dx dy$$

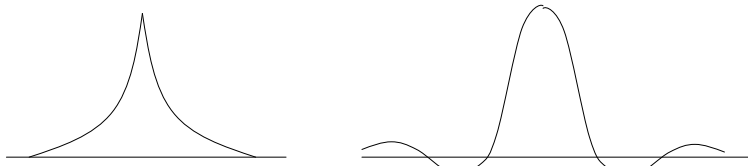
plate solution

## GED

- It is shown that this is equivalent to filtering with:

$$R_1(x, \lambda) = \frac{1}{2\lambda} e^{-|x|/\lambda} \quad \text{for the membrane}$$

$$R_2(x, \lambda) = \frac{1}{2\lambda^{1/4}} e^{-|x|/\sqrt{2}\lambda^{1/4}} \cos\left(\frac{|x|}{\sqrt{2}\lambda^{1/4}} - \frac{\pi}{4}\right) \quad \text{for the plate}$$



This looks like a Gaussian  
except for the tails

## GED

- Approximate  $d(x,y)$  by a smooth function  $f(x,y)$
- Then, find the directional derivative in the direction of the tangent contour
- Use two thresholds to eliminate broken edge segments

## Assignment 2

- Due: in 2 weeks: Nov 22
- 1. Apply Canny Edge detector to the given image
- 2. Do K-means line fitting by incrementing K.
  - Parameters: residual error threshold
  - How to initialize the K-means
    - Random
    - Uniform
    - heuristics