

# ***Modellazione Grafica computazionale 2D e 3D***

*Autori:*

*Anna Lamboglia*

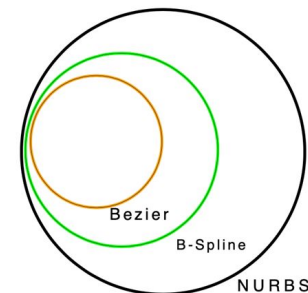
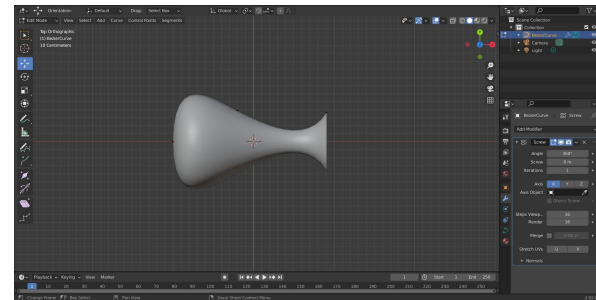
*Agostino Vitaglione*

*Mario Vitaglione*

# Introduzione

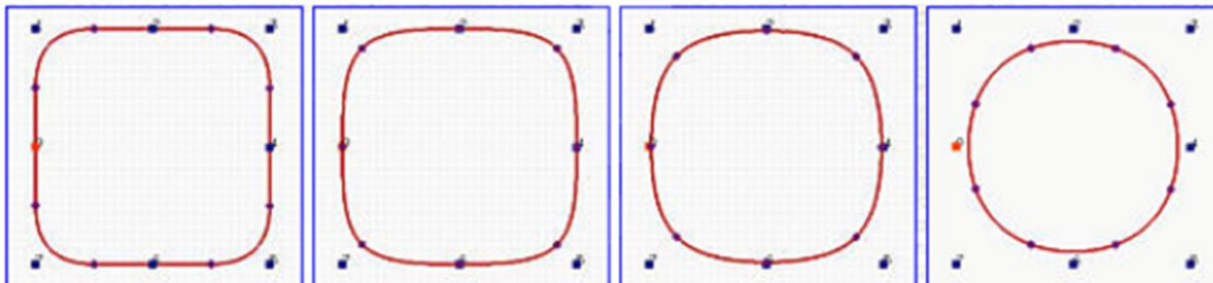
Durante il percorso effettuato sono state visti vari modi per rappresentare i dati, in particolare varie tipologie di curve, ognuna con i suoi pregi, che vengono sempre di più utilizzate nella modellazione 2D e 3D in campi come:

- Applicazioni economiche
- Animazione e videogames
- Progettazione di componenti meccanici ed elettronici
- Stampa 3D
- Programmazione di circuiti integrati e schede



# Introduzione

Essendo le curve B-Spline e di Bézier realizzate attraverso dei polinomi, non è possibile rappresentare senza errori le curve goniometriche, anche utilizzando gli sviluppi in serie di Taylor poiché l'ambiente di calcolo è un ambiente finito.



Le 4 immagini rappresentano delle curve B-Spline di grado diverso con 8 punti di controllo.

Da sinistra verso destra i gradi sono 2, 3, 5, 10.

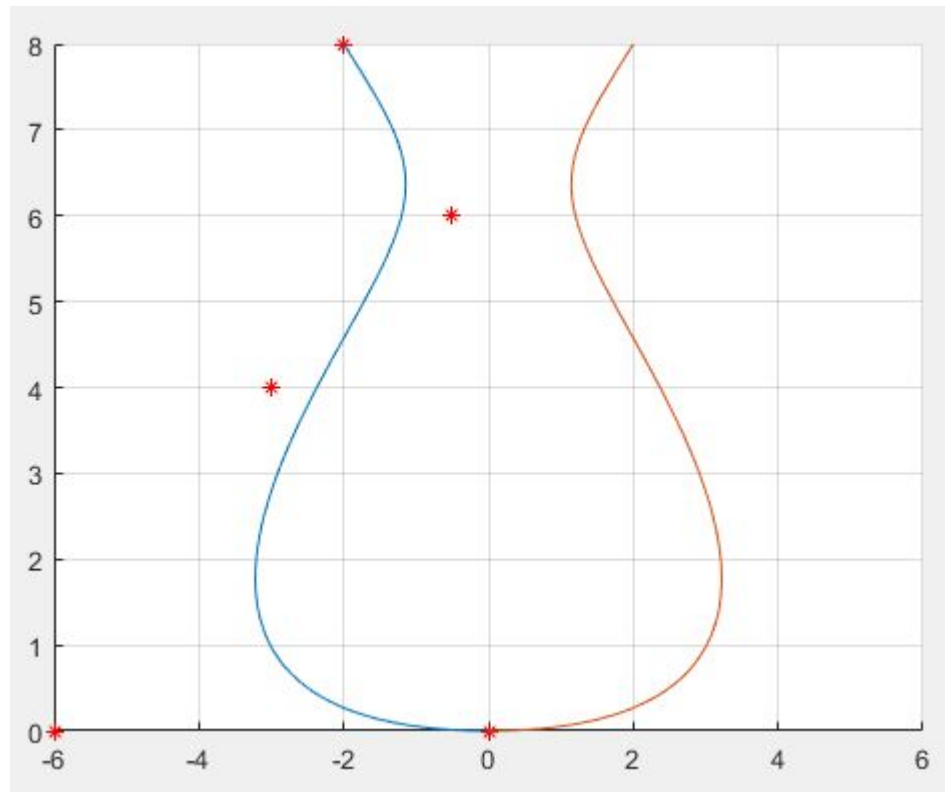
Notiamo che dobbiamo arrivare al grado 10 per avere un risultato simile ad una circonferenza.

Ciò risulta inaccettabile!

# Esempio Curva di Bézier

## Curva di Bézier con i polinomi di Bernstein

```
1 - clear all; close all; clc;
2
3 - P = [0 0; -6 0; -3 4; -0.5 6; -0.5 6; -2 8];
4
5 - syms t
6 - B = bernsteinMatrix(length(P(:,1)) - 1, t);
7 - curve = simplify(B*P);
8
9 - figure
10 - hold on
11 - grid on
12 - xlim([-6 6])
13 - plot(P(:,1), P(:, 2), '*r')
14 - fplot(curve(1), curve(2), [0 1])
15 - fplot(-curve(1), curve(2), [0 1])
```



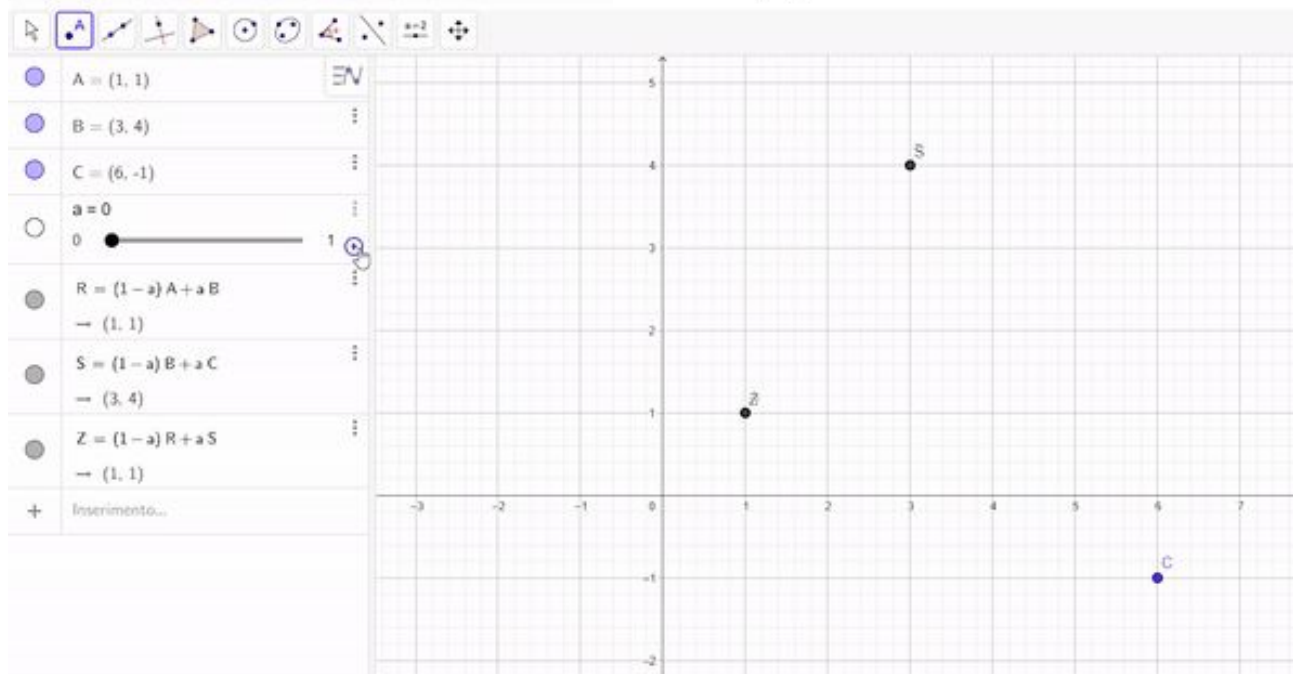
6 punti di controllo, curva di grado 5

# Algoritmo di de Casteljau

## Curva di Bézier con $i$ polinomi di Bernstein

$$P_0^n(t) = (1 - t)P_0^{n-1}(t) + tP_1^{n-1}(t).$$

$$B(t) = \sum_{i=0}^n \binom{n}{i} P_i (1 - t)^{n-i} t^i \quad t \in [0, 1].$$



# Perché le NURBS e non le B-spline?

---

**Alcuni motivi per la diffusa accettazione e popolarità di NURBS nella comunità CAD/CAM e grafica come segue:**

Offrono una forma matematica comune per rappresentare e progettare sia forme analitiche standard (coniche, quadriche, superfici di rivoluzione, ecc...) sia curve e superfici in forma libera

La valutazione è ragionevolmente veloce e computazionalmente stabile.

Le NURBS hanno chiare interpretazioni geometriche rendendole particolarmente utili per i progettisti che hanno una buona conoscenza della geometria descrittiva.

Le NURBS sono invarianti rispetto al ridimensionamento, alla rotazione, alla traslazione, al taglio e alla proiezione parallela e prospettica.

NURBS dispone di un potente kit di strumenti geometrici che può essere utilizzato per progettare, analizzare ed elaborare gli oggetti.

Le NURBS sono autentiche generalizzazioni delle B-spline e delle curve e superfici di Bézier razionali e non razionali

# NURBS (*Non Uniform Rational B-Spline*)

Con le NURBS è possibile rappresentare precisamente qualunque forma, sia in 2D che in 3D, con quattro informazioni principali:

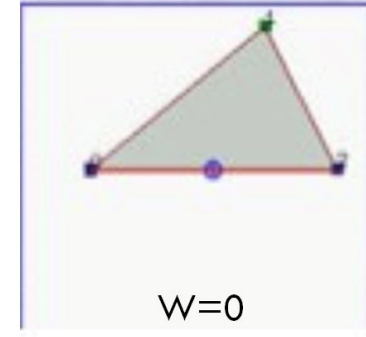
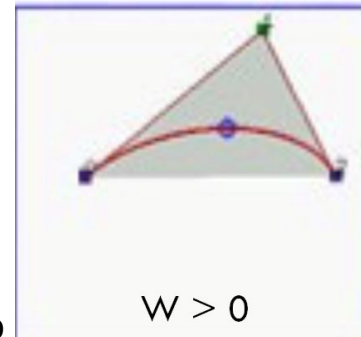
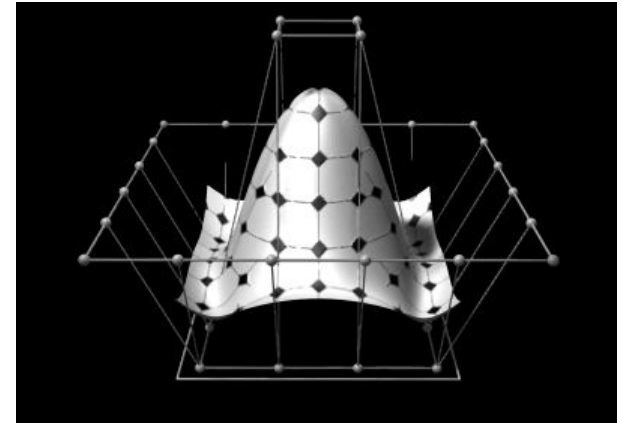
1. *un vettore di nodi*
2. *un insieme di funzione di base  $N_{i,p}(u)$*
3. *un grado fissato  $p$*
4. *punti di controllo*

Ad ogni punto di controllo è associato un peso  $w_i$  che rappresenta la sua capacità di "attrarre la curva".

***Quando i punti di controllo hanno lo stesso peso, la curva viene definita non razionale, razionale altrimenti.***

Con  $w$  grande, la curva tenderà ad avvicinarsi al punto di controllo ad esso associato

Con  $w = 0$ , la curva non sarà influenzata da quel determinato punto di controllo.



# NURBS (*Non Uniform Rational B-Spline*)

L'equazione che rappresenta in maniera univoca una NURBS è data da:

$$C(u) = \frac{\sum_{i=0}^n w_i P_i N_{i,p}(u)}{\sum_{i=0}^n w_i N_{i,p}(u)}$$

Dove  $N_{i,p}(u)$  sono le funzioni di base B-spline di grado  $p$  sul vettore dei nodi  $U$

$$N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{altrimenti} \end{cases}$$

La generica funzione  $N_{i,k}(u)$  con  $k > 0$  si calcola con la formula ricorsiva di **de Boor**.

Senza il polinomio al denominatore, la funzione non sarebbe razionale. Se tutti i pesi fossero uguali ad 1, la somma al denominatore diventa unitaria e la NURBS degenererebbe in una B-Spline.



# NURBS (*Non Uniform Rational B-Spline*)

---

È possibile dare una definizione più sintetica alle NURBS.

Posto un insieme con  $n + 1$  punti di controllo  $P_0, P_1, \dots, P_n$ , ad ognuno dei quali è associato un peso non negativo  $w_i$ , e un vettore di nodi  $U = [u_0, u_1, \dots, u_m]$  costituito da  $m + 1$  nodi, la curva NURBS di grado  $p$  è definita come segue:

$$C(u) = \sum_{i=0}^n R_{i,p}(u) P_i,$$

La funzione  $R_{i,p}(u)$  rappresenta la funzione di base della curva NURBS ed è definita come:

$$R_{i,p}(u) = \frac{N_{i,p}(u) \cdot w_i}{\sum_{j=0}^n N_{j,p}(u) \cdot w_j}$$

Le lettere NU in NURBS stanno per non uniformi, indicando che i nodi  $U$  possono essere non uniformi.

Se il vettore dei nodi uniforme è costituito con il primo e l'ultimo nodo a molteplicità piena, la curva è una curva di Bézier: interpola il primo e l'ultimo punto ed è tangente al primo e all'ultimo segmento del poligono di controllo.

# Non solo lati positivi

---

- *Vi è la necessità di conservare più informazioni anche per curve e superfici tradizionali. Per esempio, per rappresentare una circonferenza usando un quadrato circoscritto servono 7 punti di controllo e 10 nodi. In una rappresentazione tradizionale basterebbero il centro, il raggio e il vettore normale al piano contenente la circonferenza. In grafica 3D bisognerebbe utilizzare 38 punti.*
- *Alcune interrogazioni sono più facili con le tecniche tradizionali piuttosto che con le NURBS. Per esempio è più difficile individuare l'intersezione tra due superfici, trovare punti di tangenza...*
- *Uno sbagliato uso dei pesi dei punti di controllo può portare ad una cattiva parametrizzazione.*

# Proprietà delle NURBS

## 1. Generalizzazione

Se  $w_i = 1 \quad \forall i$ , allora  $R_{i,p}(u) = \begin{cases} B_{i,p}(u) & \text{se } U = [0, 0, \dots, 0, 1, 1, \dots, 1] \\ N_{i,p}(u) & \text{altrimenti} \end{cases}$

dove gli 0 e gli 1 sono ripetuti con molteplicità  $p+1$  e  $B_{i,p}(u)$  rappresenta il polinomio di Bernstein di grado  $p$ .

2. **Località:**  $\longrightarrow R_{i,p}(u) = 0$  se  $u \notin [u_i, u_{i+p+1})$

3. **Partizionamento dell'unità:**  $\longrightarrow \sum_i R_{i,p} = 1$

## 4. Differenziabilità:

Tra i nodi, la curva appartiene a  $C^\infty$ . Sui nodi, la curva appartiene a  $C^{p-k}$ . Ciò comporta che all'aumentare della molteplicità di un nodo, decresce il livello di continuità; inversamente, all'aumentare del grado  $p$ , quest'ultimo aumenta.

5.  $R_{i,p}(u; w_i = 0) = 0$

7.  $R_{i,p}(u; w_j \rightarrow +\infty) = 0 \quad j \neq i$

6.  $R_{i,p}(u; w_i \rightarrow +\infty) = 1$

# Conseguenze

---

1. **Local approximation:** se un punto di controllo viene spostato o ne viene modificato il peso, il numero di intervalli della curva che vengono affetti da questa modifica sono  $p+1$ ; in tal modo, non viene modificata tutta la curva, come accade per le curve di Bézier.
2. **Strong convex hull:** se  $u \notin [u_i, u_{i+1})$ , allora, la curva  $C(u)$  giace all'interno dell'involuppo convesso di  $P_{i-p}, \dots, P_i$ .
3. Settando il peso di un punto di controllo a zero, tale punto non ha alcun effetto sull'intera curva.
4. Se  $w_i \rightarrow +\infty$ , allora:

$$C(u) = \begin{cases} P_i & u \in (u_i, u_{i+p+1}) \\ C(u) & \text{altrimenti} \end{cases}$$


# Invarianza delle trasformazioni affini

Come affermato precedentemente, le curve NURBS sono invarianti rispetto al ridimensionamento, alla rotazione, alla traslazione e alla proiezione parallela e prospettica.

Applicando una trasformazione affine alla curva, si ottiene:

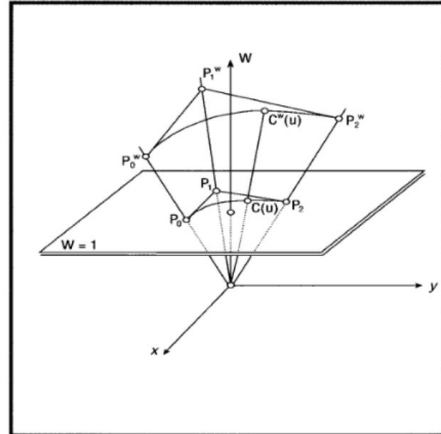
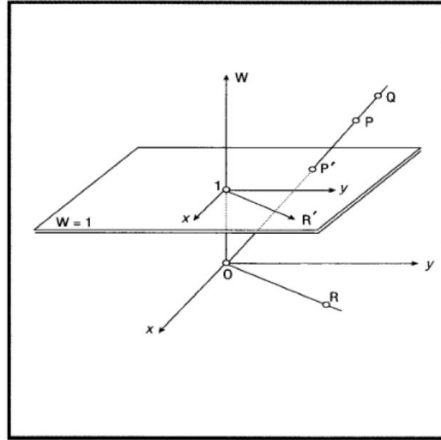
$$A[C(u)] = L[C(u)] + T = \sum_i L[P_i] R_{i,p}(u) + T.$$

D'altra parte, si ha:

$$\sum_i A[P_i] R_{i,p}(u) = \sum_i (L[P_i] + T) R_{i,p}(u) = \sum_i L[P_i] R_{i,p}(u) + T \sum_i R_{i,p}(u) = \sum_i L[P_i] R_{i,p}(u) + T$$


**Per la proprietà del Partizionamento dell'unità = 1**

# Coordinate Omogenee



$$\varphi(X, Y, W) = \begin{cases} \left( \frac{X}{W}, \frac{Y}{W} \right) & W \neq 0 \\ \text{direction}(X, Y) & W = 0 \end{cases}$$



$$C^w(u) = \sum_{i=0}^m P_i^w N_{i,p}(u)$$



$$C(u) = \varphi(C^w(u)) = \frac{\sum_{i=0}^n w_i P_i N_{i,p}(u)}{\sum_{i=0}^n w_i N_{i,p}(u)}$$

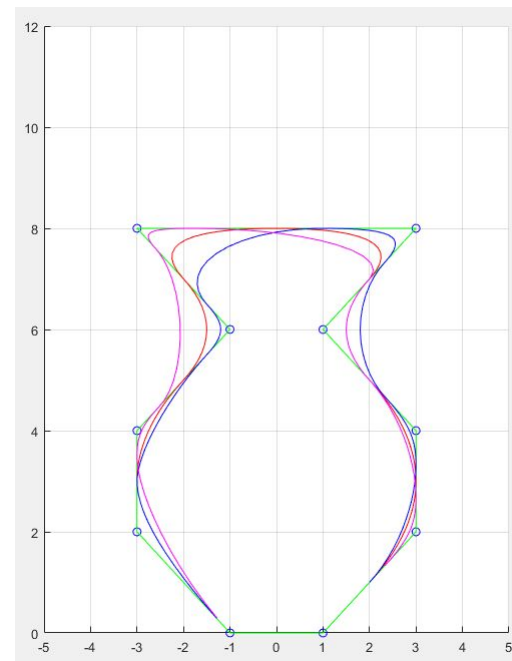
# Esempio

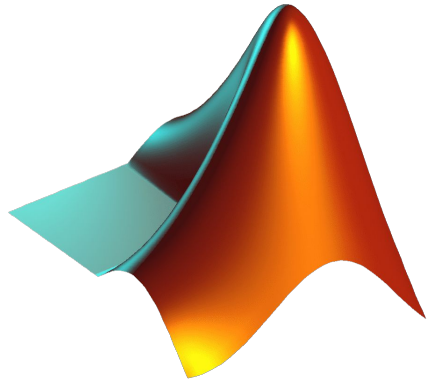
```

1      % Punti di controllo
2 -    P = [-1 -3 -3 -1 -3 3 1 3 3 1 -1; 0 2 4 6 8 8 6 4 2 0 0];
3 -    n = 3; % Ordine dei polinomi
4      % Vettore dei nodi
5 -    t = [0 0 0:1/(length(P)-n + 1):1 1 1];
6
7      % Peso
8 -    w = ones(1,length(P));
9      C = nurbsfun(n, t, w, P);
10 -   hold on
11 -   xlim([-5, 5])
12 -   ylim([0, 12])
13 -   plot(P(1, :), P(2, :), 'bo')
14 -   plot(P(1, :), P(2, :), 'g')
15 -   plot(C(1, :), C(2, :), 'r')
16 -   grid on;
17
18      % Cambio dei pesi
19 -    w = [1 1 1 3 1 2 1 2 1 1 1];
20 -    C = nurbsfun(n, t, w, P);
21 -    plot(C(1, :), C(2, :), 'b')
22
23      % Cambio pesi
24 -    w = [1 1 3 1 4 1 1 1 2 1 2];
25 -    C = nurbsfun(n, t, w, P);
26 -    plot(C(1, :), C(2, :), 'm')

```

Esempio di raffigurazione di curve NURBS al variare dei pesi e del vettore dei nodi.



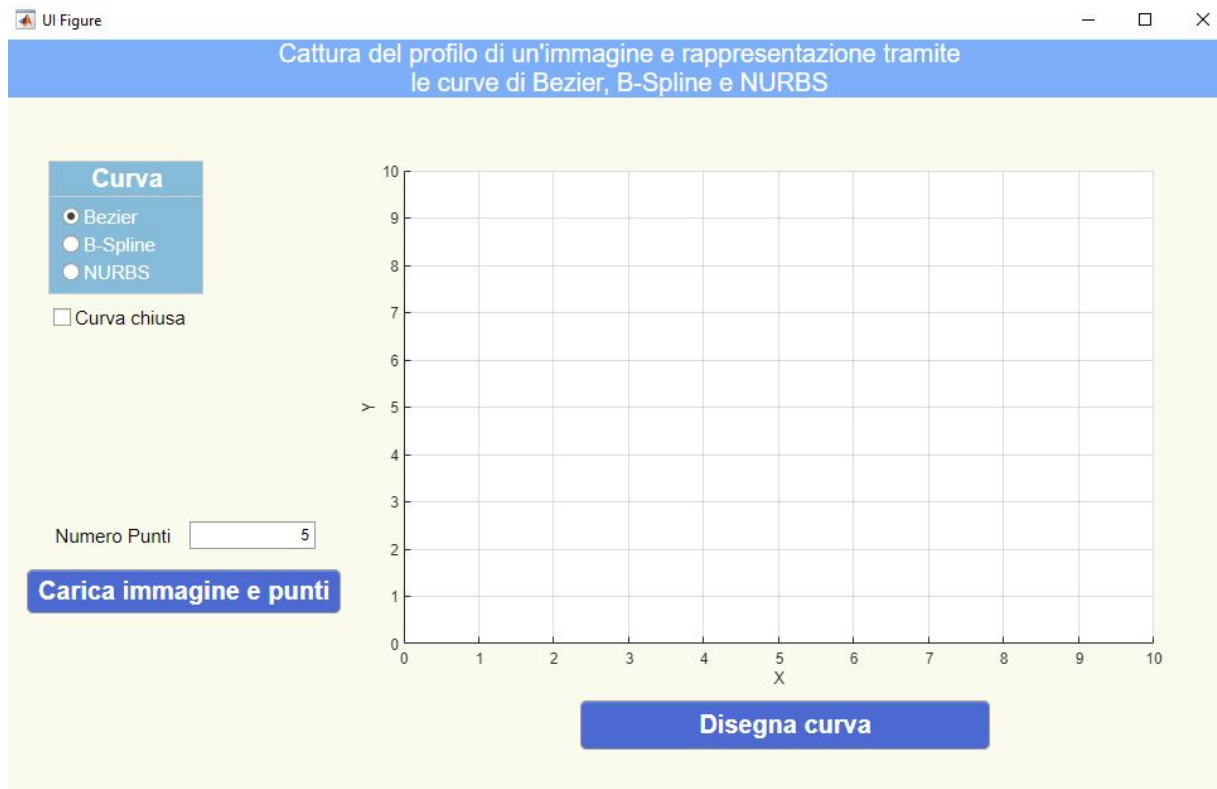


# ***Applicazione in Matlab App Designer***





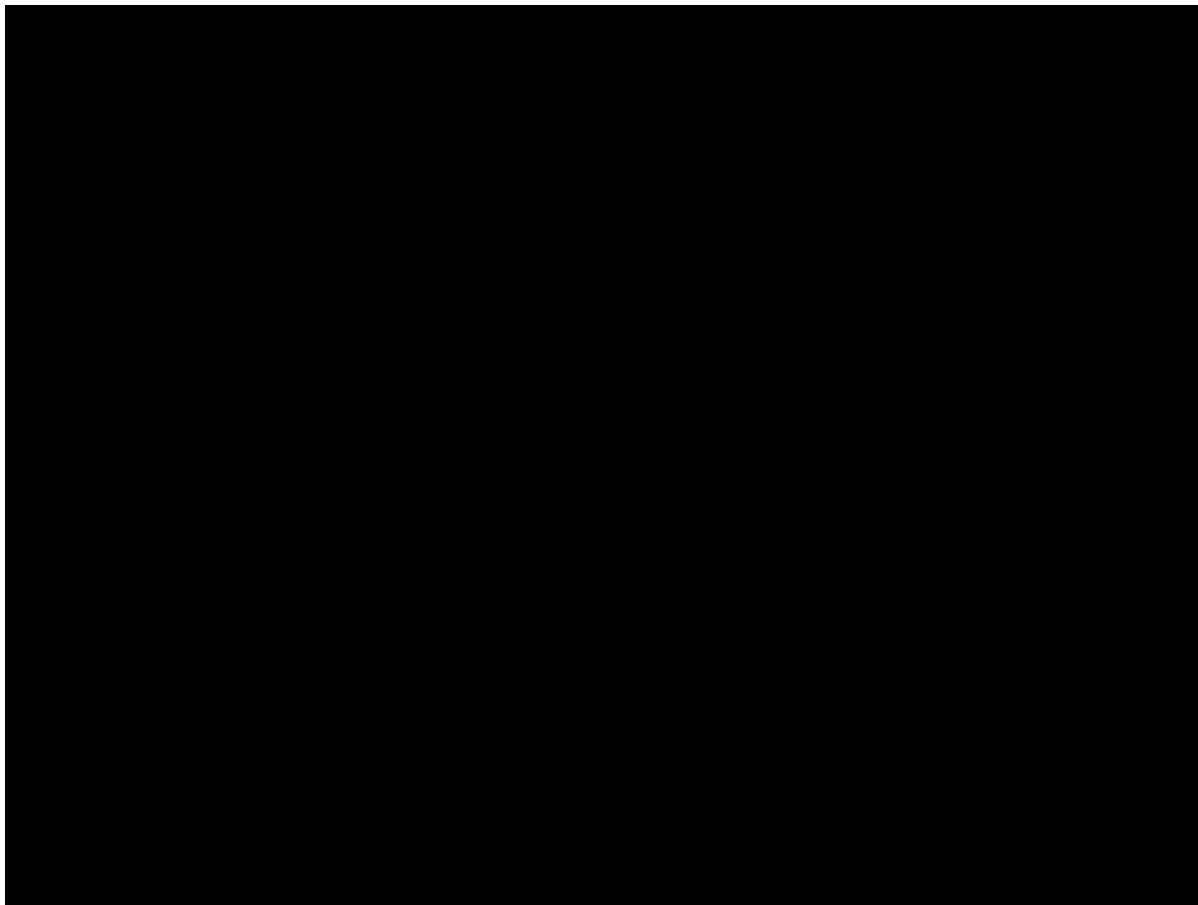
# Interfaccia



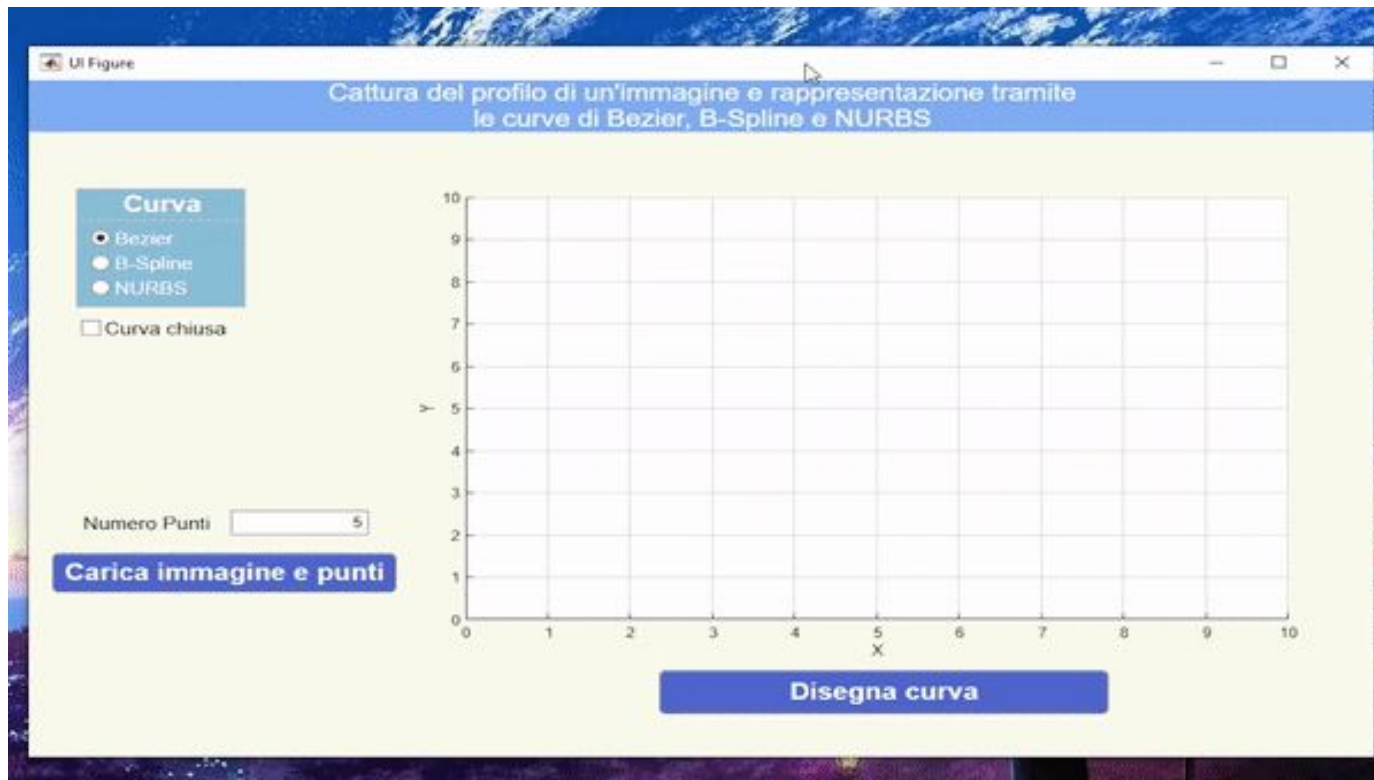


# Video del funzionamento

---

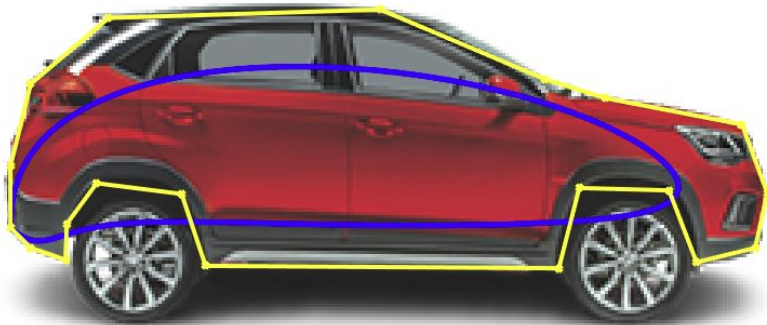


# Video del funzionamento

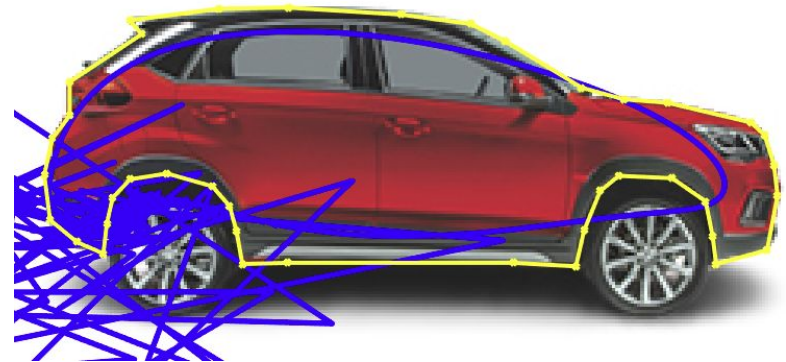


# Esempio di utilizzo dell'applicazione

## Bézier



20 punti



40 punti

# Esempio di utilizzo dell'applicazione

## B-Spline



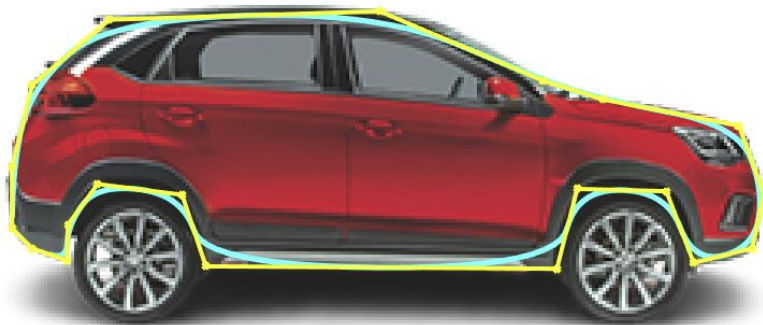
20 punti



40 punti

# Esempio di utilizzo dell'applicazione

## NURBS



20 punti



40 punti



# Codice

```
% Button pushed function: CaricaimmagineepuntiButton  
function CaricaimmagineepuntiButtonPushed(app, event)
```

```
    global imm;  
    global isRead;  
    [name, path] = uigetfile;  
    if name == 0  
        isRead = false;  
        return  
    end  
    isRead = true;  
  
    imm = imread(strcat(path, name));  
    [dimx, dimy] = size(imm(:, :, 1));  
    app.UIAxes.XLim = [0 dimx];  
    app.UIAxes.YLim = [0 dimy];  
    imshow(imm, [], 'parent', app.UIAxes);  
    hold(app.UIAxes, 'on')
```

```
    figure(1);  
    imshow(imm, []);  
    grid on;  
    hold on;  
    n_punti = app.NumeroPuntiEditField.Value;  
    global xp;  
    global yp;
```

```
    try  
        [xp, yp] = ginput(n_punti);  
    catch ignore  
        % CHIUSURA FIGURE CON X  
        return  
    end
```

```
    close(1)  
end
```

```
% Button pushed function: DisegnacurvaButton  
function DisegnacurvaButtonPushed(app, event)
```

```
    global isRead;  
    if isRead == false  
        return  
    end  
  
    hold(app.UIAxes, 'off')  
    global imm;  
    imshow(imm, [], 'parent', app.UIAxes);  
    hold(app.UIAxes, 'on')  
  
    global xp;  
    global yp;  
  
    if app.CurvachiusaCheckBox.Value == true  
        xp1 = [xp; xp(1)];  
        yp1 = [yp; yp(1)];  
    else  
        xp1 = xp;  
        yp1 = yp;  
    end  
  
    if app.BezierButton.Value == true  
        syms t  
        B = bernsteinMatrix(length(xp1) - 1, t);  
        P = [xp1 yp1];  
        curve = simplify(B*P);  
        fplot(app.UIAxes, curve(1), curve(2), [0 1], 'b', 'LineWidth', 4);  
        plot(app.UIAxes, xp1, yp1, '-xy', 'LineWidth', 3);
```

```
    else if app.BSplineButton_2.Value == true  
        cps = [xp1'; yp1'];  
        tpts = [0 5];  
        tvec = 0:0.01:5;  
        [Q, ~, ~, pp] = bsplinepolytraj(cps, tpts, tvec);  
        plot(app.UIAxes, cps(1,:), cps(2,:), '-xy', 'LineWidth', 3);  
        [point, E] = fnplt(pp);  
        plot(app.UIAxes, point(1,:), point(2,:), 'g', 'LineWidth', 4);  
    else  
        P = [xp1'; yp1'];  
        n_punti = length(xp1);  
        n = 3;  
        t = [0 0:1/(n_punti - n + 1):1 1 1];  
        w = ones(1, n_punti);  
        C = nurbsfun(n, t, w, P);  
        plot(app.UIAxes, C(1,:), C(2,:), 'c', 'LineWidth', 4);  
        plot(app.UIAxes, xp1, yp1, '-xy', 'LineWidth', 3);
```

```
    end  
end
```

***Grazie per l'attenzione!***