Computer Aided Design
Outline

- Interpolation and polynomial approximation
  - Interpolation
    - Lagrange
    - Cubic Splines
  - Approximation
    - Bézier curves
    - B-Splines
A some vocabulary (again ;)

- Control point: Geometric point that serves as support to the curve
- Knot: a specific value of the parameter $u$ corresponding to a joint between pieces of a curve
- Knot sequence: the set of knots values (in an increasing order).
- **Interpolation**
  - The curve passes through the control points

- **Approximation**
  - The curve doesn't necessarily pass through the control points
    - But these have an influence ...

- **Statistic approaches?**
  - Least squares
  - « Kriging »
  - Not very adapted to geometric modelling
Computer Aided Design

Interpolation
Interpolation

- We want to draw a regular and smooth parametric curve through a certain number $n$ of points $P_i$
  - Several families of base functions are available
  - Most obvious are polynomials
  - There are others -
    - Trigonometric functions (by mean of a Fourier decomposition for instance)
    - Power functions
    - etc...
We must choose the parametrization (nodal sequence)

- Uniform
Computer Aided Design
Interpolation

- Or non-uniform

\[ u_0 = 0 \]
\[ u_1 = 2.5 \]
\[ u_2 = 7 \]
\[ u_{n-1} = 15 \]
Can we choose $u$ as a curvilinear abscissa?

- In principle no since we don't know the final shape of the curve beforehand (with the exception of interpolation points).
- We will see later that it is often impossible that $u$ corresponds with $s$ exactly along the whole curve using analytical functions.
- But nothing forbids to get close to that – numerically...
Parametrization as an approximate arc length

\[ u_2 = d_1 + d_2 \]
\[ u_1 = d_1 \]
\[ u_0 = 0 \]
\[ d_n^{-1} u_{n-1} = d_1 + \ldots + d_{n-1} \]
In all cases, we want to have:

\[ P(u_i) = P_i \equiv \begin{cases} 
  x(u_i) = x_i \\
  y(u_i) = y_i 
\end{cases} \]

We are going to interpolate the functions \( x(u) \) and \( y(u) \) with ONE polynomial with \( n \) parameters

- This one must be of order \( p = n-1 \):

  \[ x(u) = a_0 + a_1 u + a_2 u^2 + \cdots + a_{n-1} u^{n-1} = \sum_{j=0}^{n-1} a_j u^j \]

- We set the linear system and solve...

\[
\begin{align*}
  x(u_1) &= x_1 \\
  &\vdots \\
  x(u_{n-1}) &= x_{n-1}
\end{align*}
\]
Computer Aided Design

Interpolation

- Vandermonde matrix

\[
\begin{align*}
\begin{pmatrix}
1 & u_0 & u_0^2 & \cdots & u_0^{n-1} \\
1 & u_1 & u_1^2 & \cdots & u_1^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & u_{n-1} & u_{n-1}^2 & \cdots & u_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1}
\end{pmatrix} =
\begin{pmatrix}
x_0 \\
x_1 \\
\vdots \\
x_{n-1}
\end{pmatrix}
\end{align*}
\]

- Can be solved by classical numerical methods, but ...

  - The condition number of this matrix is **VERY** bad
  - It must be solved for each RHS member \((x_i)\) or \((y_i)\), or have to take the inverse of this matrix, or perform an LU decomposition.

Instead of setting the polynomial in $x$ and in $y$ and solving, we can put it under the following form:

\[
\begin{align*}
  x(u) &= \sum_{i=0}^{n-1} x_i l_i^p(u) \\
y(u) &= \sum_{i=0}^{n-1} y_i l_i^p(u)
\end{align*}
\]

\(\iff P(u) = \sum_{i=0}^{n-1} P_i l_i^p(u)\)

where the $l_i^p(u)$ are a polynomial basis of order $p=n-1$.

These polynomials verify, for an interpolation:

\(l_i^p(u_j) = \delta_{ij}\)

We have only one computation to do for any position of the interpolation points, knowing the $u_i$ (the parametrization).
Let's determine the \( l_i(u) \) : they are of order \( n-1 \) and such that:

\[
\begin{align*}
    l_i^p(u_j) &= \delta_{ij} \\
    l_i^p(u_j) &= 1; i = j \\
    l_i^p(u_j) &= 0; i \neq j
\end{align*}
\]

\( l_i^p \) may be factored by \((u-u_j)\) for \( j \neq i \)

\[
l_i^p(u) = k(u-u_0)(u-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u-u_{n-1})
\]

\( k \) is such that \( l_i^p(u_i) = 1 \)

\[
k = ((u_i-u_0)(u_i-u_1)\cdots(u_i-u_{i-1})(u_i-u_{i+1})\cdots(u_i-u_{n-1}))^{-1}
\]

\[
l_i^p(u) = \prod_{j=0, j \neq i}^{n-1} \frac{(u-u_j)}{(u_i-u_j)}
\]

The \( l_i^p \) are the Lagrange polynomials of order \( p \)
Lagrange polynomials

\[ l_i^p(u) = \frac{l^p(u)}{l^p'(u_i)(u-u_i)} = \prod_{j=0, i \neq j}^{n-1} \frac{(u-u_j)}{(u_i-u_j)} \]

\[ l^p(u) = \prod_{j=0}^{n-1} (u-u_j) \]

\[ l^p'(u_i) = \prod_{j=0, j \neq i}^{n-1} (u_i-u_j) \] (first derivative w/r to \( u \) at \( u = u_i \))

Properties

Interpolation \( l_i^p(u_j) = \delta_{ij} \)

Partition of unity \( \sum_{0}^{n-1} l_i^p(u) = 1 \)

\[ x(u) = \sum_{0}^{n-1} x_i l_i^p(u) \] with \( x_i = 1 \forall i \)
Lagrange polynomials

- Order 4
- The \( u_i \) are evenly distributed between \( u=0 \) and \( u=1 \)
- The sum is equal to 1 (partition of unity)
- Presence of negative values
- Order 10
- Presence of huge overshoots near the boundaries
The interpolation is represented as the following form:

\[ P(u) = \sum_{i=0}^{n-1} P_i l_i^p(u) \quad \text{avec} \quad l_i^p(u) = \prod_{j=0, i \neq j}^{n-1} \frac{(u-u_i)}{(u_j-u_i)} \]

Two things worth noting:

- The curve depends **linearly** on the position of the points
- It is formed by a weighted sum of **basis functions** that express the influence of each point on the curve
An experiment

- We approximate a circle by an increasing number of points
- Simultaneously, approximation order increases
- In every case, the curve is $C_\infty$
Computer Aided Design
Lagrange polynomials

3 points, order 2 (a parabola !)
Computer Aided Design
Lagrange polynomials

5 points, order 4
11 points, order 10
Computer Aided Design
Lagrange polynomials

21 points, order 20
Does it work?

The points were set exactly on the circle
  - What occurs if their position is inaccurate?
  - Or if the approximated shape is not so simple?

We are going to see two cases
  - The coordinates of the points are perturbed randomly
  - A deterministic increase and decrease of the radius
Random perturbation

- Each point is moved radially by a value between -0.5 and +0.5 % of the circle's radius
Computer Aided Design
Lagrange polynomials

3 points, random perturbation 1%
Computer Aided Design
Lagrange polynomials

5 points, random perturbation 1%
Computer Aided Design
Lagrange polynomials

11 points, random perturbation 1%
Computer Aided Design
Lagrange polynomials

21 points, random perturbation 1%
- Runge phenomenon
  - Similar to Gibbs phenomenon of the decompositions in harmonic functions
Deterministic perturbation

- Each point is moved radially by a value of -5 or +5 % of the circle's radius
Computer Aided Design
Lagrange polynomials

3 points, deterministic perturbation 10%
Computer Aided Design
Lagrange polynomials

5 points, deterministic perturbation 10%
Computer Aided Design
Lagrange polynomials

11 points, deterministic perturbation 10%
Computer Aided Design
Lagrange polynomials

21 points, deterministic perturbation 10%
How to minimize Runge's phenomenon?

- The problem here is the use of a unique polynomial and regular intervals between knots.
- Instead, if we concentrate knots at the extremities, the interpolation is less prone to Runge's phenomenon.
- Make use of Chebyshev knots:

\[ u_i^n = \cos \left( \frac{2i - 1}{2n} \pi \right) \text{ in the interval } [-1,1] \]

or

\[ u_i^n = \frac{1}{2} + \frac{1}{2} \cos \left( \frac{2i - 1}{2n} \pi \right) \text{ in the interval } [0,1] \]
21 points, random perturbation 1%

Uniform nodal intervals – Lagrange interpolation
Computer Aided Design

Lagrange polynomials

21 points, random perturbation 1%

Using Chebychev knots
Perturbation of a point
- We shift one point by a big displacement
Computer Aided Design
Lagrange polynomials

11 points, using Chebychev knots
11 points, using Chebychev knots
Computer Aided Design
Lagrange polynomials

99 points, using Chebychev knots
499 points, using Chebychev knots
Even without perturbations, we still do numerical approximations

- Up to how many points can we pass through?
Computer Aided Design
Lagrange polynomials

51 points (lagrange polynomial of order 50)
61 points (lagrange polynomial of order 60)
Here it has nothing to do with Runge's phenomenon. It's rather an accumulation of errors due to arithmetic calculations done in finite precision.

In our case, near the boundaries, we do an inexact sum of very big numbers in absolute value, but whose sum (the exact sum) is close to the unit.

\[
P(u) = \sum_{i=0}^{n-1} P_i l_i^p(u) \quad \text{with} \quad l_i^p(u) = \prod_{j=0, i \neq j}^{n-1} \frac{(u-u_i)}{(u_j-u_i)}
\]

Big numbers in absolute value, computed with a good relative accuracy. Sum increasing the relative error.
Computer Aided Design
Lagrange polynomials

- How to evaluate a polynomial?
  - Under its Lagrange form (here): robust except at the boundaries
    - Alternative: Newton's formula and divided differences
  - Horner’s scheme
    \[ P(u) = \sum_{i=0}^{n-1} \alpha_i u^i \leftrightarrow P(u) = (a_0 + u(a_1 + u(a_2 + \ldots + u a_{n-1})\ldots)) \]
    - Optimal method with respect to the number of arithmetic operations
      for a polynomial expressed as a sum of monomials – but sometimes numerically unstable
  - Others techniques adapted to particular forms of polynomials later in the course e.g.
    - Algorithm of De Casteljau for Bernstein polynomials
    - Algorithm of De Boor for the evaluation of Bsplines
  - In every case, one should take care of to numerical errors!
Numerical errors

- Floating point calculations are (most of the time) inaccurate
- Analysis of the rounding done during operations in floating point:

\[
x \ominus y = (x - y)(1 + \delta_1), \quad |\delta_1| \leq 2\varepsilon
\]
\[
x \oplus y = (x + y)(1 + \delta_2), \quad |\delta_2| \leq 2\varepsilon
\]
\[
x \otimes y = (xy)(1 + \delta_3), \quad |\delta_3| \leq \varepsilon
\]

- Mathematically equivalent calculations but expressed differently give distinct results
  - An example with \( x^2 - y^2 = (x + y)(x - y) \)
Error made with the expression \((x + y)(x - y)\)
\[(x \oplus y) \otimes (x \ominus y) = (x - y)(1 + \delta_1)(x + y)(1 + \delta_2)(1 + \delta_3)\]
\[= (x + y)(x - y)(1 + \delta_1 + \delta_2 + \delta_3 + \delta_1 \delta_2 + \delta_2 \delta_3 + \delta_1 \delta_3 + \delta_1 \delta_2 \delta_3)\]
\[\approx 5 \varepsilon\]

No catastrophic increase of the relative error

Error made with the expression \(x^2 - y^2\)
\[(x \otimes x) \Theta (y \otimes y) = [x^2(1 + \delta_1) - y^2(1 + \delta_2)](1 + \delta_3)\]
\[= ((x^2 - y^2)(1 + \delta_1) + (\delta_1 - \delta_2)y^2)(1 + \delta_3)\]
\[= ((x^2 - y^2)(1 + \delta_1 + \delta_3 + (\delta_1 - \delta_2)y^2 + \delta_1 \delta_3 + (\delta_1 - \delta_2)y^2 \delta_3))\]

When \(x\) is close to \(y\), the error can be of the order of magnitude of the calculated result...
Some useful rules (not a comprehensive list!)

- Prefer \((x + y)(x - y)\) to \(x^2 - y^2\)
  - Lagrange form more accurate than Horner's scheme ...
- E. g. sum of many terms
  - Naive algorithm:
    
    ```
    S=0;
    for (j=1;j<=N;j++) { S=S+X[j] ; } 
    return S ; 
    ```
  
  involves an error \( \approx N \varepsilon \)
  - Kahan's summation algorithm:
    
    ```
    S=X[1]; C=0
    for (j=2;j<=N;j++)
      { Y=X[j]-C; T=S+Y; C=(T-S)-Y; S=T } 
    return S ; 
    ```
  
  involves an error \( \approx 2\varepsilon \)
Example of catastrophic rounding

Computation of an integral:

\[ S = \int_{\Omega} f(x, y) \, dx \, dy \quad \text{with} \quad f(x, y) = x^2 + y^2 \]

\[ S \approx \sum_{i=0}^{nx-1} \sum_{j=0}^{ny-1} f(x(i), y(j)) \det J \]

\[
\begin{align*}
\Delta x & = \frac{x_{\text{max}} - x_{\text{min}}}{nx} \\
\Delta y & = \frac{y_{\text{max}} - y_{\text{min}}}{ny} \\
x(i) & = x_{\text{min}} + i \Delta x + \Delta x / 2 \\
y(j) & = y_{\text{min}} + j \Delta y + \Delta y / 2 \\
\det J & = \Delta x \, \Delta y
\end{align*}
\]
Computer Aided Design
Numerical errors

- Computations made with the following parameters:

\[ x_{\text{min}} = y_{\text{min}} = 0.0 \; ; \; x_{\text{max}} = y_{\text{max}} = 1.0 \; ; \; n_x = n_y = 10, \quad S_{\text{exact}} = \frac{2}{3} \]

1) Single precision floating point numbers
2) Double precision floating point numbers
3) Quad precision floating point numbers
4) Single precision floating points numbers with Kahan's summation algorithm

bechet@yakusa:~ floating_error$ ./test 10

1) sum (float ) = 0.66500002145767211914
2) sum (double ) = 0.665000000000000025757
3) sum (ldouble) = 0.66499999999999999997
4) sum (kahan ) = 0.66499999999999999997

Note: Program should be compiled without optimization!
Numerical errors

bechet@yakusa:floating_error$ ./test 100
1) sum (float )=0.66664981842041015625
2) sum (double )=0.66665000000000051994
3) sum (ldouble)=0.6666500000000000093
4) sum (kahan   )=0.66664993762969970703

bechet@yakusa:floating_error$ ./test 1000
1) sum (float )=0.66668075323104858398
2) sum (double )=0.666666499999999805232
3) sum (ldouble)=0.66666649999999999982
4) sum (kahan   )=0.66666668653488159180
bechet@yakusa:~ $ ./test 10000
1) sum (float ) = 0.36836880445480346680
2) sum (double ) = 0.66666666499985449690
3) sum (ldouble) = 0.66666666499999997623
4) sum (kahan  ) = 0.66666656732559204102

bechet@yakusa:~ $ ./test 100000
1) sum (float ) = 0.00390625000000000000
2) sum (double ) = 0.66666666665538221181
3) sum (ldouble) = 0.66666666665000055245
4) sum (kahan  ) = 0.66666666269302368164

Compensated summation algorithm coming from:

$y = p(x) = (1 - x)^n$
for $x = 1.333$
and $2 < n < 41$
Definition of the condition number of a numerical expression

- Ratio of the direct error to the inverse error

For a polynomial under monomial form:

\[
K(P, x) = \lim_{\varepsilon \to 0} \sup_{\delta x \in D(\varepsilon)} \left( \frac{\|\delta y\|}{\|\delta x\|} \right)
\]

\[
K(P, x) = \frac{\sum_{i=0}^{n} |a_i| |x|^i}{\left| \sum_{i=0}^{n} a_i x^i \right|}
\]
There are various compensated algorithms to carry out calculations on floating point numbers.

- Ex. Kahan summation, Compensated Horner scheme ...

In general, they allow to have similar results as when using internal floating point with a precision twice that of the input data, following by a final rounding.

See references available on the course's website for the compensated Horner scheme.
Computer Aided Design
Lagrange polynomials

- Another experiment...
Evaluation of a Lagrange interpolation directly based on the Vandermonde matrix

\[
\begin{pmatrix}
1 & u_0 & u_0^2 & \cdots & u_0^{n-1} \\
1 & u_1 & u_1^2 & \cdots & u_1^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & u_{n-1} & u_{n-1}^2 & \cdots & u_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1}
\end{pmatrix}
=
\begin{pmatrix}
x_0 \\
x_1 \\
\vdots \\
x_{n-1}
\end{pmatrix}
\]

\[
P(u) = \sum_{i=0}^{n-1} \alpha_i u^i \iff P(u) = \left( a_0 + u(a_1 + u(a_2 + \ldots + u(a_{n-1} \ldots))) \right)
\]
n=25;
minu=0.00;
maxu=1.00;

U=linspace(minu,maxu,n)';  % vector [minu,minu+(maxu-minu)/(n-1),... maxu]

m=zeros(n);
for i=1:n
    for j=1:n
        m(i,j)=U(i)^(n-j);
    endfor
endfor

x=zeros(n,1);
y=zeros(n,1);
for i=1:n
    x(i)=cos(((i-1)/(n-1))*pi);
    y(i)=sin(((i-1)/(n-1))*pi);
endfor

mm=m^-1;
ax=mm*x;
ay=mm*y;

filename = "myfile.txt";
fid = fopen (filename, "w");

nbp=10000;

for k=0:nbp
    u=minu+(maxu-minu)*k/nbp;
    resx=0;
    resy=0;
    for i=1:n
        resx=u*resx+ax(i);
        resy=u*resy+ay(i);
    endfor
    fprintf(fid,"%e %e %e \n",u,resx,resy);
endfor

fclose (fid);
Computer Aided Design
Lagrange polynomials

\[ u = 0 \]

\[ u = 1 \]
n=25;
minu=-1.0;
maxu=0.0;

U=linspace(minu,maxu,n)';  \% vecteur [minu,minu+(maxu-minu)/(n-1),... maxu]

m=zeros(n);
for i=1:n
    for j=1:n
        m(i,j)=U(i)^-(n-j);
    endfor
endfor

x=zeros(n,1);
y=zeros(n,1);
for i=1:n
    x(i)=cos(((i-1)/(n-1))*pi);
    y(i)=sin(((i-1)/(n-1))*pi);
endfor

mm=m^-1;
ax=mm*x;
ay=mm*y;

filename = "myfile.txt";
fid = fopen (filename, "w");

nbp=1000;

for k=0:nbp
    u=minu+(maxu-minu)*k/nbp;
    resx=0;
    resy=0;
    for i=1:n
        resx=u*resx+ax(i);
        resy=u*resy+ay(i);
    endfor
    fprintf(fid,%e %e %e \n",u,resx,resy);
endfor

fclose (fid);
Computer Aided Design
Lagrange polynomials

\[ u = 0 \]

\[ u = -1 \]
• The evaluation by the Vandermonde matrix is relatively stable in the vicinity of \( u = 0 \)
  • By change of parameter: we move the area of stability.
• As soon as we go away from \( u = 0 \), numerical errors become predominant.
Morality:

- Lagrange interpolation is not suited beyond 10's of control points because of the Runge phenomenon.
- A modification of the position of a control point leads to global changes of the curve.
- The evaluation of high order polynomials expressed as monomials leads to numerical problems.
- No control of the slopes at the boundary of the curve (start and finish).
Motivation
Let us imagine that we have many (100's of) control points

- But we don't want a Lagrange interpolation!
- We should stay with a low order scheme but conserve enough freedom to pass through every point
  - Curve defined by pieces ... and of low order (1)
We are going to build a low order interpolation for each knot interval, such that we can impose slopes at the knots.
In each range \([i, i+1]\), we want to have an independent polynomial.

We have 4 parameters: position at each knot and associated tangents.

The basis must have 4 degrees of freedom, thus be of order 3 in the case of polynomials.

\[
P(u_i) = P_i \equiv \begin{cases} 
  x(u_i) = x_i \\
  y(u_i) = y_i
\end{cases}
\]

\[
x_{[i]}(u) = A_{[i]0} + A_{[i]1} u + A_{[i]2} u^2 + A_{[i]3} u^3, \quad u \in [u_i, u_{i+1}]
\]
First, every interval has a unit length i.e.
\[ u_{i+1} - u_i = 1 \]

Then we ensure identical intervals \([0...1]\) between each interpolation point:
\[
\bar{u} = \frac{u - u_i}{u_{i+1} - u_i} = u - u_i \quad \frac{d \bar{u}}{du} = 1
\]

On each interval \( i \), we thus have the following relation:
\[
x_{[i]}(\bar{u}) = a_{[i]0} + a_{[i]1} \bar{u} + a_{[i]2} \bar{u}^2 + a_{[i]3} \bar{u}^3 \quad , \quad \bar{u} \in [0,1]
\]
Computer Aided Design
Splines

\[ P = P_{i+1} \]

\[ P' = P'_{i+1} \]

\[ \bar{u} = 1 \]

\[ P' = P'_i \]

\[ \bar{u} = 0 \]
We pass through both control points:

\[ P(\bar{u}_0 = 0) = P_i \iff a_{[i]0} + a_{[i]1} \bar{u}_0 + a_{[i]2} \bar{u}_0^2 + a_{[i]3} \bar{u}_0^3 = x_i \]

\[ P(\bar{u}_1 = 1) = P_{i+1} \iff a_{[i]0} + a_{[i]1} \bar{u}_1 + a_{[i]2} \bar{u}_1^2 + a_{[i]3} \bar{u}_1^3 = x_{i+1} \]

We impose both slopes:

\[ P'(\bar{u}_0 = 0) = P'_i \iff a_{[i]1} + 2a_{[i]2} \bar{u}_0 + 3a_{[i]3} \bar{u}_0^2 = x'_i \]

\[ P'(\bar{u}_1 = 1) = P'_{i+1} \iff a_{[i]1} + 2a_{[i]2} \bar{u}_1 + 3a_{[i]3} \bar{u}_1^2 = x'_{i+1} \]

At the end:

\[
\begin{align*}
    a_{[i]0} &= x_i \\
    a_{[i]1} &= x'_i \\
    a_{[i]2} &= 3(x_{i+1} - x_i) - 2x'_i - x'_{i+1} \\
    a_{[i]3} &= 2(x_i - x_{i+1}) + x'_i + x'_{i+1}
\end{align*}
\]
- We have continuity
- We have continuity of the derivatives
- But how to choose the slopes?
  - Let the user choose ("artistic" freedom)
  - Automatically ...
By finite differences with three points:

\[ x_i' = \frac{x_{i+1} - x_i}{2(u_{i+1} - u_i)} + \frac{x_i - x_{i-1}}{2(u_i - u_{i-1})} \]

- At the boundaries, we use finite differences (asymmetric)

\[ x_0' = \frac{x_1 - x_0}{u_1 - u_0} \quad x_{n-1}' = \frac{x_{n-1} - x_{n-2}}{u_{n-1} - u_{n-2}} \]

- The result depends on the parametrization!

- Cardinal spline

\[ x_i' = (1 - c) \frac{x_{i+1} - x_{i-1}}{2}, \quad 0 \leq c \leq 1 \]

\[ x_0' = (1 - c)(x_1 - x_0) \quad x_{n-1}' = (1 - c)(x_{n-1} - x_{n-2}) \]

- \( c \) is a « tension » parameter. \( c=0 \) gives yields the so called “Catmull-Rom” spline, \( c=1 \) a zigzagging line.
Continuity of the curve/parameter but loss of regularity (and of geometric continuity in many cases)

5 points, finite differences by varying the parametrization $[0..1], [0..2], [0..5], [0..10]$
5 points, Cardinal Spline (Catmull-Rom) \( c=0 \)
Catmull-Rom Splines are widely used in computer graphics

- Simple to compute, effective
- Local control (price to pay: discontinuous second derivative)
- Animations with keyframing
  - Ensures a fluid motion because of the continuity of the slope
5 points, Cardinal Spline $c=0.25$
Computer Aided Design
Splines

5 points, Cardinal Spline $c=0.5$
Computer Aided Design
Splines

5 points, Cardinal Spline $c=0.75$
Computer Aided Design
Splines

5 points, Cardinal Spline $c=1.0$
We can impose the continuity of second derivatives...

- On a curve with $n$ points, we have $n$ extra relations to impose.
- We may impose the continuity of the second derivative only on the $n-2$ interior knots.

What about the 2 points on the boundary?

- Impose a vanishing second derivative. We obtain what is called « natural spline ».
- We could also impose the slopes (i.e. only $n-2$ relations remaining).
- Or, impose that the third derivative is zero on the points 1 and $n-2$.
  - That means a single polynomial expression for the first two knot intervals, and the last two.
Natural Spline: mathematical approximation of the spline historically used in naval construction.
Computer Aided Design
Splines
We impose the continuity of the second derivatives

\[ x''_{[i-1]}(1) = x''_{[i]}(0) \Leftrightarrow 2a_{[i-1]2} + 6a_{[i-1]3} = 2a_{[i]2} \]

We substitute in the “internal” equations

\[
2 \left[ 3(x_i - x_{i-1}) - 2x'_{i-1} - x_i \right] + 6 \left[ 2(x_{i-1} - x_i) + x'_{i-1} + x_i \right] \\
= 2 \left[ 3(x_{i+1} - x_i) - 2x_i - x'_{i+1} \right]
\]

Finally we obtain:

\[ x'_{i-1} + 4x_i + x'_{i+1} = 3(x_{i+1} - x_{i-1}) \]
At the boundaries we want

\[ x''_0(0) = 0 \Leftrightarrow 2a_{[0]_2} = 0 \]
\[ x''_{n-2}(1) = 0 \Leftrightarrow 2a_{[n-2]_2} + 6a_{[n-2]_3} = 0 \]

We have then a linear system with \( n \) unknowns:

\[
\begin{pmatrix}
2 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
\vdots \\
1 & 4 & 1 \\
1 & 2 \\
\end{pmatrix}
\begin{pmatrix}
x'_0 \\
x'_1 \\
x'_2 \\
\vdots \\
x'_{n-2} \\
x'_{n-1} \\
\end{pmatrix}
\begin{pmatrix}
3(x_1-x_0) \\
3(x_2-x_0) \\
3(x_3-x_1) \\
\vdots \\
3(x_{n-1}-x_{n-3}) \\
3(x_{n-1}-x_{n-2}) \\
\end{pmatrix}
\]
By solving the system, we have:

\[
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  \vdots \\
  x_{n-2} \\
  x_{n-1}
\end{bmatrix}
\]

, which is substituted in

\[
\begin{align*}
a_{[i]0} &= x_i \\
a_{[i]1} &= x_i \\
a_{[i]2} &= 3(x_{i+1} - x_i) - 2x_i' - x_{i+1}' \\
a_{[i]3} &= 2(x_i - x_{i+1}) + x_i' + x_{i+1}'
\end{align*}
\]

, to get the polynomial in each portion:

\[
x_{[i]}(\bar{u}) = a_{[i]0} + a_{[i]1} \bar{u} + a_{[i]2} \bar{u}^2 + a_{[i]3} \bar{u}^3, \quad 0 \leq \bar{u} < 1
\]

From the global parameter \( u \), we have to find in which portion we are (the value of \( i \)), then compute right polynomial...
Computer Aided Design
Splines

Catmull-Rom spline

5 control points, natural spline
Another experiment

- We approximate a circle by a number of increasing points.
- Simultaneously, the order of the approximation--the number of pieces increases.
- In all the cases, the curve is $C_\infty C_2$. 
Computer Aided Design
Splines

3 points, order 3!
Computer Aided Design
Splines

5 points
Computer Aided Design
Splines

11 points
Computer Aided Design
Splines

21 points
Random perturbation

- Each point is moved radially by a value between -0.5 and +0.5 % of the circle's radius.
Computer Aided Design
Splines

3 points, random perturbation 1%
Computer Aided Design
Splines

5 points, random perturbation 1%
Computer Aided Design
Splines

11 points, random perturbation 1%
21 points, random perturbation 1%
Deterministic perturbation

Each point is shifted radially depending on its position by -5 or +5 % of the circle's radius
Computer Aided Design
Splines

3 points, deterministic perturbation 5%
Computer Aided Design
Splines

5 points, deterministic perturbation 5%
Computer Aided Design
Splines

11 points, deterministic perturbation 5%
Computer Aided Design
Splines

21 points, deterministic perturbation 5%
Computer Aided Design

Splines

11 points

non local control
Computer Aided Design
Splines
Perturbation of a point

- We shift one point by a significant amount
Computer Aided Design
Splines

21 points
Computer Aided Design

Splines

99 points
Computer Aided Design
Splines

999 points
Stable interpolation scheme
Weak Runge phenomenon
The displacement of a point yet affects all the curve

- Nevertheless, the perturbation fades very quickly further away from the shifted point
- « Overshoots » are limited.
Closed curve?

The curve can be closed, just impose everywhere that the second derivative is continuous.
Instead of

\[ x''_0(0) = 0 \iff 2a_{[0]2} = 0 \]
\[ x''_{[n-2]}(1) = 0 \iff 2a_{[n-2]2} + 6a_{[n-2]3} = 0 \]

... we have

\[ x''_{[n-2]}(1) = x''_0(0) \iff 2a_{[n-2]2} + 6a_{[n-2]3} = 2a_{[0]2} \]

\[
\begin{vmatrix}
4 & 1 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 1 \\
1 & 1 & 4 & 1 & 1 \\
& & & \ddots & \\
1 & 1 & 1 & 4 & 1 \\
1 & 1 & 1 & 1 & 4
\end{vmatrix}
\begin{pmatrix}
x_0' \\
x_1' \\
x_2' \\
\vdots \\
x_{n-3}' \\
x_{n-2}'
\end{pmatrix}
= \begin{pmatrix}
3(x_1-x_{n-2}) \\
3(x_2-x_0) \\
3(x_3-x_1) \\
\vdots \\
3(x_{n-2}-x_{n-4}) \\
3(x_0-x_{n-3})
\end{pmatrix}
\]

and

\[ x_{n-1} = x_0 \]
\[ x_{n-1} = x_0' \]
General case: arbitrary parametrization

\[ P(u_i) = P_i \equiv \begin{cases} 
  x(u_i) = x_i \\
  y(u_i) = y_i 
\end{cases} \]

\[ x_{[i]}(u) = A_{[i]}^0 + A_{[i]}^1 u + A_{[i]}^2 u^2 + A_{[i]}^3 u^3 \quad , \quad u \in [u_i, u_{i+1}] \]

We again change the parametrization...

\[ \overline{u} = \frac{u - u_i}{u_{i+1} - u_i} \quad \quad \frac{d \overline{u}}{du} = \frac{1}{u_{i+1} - u_i} = \frac{1}{h_i} \]

\[ x_{[i]}(\overline{u}) = a_{[i]}^0 + a_{[i]}^1 \overline{u} + a_{[i]}^2 \overline{u}^2 + a_{[i]}^3 \overline{u}^3 \quad , \quad \overline{u} \in [0,1] \]
We pass through both points:

\[ P(\bar{u}_0 = 0) = P_i \iff a_{[i]0} + a_{[i]1} \bar{u}_0 + a_{[i]2} \bar{u}_0^2 + a_{[i]3} \bar{u}_0^3 = x_i \]

\[ P(\bar{u}_1 = 1) = P_{i+1} \iff a_{[i]0} + a_{[i]1} \bar{u}_1 + a_{[i]2} \bar{u}_1^2 + a_{[i]3} \bar{u}_1^3 = x_{i+1} \]

We impose both slopes:

\[ P'(\bar{u}_0 = 0) = \frac{d P}{d \bar{u}}(0) = \frac{d P}{d \bar{u}}(0) \frac{1}{h_i} = P'_i \iff a_{[i]1} + 2a_{[i]2} \bar{u}_0 + 3a_{[i]3} \bar{u}_0^2 = x'_i h_i \]

\[ P'(\bar{u}_1 = 1) = \frac{d P}{d \bar{u}}(1) = \frac{d P}{d \bar{u}}(1) \frac{1}{h_i} = P'_{i+1} \iff a_{[i]1} + 2a_{[i]2} \bar{u}_1 + 3a_{[i]3} \bar{u}_1^2 = x'_{i+1} h_i \]

Finally:

\[
\begin{align*}
a_{[i]0} &= x_i \\
a_{[i]1} &= x'_i h_i \\
a_{[i]2} &= 3(x_{i+1} - x_i) - 2x'_i h_i - x'_{i+1} h_i \\
a_{[i]3} &= 2(x_i - x_{i+1}) + x'_i h_i + x'_{i+1} h_i
\end{align*}
\]
We impose the second derivative for a natural spline

\[
\frac{d^2 P[i]}{du^2}(\bar{u}) = \frac{d^2 P[i]}{d\bar{u}^2} \frac{d^2 \bar{u}}{du^2} = \frac{d^2 P[i]}{d\bar{u}^2} \frac{1}{h_i^2}
\]

\[
x''_{[i-1]}(1) = x''_i(0) \iff \frac{2a_{[i-1]2}+6a_{[i-1]3}}{h_{i-1}^2} = \frac{2a_{[i]2}}{h_i^2}
\]

We substitute in the internal equations

\[
2\left[3(x_i - x_{i-1}) - 2x_{i-1}h_{i-1} - x_i h_{i-1}\right] + 6\left[2(x_{i-1} - x_i) + x_{i-1}' h_{i-1} + x_i' h_{i-1}\right]
\]

\[
= \frac{2a_{[i-1]2}+6a_{[i-1]3}}{h_{i-1}^2} = \frac{2a_{[i]2}}{h_i^2}
\]
We obtain finally:

\[
\frac{x_{i-1} + 2x_i}{h_{i-1}} + \frac{2x_i + x_{i+1}}{h_i} = 3 \left( \frac{x_i - x_{i-1}}{h_{i-1}^2} \right) + 3 \left( \frac{x_{i+1} - x_i}{h_i^2} \right)
\]

\[
h_i (x_{i-1} + 2x_i) + h_{i-1} (2x_i + x_{i+1}) = 3 \frac{h_i}{h_{i-1}} (x_i - x_{i-1}) + 3 \frac{h_{i-1}}{h_i} (x_{i+1} - x_i)
\]
- At the boundaries we want a vanishing second derivative ...

\[ x''[0](0) = 0 \iff \frac{2a_{[0]}^2}{(h_0^2)} = 0 \]

\[ x''[n-2](1) = 0 \iff \frac{2a_{[n-2]}^2 + 6a_{[n-2]}^3}{(h_{n-2}^2)} = 0 \]

- We have then a linear system of \( n \) unknowns:

(next page)
Computer Aided Design

Splines

\[
\begin{pmatrix}
2h_0 & h_0 \\
h_1 & 2h_1+2h_0 & h_0 \\
\vdots & & \ddots \\
h_{n-2} & 2h_{n-2}+2h_{n-3} & h_{n-3} \\
h_{n-2} & & 2h_{n-2}
\end{pmatrix}
\begin{pmatrix}
\cdot \\
\cdot \\
\cdot \\
x_0 \\
x_1 \\
\vdots \\
x_{n-2} \\
x_{n-1}
\end{pmatrix}
= \\
\begin{pmatrix}
3(x_1-x_0) \\
3\frac{h_1}{h_0}(x_1-x_0)+3\frac{h_0}{h_1}(x_2-x_1) \\
\vdots \\
3\frac{h_{n-2}}{h_{n-3}}(x_{n-2}-x_{n-3})+3\frac{h_{n-3}}{h_{n-2}}(x_{n-1}-x_{n-2}) \\
3(x_{n-1}-x_{n-2})
\end{pmatrix}
\]
Computer Aided Design
Splines

Natural Spline
(uniform parametrization)

Natural Spline
(non uniform parametrization)
Compact notation

We would like a compact representation as for the Lagrange interpolation.

\[ x_{[i]}(\bar{u}) = a_{[i]0} + a_{[i]1} \bar{u} + a_{[i]2} \bar{u}^2 + a_{[i]3} \bar{u}^3, \quad 0 \leq \bar{u} < 1 \]

\[
\begin{align*}
  x_{[i]}(\bar{u}) &= \sum_{0}^{n} x_i h_{i0}^p(\bar{u}) + \sum_{0}^{n} x_i h_{i1}^p(\bar{u}) \\
  y_{[i]}(\bar{u}) &= \sum_{0}^{n} y_i h_{i0}^p(\bar{u}) + \sum_{0}^{n} y_i h_{i1}^p(\bar{u})
\end{align*}
\]

\[
\Leftrightarrow P_{[i]}(\bar{u}) = \sum_{0}^{n} P_i h_{i0}^p(\bar{u}) + \sum_{0}^{n} P_i h_{i1}^p(\bar{u})
\]

This on each interval \([i,i+1]\).
We have on each interval:

\[
\begin{align*}
a_{[i]0} &= x_i \\
a_{[i]1} &= x_i' \\
a_{[i]2} &= 3(x_{i+1} - x_i) - 2x_i' - x_{i+1}' \\
a_{[i]3} &= 2(x_i - x_{i+1}) + x_i' + x_{i+1}'
\end{align*}
\]

\[
x_{[i]}(\bar{u}) = x_i + x_i'\bar{u} + (3(x_{i+1} - x_i) - 2x_i' - x_{i+1}')\bar{u}^2 + (2(x_i - x_{i+1}) + x_i' + x_{i+1}')\bar{u}^3
\]

By rearranging equations:

\[
x_{[i]}(\bar{u}) = x_i(1 - 3\bar{u}^2 + 2\bar{u}^3) + x_i'(\bar{u} - 2\bar{u}^2 + \bar{u}^3)
\]

\[
+ x_{i+1}(3\bar{u}^2 - 2\bar{u}^3) + x_{i+1}'(-\bar{u}^2 + \bar{u}^3)
\]
Hermite polynomials (for two points)

\[
\begin{align*}
h^p_{00} &= 1 - 3\bar{u}^2 + 2\bar{u}^3 \\
h^p_{10} &= 3\bar{u}^2 - 2\bar{u}^3 \\
h^p_{01} &= \bar{u} - 2\bar{u}^2 + \bar{u}^3 \\
h^p_{11} &= -\bar{u}^2 + \bar{u}^3
\end{align*}
\]
The more general Hermite basis of degree $2n-1$ allows to do an interpolation of $n$ points on an interval, as the Lagrange basis, additionally imposing slopes at each point.

$$h_{i0}^n(u) = \left[ 1 - \frac{l''(u_i)}{l'(u_i)} (u-u_i) \right] [l_i^n(u)]^2$$

$$h_{i1}^n(u) = (u-u_i)[l_i^n(u)]^2$$

$$l_i^n(u) = \frac{l^n(u)}{l'(u_i)(u-u_i)} = \prod_{j=0, i \neq j}^{n-1} \frac{(u-u_i)}{(u_j-u_i)}$$

$$l^n(u) = \prod_{j=0}^{n-1} (u-u_j)$$
Properties of the Hermite basis

\[ h_{i0}^n(u_j) = \delta_{ij} \]
\[ (h_{i0}^n)'(u_j) = 0 \]
\[ h_{i1}^n(u_j) = 0 \]
\[ (h_{i1}^n)'(u_j) = \delta_{ij} \]

\[ \sum_i h_{i0}^n(u) = 1 \]
Property of affine invariance

It is a useful property that the curves we define for a set of control points can undergo linear affine transformations.

- Let \( P_i^* \) the affine transformation of the control points \( P_i \)
- Let \( P^*(P_i) \) the affine transformation of the points of the curve \( P(P_i) \) defined from the original points \( P_i \)
- Let \( P(P_i^*) \) the new curve based on the modified control points \( P_i^* \), with the same parametrization.

The affine invariance is verified iff \( P^*(P_i) = P(P_i^*) \) for all \( u \).
**Affine transformations**  \( \Phi(P) = A \cdot P + u \)

- **Translation**
  \[ u = \begin{bmatrix} a \\ b \\ c \end{bmatrix} ; \quad A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

- **Scaling**
  \[ u = 0 ; \quad A = \begin{bmatrix} d & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & f \end{bmatrix} \]

- **3 rotations**
  \[ u = 0 ; \quad A = \begin{bmatrix} 1 & g & h \\ 0 & 1 & i \\ 0 & 0 & 1 \end{bmatrix} \]

- **Shear**
  \[ u = 0 ; \quad A = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

- **12 degrees of freedom**
Let $P$ a parametric curve built this way:

$$P(u) = \sum_{0}^{n-1} P_{i} K_{i}^{n}(u)$$

- Let's verify the invariance by a translation $t$:

$$P(P_{i}^*) = \sum_{0}^{n-1} (P_{i} + t) K_{i}^{n}(u) = \sum_{0}^{n-1} P_{i} K_{i}^{n}(u) + \sum_{0}^{n-1} t K_{i}^{n}(u)$$

$$= P(u) + \sum_{0}^{n-1} t K_{i}^{n}(u)$$

$$= P(u) + t = P^*(P_{i}) \quad \text{iff} \quad \sum_{0}^{n-1} K_{i}^{n}(u) = 1$$

Partition of unity
For the other multiplicative transformations

\[ P(P_i^*) = \sum_{0}^{n-1} (A \cdot P_i) K_i^n(u) = A \cdot \sum_{0}^{n-1} P_i K_i^n(u) \]

\[ = A \cdot P(u) = P^*(P_i) \]

(no particular conditions except linearity with respect to the control points coordinates)

Consequently, iff the basis functions form a partition of unity, and the dependence with respect to the control points is linear, the representation is invariant by any affine transformation.
Do Hermite's basis form a partition of unity?

\[ \sum_{0}^{n-1} h_{??}^n(u) = 1 \]

\[ F(u) = 1 \]

\[ F(0) = 1 \quad F(1) = 1 \quad F'(0) = 0 \quad F'(1) = 0 \]

\[ \begin{align*}
    h_{00}^p &= 1 - 3 \bar{u}^2 + 2 \bar{u}^3 \\
    h_{10}^p &= 3 \bar{u}^2 - 2 \bar{u}^3 \\
    h_{01}^p &= \bar{u} - 2 \bar{u}^2 + \bar{u}^3 \\
    h_{11}^p &= -\bar{u}^2 + \bar{u}^3
\end{align*} \]
Let's check the invariance

- If we apply a translation to the control points $P_i$, the derivatives $P'_i$ don't change ...

\[
P^*_i = P_i + t \quad P^{'*}_i = P'_i
\]

\[
P(P^*_i) = \sum_0^1 (P_i + t) h_{i0}^n(u) + \sum_0^1 P'_i h_{i1}^n(u)
\]

\[
= \sum_0^1 t h_{i0}^n(u) + \sum_0^1 P_i h_{i0}^n(u) + \sum_0^1 P'_i h_{i1}^n(u)
\]

\[
= t + P(P_i) = P^*(P_i)
\]
We must also check the invariance for the other transformations ...

\[ P_0^* = A \cdot P_0 \]
\[ P'_0^* = A \cdot P'_0 \]

\[ P(P_0^*) = \sum_{0}^{1} (A \cdot P_0) h_{i0}^n(u) + \sum_{0}^{1} (A \cdot P'_0) h_{i1}^n(u) \]

\[ = A \cdot \left( \sum_{0}^{1} P_0 h_{i0}^n(u) + \sum_{0}^{1} P'_0 h_{i1}^n(u) \right) \]

\[ = A \cdot P(P_0) = P^*(P_0) \]

QED
Beware of the computation of the slopes $x'_i$ ...

Natural Splines:

$$
\begin{bmatrix}
2 & 1 \\
1 & 4 & 1 \\
& 1 & 4 & 1 \\
& & \ddots \\
1 & 4 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x'_0 \\
x'_1 \\
x'_2 \\
\vdots \\
x'_{n-2} \\
x'_{n-1} \\
\end{bmatrix}
= 
\begin{bmatrix}
3(x_1 - x_0) \\
3(x_2 - x_0) \\
3(x_3 - x_1) \\
\vdots \\
3(x_{n-1} - x_{n-3}) \\
3(x_{n-1} - x_{n-2}) \\
\end{bmatrix}
$$

Linear operator

$P'_i = L(P_i - P_j) \Rightarrow P'^{*} = L(P^{*} - P^{*}) = L((A \cdot P_i + t) - (A \cdot P_j + t))$

$= A \cdot L(P_i - P_j) = A \cdot P'_i$

It is OK in this case
- Rotation of 45°
- Scaling $x$ direction (times 0.5)
- Followed by a translation
3D Curves

Minimal order so that a curve can have a torsion (non planar curve)
- Let's consider a Lagrange interpolation
  \[ P(u) = \sum_{i=0}^{n-1} P_i l_i^p(u) \]
- 2 points → on a straight line (no curvature)
- 3 points → in a plane (no torsion)
- 4 points → torsion becomes possible

Minimal order to join smoothly two arbitrarily oriented curves = 3