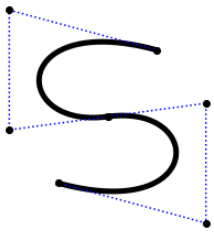


# tudy on Bézier curves and B-Splines

**Ayyar Meghna Parameswaran (HKJXSC)**

**This report documents and presents the study on Bézier Curves and B-Splines done by the author. The main objective was to understand the history, and the formulation of the Bézier curves and B-Splines which propelled the whole field of Computer Aided Graphic Design(CAGD) and how these ideas have gained such popularity. The report also delves into the theory, calculation of these curves with sample figures generated using codes written by the author. A brief note on Bézier surfaces and the idea of Non-Uniform Ration B-splineS (NURBS) has been discussed with simulations to show their application. A few interesting use cases of splines and Bézier curves has been presented to show the different domains where they are being applied. The Appendix section includes the details of the codes used to generate the figures that have been presented.**

Splines | Bézier Curves | Bézier Surfaces| B-Splines | NURBS

The use of curves for manufacturing has been recorded from as early as the AD Roman times. Mostly used for the purpose of shipbuilding, the vessel's basic geometry was stored in forms of wooden planks. The vessel's basic geometry was recorded as multiple curves that were used to build different parts (which can be seen as the NURBS and splines in modern parlance). There are sufficient recorded uses of splines, even before the existence of computer graphics. A spline was generally a flexible piece of wood or plastic held in place using heavyweights with protrusions called ducks. These ducks were used to hold the spline in a fixed position on a drawing board [Beach (1991)]. This made the spline conform to a particular shape between the ducks. These ducks then allow the designer to move them so as to change the shape of the spline, as seen in Fig 1. It is fascinating to see how before the advent of computers simple mechanical inventions were used to perform all the tasks that seem cumbersome without using a computer to work on. The disadvantages of such a method are many, especially recording the positions, storing all the equipment that would take up lots of storage space which would, in turn, increases the costs that have to borne by the consumer.

It was in 1959 that de Castlejau a mathematician was hired by the car company Citroën where he was asked to work on solving problems of converting paper blueprints into computer designs. Instead of converting existing blueprints to computer designs, he started working on a system on which the curves could be designed from scratch. He made use of the Bernstein polynomials as shown in Equation 2 for defining his curves and surfaces. His most significant insight was to use control polygons that could be manipulated to create many different types of curves. In the next decade, Pierre Bézier who was working on a similar task at a rival car company Renault came up independently with what de

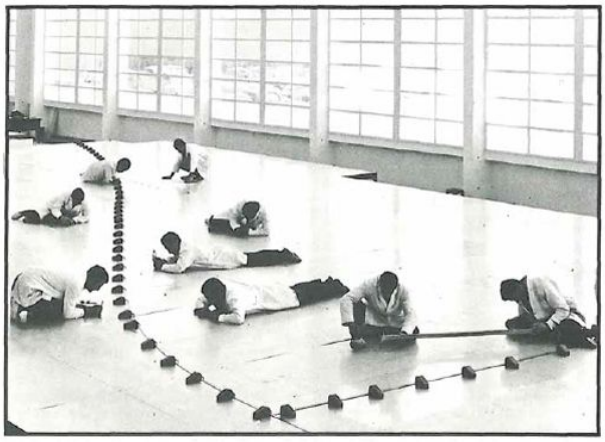


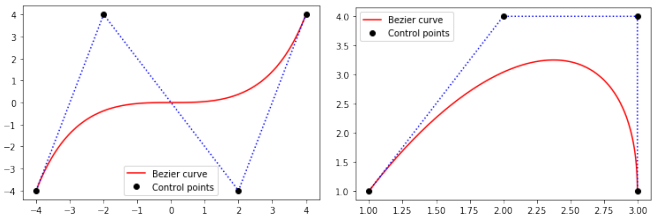
**Fig. 1.** People designing using a mechanical spline and ducks

Catlejau's team had also arrived on. The result turned out to be identical to de Casteljau's curves, only the mathematics involved was a little different. After Bézier published his work, A.R.R Forrest realized that Bézier curves could be expressed in terms of Bernstein polynomials, the same way that de Castlejau had used. Forrest's article on Bézier curves [Forrest (1972)] was very influential and helped popularize Bézier curves considerably. Bézier developed a new software called the UNISURF which was used to draw smooth curves on a computer screen, used less storage and physical space and paved the way for other software like CAD, Maya, Blender etc.

The following report will start at the point with Bézier curves and will explain their calculation, applications, advantages and disadvantages. From there the concept of B-Splines will be discussed in terms of how it ties up with Bézier curves. In addition, the calculations, improvements and some major applications of these curves and splines will be presented.

## 1. Bézier Curves

Bézier curves are parametric curves that are used to create smooth looking curves. They are defined by a set of points called the **control points**. The curve is controlled by the relative positions of these control points and doesn't need to pass through all of these control points. These control points can be seen as the ducks used in the mechanical splines that




(a) (b)

**Fig. 2.** a and b are cubic Bézier curves drawn with different control points

would determine how the curve would finally look like.

Bézier Curve is defined as the weighted sum of these control points with the weights as the Bernstein's polynomials of the parameter t. Varying the value of t we can draw the different points on the curve.

If P is the set of the points which will control our curve, where $P = \{P_0, P_1, P_2, P_3, ..., P_n\}$ and C(t) be the parametrically defined vector function of the curve with the parameter t, where $0 \leq t \leq 1$, then the curve can be calculated by equation 1.

$$C(t) = \sum_{i=0}^{n} B_i^n(t) P_i \quad [1]$$

The weights $B_i^n(t)$ are the Bernstein's polynomial for the different values of t given by equation 2.

$$B_n^i(t) = \binom{n}{i} t^i (1-t)^(n-i) \quad [2]$$

where i = 0, 1, 2, ..., n, and

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!} \quad [3]$$

Let P be a set of given control points with $P = \{P_0, P_1, P_2, P_3\}$. Then the curve C(t) with $n = 3$ defined by these points will be:

$$C(t) = B_0^3(t)P_0 + B_1^3(t)P_2 + B_2^3(t)P_2 + B_3^3(t)P_3 \quad [4]$$

The Bernstein polynomial weights can be expanded as

$$B_0^3 = (1-t)^3 \quad [5]$$

$$B_1^3 = 3t(1-t)^2 \quad [6]$$

$$B_2^3 = 3t^2(1-t) \quad [7]$$

$$B_3^3 = 3t^3 \quad [8]$$

So the complete equation of the curve can be written as,

$$C(t) = (-t^3 + 3t^2 - 3t + 1)P_0 + (3t^3 - 6t^2 + 3t)P_1 + (-3t^3 + 3t^2)P_2 + t^3 P_3 \quad [9]$$

Observing this the curve C(t) can be further written as equation 10 in the matrix form.

$$C(t) = \begin{pmatrix} P_0 & P_1 & P_2 & P_3 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix} \quad [10]$$

Once we get the matrix form, implementing any Bézier curve with four control points is doing two matrix multiplications for the different values of t for $0 \leq t \leq 1$. Fig 2 shows the output of the implementation of the Bézier curve for four control points using the equation 10 generated by the code mentioned in Appendix. Therefore, it is as simple as clicking four points for a designer to create a smooth curve.

Based on the above calculations the following observations can be made regarding Bézier curves:

1. The curve always passes through the first and last control point ($P_0$ and $P_n$).

2. The degree of the polynomial is one less than the number of control points we have. This because of the form of the Bernstein polynomials. For e.g. in equation 9, we see that the highest power of t is 3 when we have 4 control points.

3. A direct use of the above point is that we require a minimum of a cubic polynomial for the human eye to feel that curve is smooth. By using just four points from the space we can design a really smooth curve. Cubic Bézier curves are mostly frequently used for designing and it is not very common to use other higher order Bézier curves.

4. Each of the Bernstein polynomial weight will be a real.

5. A direct consequence of using Bernstein weights is that $\sum_{i=0}^{k} B_i^k(t) = 1$ where n = k+1 are the control points and k is the degree of the curve. This means that each point on the curve C(t) is the weighted average of the control points.

6. A **closed curve** can be drawn by keeping the first and the last control point the same ($P_0 = P_n$). An example is shown in Fig 3

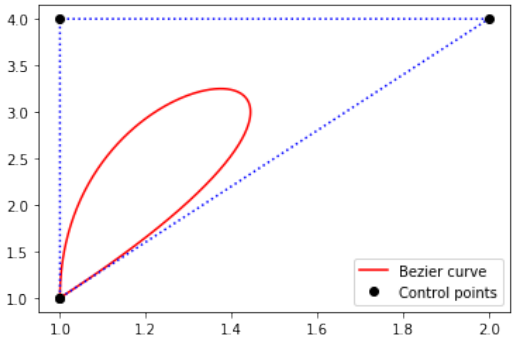



**Fig. 3.** A closed cubic Bézier curve

## 2. Bézier Surface

Extending the Bézier curve to two dimensions leads to formation of a Bézier surface. Similar to the Bézier curve, the Bézier surface can be defined by the parametric equation $C(u, v)$ where u and v are the parameters and $u, v \in [0, 1]$ (Note that, Bézier curve is a parametric equation with just one parameter $t \in [0, 1]$). The Bézier surface uses Bézier curve for the $x$ and $y$ direction. Due to this, in a Bézier surface we have can different degree curves in each of the two directions.

A Bézier surface of degree $(m, n)$ is defined by a $n$ degree curve in the $x$ direction and an $m$ degree curve in the $y$ direction. A degree $(m, n)$ Bézier surface is defined by $(n + 1) \times (m + 1)$ control points. Each of $x, y, z$ coordinates of the control points are written down into three different matrices denoted as $P_x, P_y, P_z$ each of size $(n+1) \times (m+1)$. This makes the calculation of coordinates on the Bézier surface simpler and we can calculate it as shown in the following equations where each of the $b$ are the Bernstein polynomial coefficients as shown in equation 2

$$x(u,v) = \sum_{j=0}^{n} \left[\sum_{i=0}^{m} b_{i,m}(u) P_{x,ij}\right] b_{j,n}(v) \quad [11]$$

$$y(u,v) = \sum_{j=0}^{n} \left[\sum_{i=0}^{m} b_{i,m}(u) P_{y,ij}\right] b_{j,n}(v) \quad [12]$$

$$z(u,v) = \sum_{j=0}^{n} \left[\sum_{i=0}^{m} b_{i,m}(u) P_{z,ij}\right] b_{j,n}(v) \quad [13]$$

As in the case of Bézier curve the surface calculation can also be written in a matrix form which makes it easier to implement in code. If we denote the $b_{n,ij}$ coefficient matrix as $N$ and the $b_{m,ij}$ matrix as $M$ we can write the general equation of a coordinate on the surface as equation 14.

$$x(u,v) = (u^m, u^{m-1}, ...u, 1) M . P_x . N \begin{pmatrix} v^n \\ v^{n-1} \\ . \\ . \\ . \\ 1 \end{pmatrix} \quad [14]$$

The final matrix form for the cubic $(3, 3)$ Bézier surface when expanded and written in the general form comes out to be equation 15

$$\mathrm{x(u,v)} = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \mathrm{P}_x \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix} \quad [15]$$

Fig 4 shows a degree $(3, 3)$ Bézier surface plot generated through code using the matrix equation. The control points were generated using the first surface given in the Utah Teapot dataset.

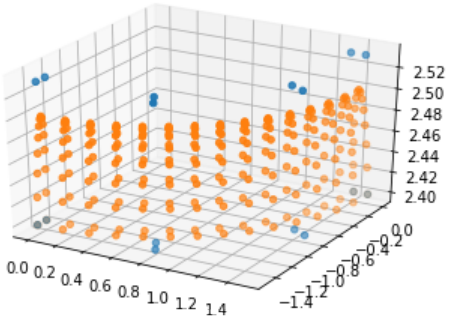



**Fig. 4.** Scatter plot of the coordinates calculated on a Bézier surface. The blue points are the control points and orange points are the surface points. The control points are taken from the first surface of the Utah teapot dataset.

In my previous work during my Bachelor's, I have worked on Bézier surfaces as part of a project for the Computer Graphics. Fig 5 has been generated reworking on my previous work to draw the Utah Teapot dataset using Bézier surfaces. The code is written in Python with the use of the Pygame library. In addition to drawing the surfaces, I extended my previous work to this dataset to also include the functions of rotation and zoom to manipulate the teapot object. The code details are mentioned in section 7

**Fig. 5.** Bézier surface plot of the Utah Teapot generated by my code

## 3. Bézier to B-Splines

While implementing the Bézier curves one main observation that can be made is that the control polygon formed by the control points are adjusted to draw different shapes of the curves. This in turn also means that changing even one control point would mean that we have to redraw/recalculate that curve again.

Also drawing complicated curves with just one Bézier curve is difficult. For a complicated shape, a higher degree Bézier curve needs to be calculated and for higher degree Bézier curves the control polygon lies farther away from the curve itself.

We don't have local control over the curve itself. If we need to add an extra control point then it means that we will have to go to a higher degree Bézier curve.

In cases when we have only a small number of control points Bézier curves are very useful but as the complexity of the curve and number of control points increase an alternative might prove useful. This is where the B-Splines come into the picture. They are essentially similar to Bézier curves but provide more control as they have the property of local control (splines are piece-wise polynomial curves). The following section presents B-Splines in detail.

## 4. B-Splines

**A. A little bit of history.** In 1946 I. Schoenberg introduced the term B-Splines (short for Basis Splines) for the case of uniform knots [Schoenberg (1946)]. C. de Boor in the year 1960 started his work at the General Motors Research Labs and used B-Splines for geometry representation. Eventually becoming one of the greatest proponents of B-Splines for approximation theory, the recursive algorithm used to calculate the B-Splines are now named the de Boor algorithm [De Boor (1972)]. Till then B-Splines was known in theory but their calculations involved a complicated divided difference approach which was numerically unstable. This recursion make the B-Splines a

viable tool for Computer-Aided Graphic Design(CAGD) and also helped with the use of splines in approximation theory.

Parametric curves like the B-Splines had become important for both CAGD and approximation theory and in the year 1974, R. Riesenfeld and W. Gordon [Gordon and Riesenfeld (1974)] realised that the de Boor recursive algorithm for B-spline evaluation was the natural generalization of the de Casteljau algorithm. B-spline curves include Bézier curves as a proper subset and soon became a core technique of almost all CAD systems. A first B-spline-to-Bézier conversion was found by W.Boehm [Boehm (1977)]. The generalization of B-spline curves to NURBS - Nonuniform rational B-splineS - has become the standard curve and surface form in the CAD/CAM industry [Gerald (1998)].

**B. Definition and computation.** A $k$ degree B-spline is formed by joining multiple pieces of polynomials (splines are piecewise polynomials) of degree utmost $k-1$ and an utmost continuity of $C^{k-1}$ at the breakpoints. B-Splines use multiple Bézier curves joined end-to-end to form a curve which is given by the control points for the spline. A $k$ degree B-Spline defined by $n+1$ control points consists of $n-k+1$ Bézier curves. The following are a few assumptions made for the B-Splines which leads to the joining of the multiple Bézier curves to form a single smooth curve without abrupt breaks or corners.

1. The last point of the first Bézier curve is the same as the first point of the second Bézier curve. ($C^0$ continuity)

2. The first and second derivative at the end of the first Bézier curve is the same as the first and second derivative at the start of the second Bézier curve respectively. ($C^1$ and $C^2$ continuity)

The parametric equation of the B-Splines is very similar to that of the Bézier curve as shown in equation 16 where $P_0, P_1, ..., P_n$ is the set of control points and instead of the Bernstein polynomials we have $N_{i,k}$ that are the basis functions that define the spline (thus the name basis splines).

$$S(t) = \sum_{i=0}^{n} N_{i,k}(t) P_i \qquad [16]$$

The basis for B-Splines are given by the Cox-de Boor recursion formula De Boor (1972). The recursion is given by the equations 17 and 18

$$N_{i,0}(t) = \begin{cases} 1, & \text{if } t_i \leq t \leq t_{i+1} \\ 0, & \text{otherwise} \end{cases} \qquad [17]$$

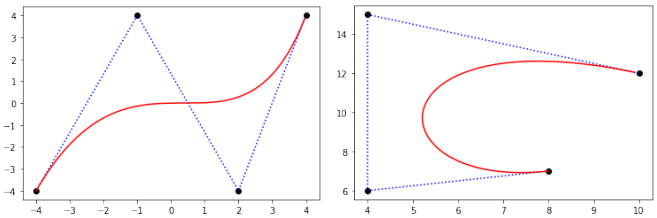
$$N_{i,j}(t) = \frac{t - t_i}{t_{i+j} - t_i} N_{i,j-1}(t) + \frac{t_{i+j+1} - t}{t_{i+j+1} - t_{i+1}} N_{i+1,j-1}(t) \qquad [18]$$

The parameter $t_i$ for the B-splines are taken from the knot vector given as

$$T = (t_0, t_1, t_2, ..., t_m) \qquad [19]$$

and $t_i \in [t_0, t_m]$. The number of knots $(m+1)$ required to define the B-spline of degree $k$ and with $n+1$ control points is given as $m = n + k + 1$.

Some observations about the B-spline:



(a) (b)

**Fig. 6.** a and b are cubic B-Splines curves drawn with different control points

1. If the distance between any two consecutive knots is the same (i.e they are equidistant) then it is called a **uniform B-spline**.

2. B-Splines do not always pass through the two end points of the control polygon. By setting the first $k$ and the last $k$ knots to be same value, we can force the B-spline to pass through the end points of the control polygon. Such a spline is said to be **clamped**.

3. The knots that $t$ lies between determines the basis function that affects the shape of the B-spline. This means that adding or changing a control point affects only the pieces of the curve near that point and we get local control over the complete curve. We can modify a curve locally without changing the shape in a global way.

4. Increasing the control points increases the number of Bézier curves that are joined together. It doesn't change the degree of the spline itself.

In comparison to Bézier curves, B-splines require more information in term of degree of the curve and the knot vector. It has a complex recursion formula when compared to the simple Bernstein polynomial of Bézier curve. But, these shortcomings are offset by the fact that B-splines provide more flexibility than Bézier curves, while satisfying all the properties of the Bézier curves.

**C. NURBS.** The calculation of B-spline mentioned in the previous section produce smooth curves but are unable to produce a simple curve like the circle. To represent a circle looking curve itself we would end up requiring a very high degree spline. It is not sensible to use to higher degree spline to represent a curve of degree 2.

One way to modify the B-spline calculation is by introducing weights for each of the control points. Thus, Non-Uniform Rational B-splineS (NURBS) were introduced and are written as shown in equation 20 while equation 21 gives the NURBS basis functions. The NURBS curve is defines a curve of degree $p$, control points $P_0, ..., P_n$, knot vector $T = t_0, ..., t_m$ and the weights for each of the control points given by $w_0, w_1, ..., w_n$. These weight points help in increasing the 'pull' of a control point on the curve thus making it more controllable in shape.

$$C(t) = \sum_{i=0}^{n} R_{i,p}(t) P_i \qquad [20]$$

$$R_{i,p}(t) = \frac{N_{i,p}(t) w_i}{\sum_{j=0}^{n} N_{j,p}(t) w_j} \qquad [21]$$

Fig 7 shows the different splines drawn using the same control points and same degree of (cubic) spline. Each line corresponds to a different weight assigned to the second control point. As it can be seen, the end points remain the same and changing the weight increases the 'pull' of the second control point on the curve and makes it move more closer without altering the smoothness of the curve.

Hence, NURBS are extremely useful to draw curves that are smooth, do not need to have higher degree to represent more number of control points and also have the ability to be modified using weights.

Some observations made about NURBS:

1. The weights are generally non-negative and non zero. If the value of a weight is $w_i = 0$ then the coefficient of the corresponding control point $Pi$ also becomes 0. So $P_i$ has no contribution while computing the curve.

2. All the properties of B-Splines still hold true. These curves have only an added extra weight term to them.

3. On making all the weights = 1, the equation reduces to the equation of the B-spline. Furthermore if the degree of the curve is equal to the number of control points minus one and the knots are clamped at both the ends, then this equation would represent a Bézier curve. Thus NURBS is the generalization of B-Splines and Bézier curves or we can call them as special cases of NURBS.

4. NURBS can be used to exactly represent conics like circle, parabola and hyperbola which is not easily done with a smaller degree simple B-Spline. NURBS allow general representation of free form surfaces.

One special property of NURBS is that are invariant to projection transformation. To apply a geometric or projection transformation to a NURBS curve, we can apply it to the control points and calculate the curve from the transformed points. This does not require us to transform the curve directly Piegl (1991). NURBS are extensively used in CAD/CAM and is an industry standard tool found in many 3D modelling softwares.

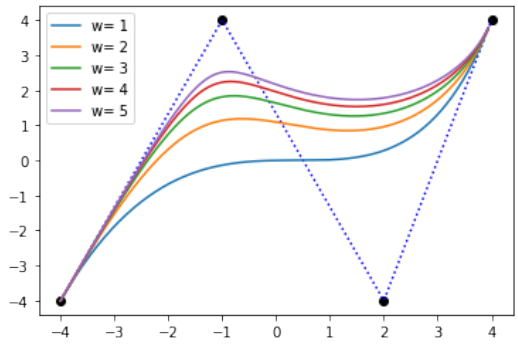



**Fig. 7.** NURBS curve. Each curve has the same control points and degree, but the weight of the second point is varied. Accordingly shape of the curve changes

## 5. Some use cases of B-Splines and Bézier curves

This section presents a few papers that use splines for an application other than computer graphics. It also presents a my previous work where I implemented B-splines.

**A. Personal Project.** As part of a summer project during my Bachelor's I had worked on generating non-photorealistic speedlines to summarize object motion in short video segments. Speedlines are the lines that we generally see in comic strips and animations drawn on the back of the object to show that it has moved fast. In that project the goal was to use speedlines to mark the start, end and the path taken by a moving object so that we could save only the image with the speedlines instead of the video itself. So to draw the speedlines, the back edge of the moving object was detected and tracked across the frames of the video. With the trajectory points of the object, various different interpolations were tried including straight line, Lagrange interpolation and among them B-spline curve drawing was one of the methods used to draw the speedlines. The use of straight line didn't capture a curved path well and using other interpolation schemes was dependent on the number of points that we had. Using B-splines was a good option as with a limited set of points the path taken by the object could be drawn well. Fig 8, shows one of the results generated during that project, that shows the speedlines drawn along the path that the character travelled in that video.

**Fig. 8.** Speedlines of the object drawn by using B-Splines

**B. Navigation for Robots.** The authors of Shuai et al. (2014) propose a new method combining the sub-targets algorithm and the cubic splines to generate a real-time path for the robots that would avoid obstacles that get detected. They combine the sub-targets algorithm which can generate sub-optimal but smooth and collision avoiding paths for the robots to navigate in a dynamic environment. The experiments are presented in the RoboCup environment where each robot has 9 obstacles it has to cross to get to the target.

Most importantly the authors use the sub-targets algorithm to generate via points or the points on a generated path such that the path would avoid all the obstacles present in the scene. Using these points as the control points they generate a cubic spline path which is a smooth geometric path along these control points. If the environment changes, i.e the via points change positions, increase or decrease in number then the cubic spline is well suited as it can be completely controlled by the via points and changing a few control points changes the shape of the spline only in pieces controlled by the changing points and not the whole path itself. This makes it possible to control the velocity and acceleration of the robot to arbitrary desired profiles. The results they have presented are an improvement and is more efficient over just using sub-targets algorithm and some other via points finding algorithms they have discussed in their work.

It is interesting to see that cubic B-splines can be coupled with other algorithms that generate path to control a robot.

In a dynamic environment if there needs to be real time path planning then splines that allow local piece wise control are extremely useful so that keeping the ends fixed, section of the path can be manipulated easily.

**C. Optimization of NURBS.** There has been previous work on using Genetic Algorithm(GA) and differnt types of splines for path planning and navigation. In their work, Jalel et al. (2015) propose the use of genetic algorithms to optimize the construction of NURBS for global path planning for mobile robot navigation. Their argument is that using just a GA does not take into consideration the size of the robot itself and consider them to be point sized which could lead to unsafe paths. Also genetic algorithms do not take into account the curvature or continuity of the path thus producing segments of the path which means that the robot would lose some extra energy to change directions along these segments.

After generating the control points by their proposed method (not discussed here as it out of scope for this study), the path has been generated using a GA-based NURBS curves. The algorithm proposed adapts GA by considering the a single NURBS curve and the path it has to generate as an individual. This curve is then defined by the set of weighted control points. Each gene is the location of the via point and its associated weight. In the algorithm the via point coordinates itself remain constant whereas the weights are kept to be varying. The fitness of the GA also takes into account the curvature of the path and tries to find a good weight parameterization for the NURBS curve.

This work presents an interesting combination of GA and NURBS based path planning. The authors have evaluated their method on different paths with different obstacles settings and parameters. The method is shown to be effective and one consequence observed is the low standard deviations of the curvature valued terms in the genetic algorithm. This is an indicates that the points are closer to the mean and hence the criterion for a smooth path is being satisfied by their algorithm.

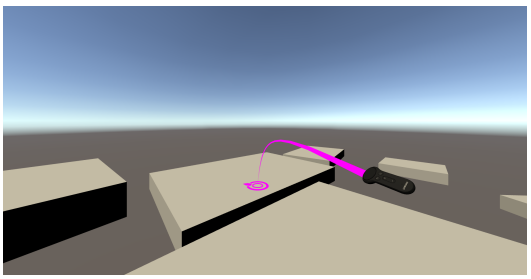


**Fig. 9.** Teleport arc example. The pink line is the arc that is calculated from the controllers angle. (The image has been taken from the blog mentioned in the Section D and hasn't been created by the author)

**D. Teleportation in Virtual Reality (VR).** In addition, to being essential for designing shapes/figures/surfaces for computer graphics, Bézier Curves and B-Splines are an integral part of game design engines like Unity and Unreal. One interesting application I came across was for VR applications and involves the basic tool of transportation. If the user wearing the VR glasses with the handheld controller has to walk to the part of the game/scene that is farther away from the confines of the room that they are in, they can use the Teleport arc 9 tool, which is one of the most basic built-in tools for VR design.

As demonstrated in the article on Teleport arc tool*, it calculates a parabola based on the angle at which you hold the controller and the destination where the users want to land that they have selected by adjusting the controller. This parabola is constructed by calculating a Bézier curve relative to the controller (the handheld tracker). By using the start point, end point and the mid point on the parabola, the quadratic Bézier curve for the path can be easily calculated. By looping through each segment of the curve that is constructed by varying the 't' parameter we can then check if the curve collides with any object that is already placed on the scene. The Unity engine has one implementation of the teleport arc with Bézier curves and the Unreal Engine has an inbuilt teleport spline tool that helps do the same using splines.

With VR and Augmented Reality(AR) gaining much popularity where even the latest smartphones have started implementing them, it was very fascinating to see how problems like teleportation were solved using a simple implementation of already well known curves and splines.

## 6. Conclusion

A breif history about Bézier curves and B-Splines was presented and discussed. The report reviews the computation of the Bézier curves and Bézier surfaces and includes the author's understanding about their behaviour. B-Splines were introduced and their computation presented with an brief overview about NURBS as a generalization of both the Bézier curves and B-Splines. Using Python codes written by the author figures for each of the topics presented has been generated and explained in detail. The codes have been attached as supplementary material along with the report. Some interesting applications were discussed based on some literature that the author had gone through during their work. It is apparent that Bézier curves and B-Splines have found varied applications in problems much more than they were initially conceived for and still remain one of the standard tools in graphics design softwares, that are also useful for path generation for mobile robots and for VR/AR game design suites.

### The "S" in the title

During my study I came across many applications and examples of using Bézier curves and B-Splines to draw letters of the alphabet and also for drawing different types of fonts from different languages. The "S" shown in the title of the report has been generated using two cubic Bézier curves. It was just an interesting exercise for me to figure out the correct control points to draw the letter S.

## 7. Appendix

This section contains the references for the codes submitted along with this report and gives details about the names of the script file and the corresponding result presented.

1. **Bézier curves and surface plot**: The code for the figures is named bezier_curves.ipynb. The notebook in-

*https://developer.oculus.com/blog/teleport-curves-with-the-gear-vr-controller/

cludes the codes, control points and the results that are shown in this report

2. **Teapot generation**: The code for Utah teapot rendering is name utah_teapot.ipynb. For the notebook to run, it needs the support of the PyGame library. On executing the cells, a window with the rendered teapot will open where it can be rotated along top, bottom, left and right directions using the WASD keys and zooming can be done by using the Z and X keys. The teapot has been rendered by constructing Bézier surfaces over the points given by the TeapotBezier.txt file and the poly.txt file.

3. **B-splines**: The code for the B-splines and NURBS figure is named b_splines.ipynb. The notebook has a function to take in any number of control points and degree of the spline and will generate a B-spline for it. It also shows a basic implementation of the NURBS where different weights can be assigned to generate different curves.

4. **Code for personal project** Due to the project being done as part of my Bachelor's internship presentation, I won't be able to share the complete code. But the function to generate the B-Spline is similar to the one used to generate the figure for B-Splines in this report.

## References


Beach, R. C. (1991). *An Introduction to the Curves and Surfaces of Computer-Aided Design*. Van Nostrand Reinhold Co.

Boehm, W. (1977). Cubic b-spline curves and surfaces in computer aided geometric design. *Computing*, 19(1):29–34.

De Boor, C. (1972). On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62.

Forrest, A. R. (1972). Interactive interpolation and approximation by bézier polynomials. *The Computer Journal*, 15(1):71–79.

Gerald, F. (1998). A history of curves and surfaces in cagd. *Arizona State University, USA*.

Gordon, W. J. and Riesenfeld, R. F. (1974). B-spline curves and surfaces. In *Computer aided geometric design*, pages 95–126. Elsevier.

Jalel, S., Marthon, P., and Hamouda, A. (2015). Optimized nurbs curves modelling using genetic algorithm for mobile robot navigation. In *International Conference on Computer Analysis of Images and Patterns*, pages 534–545. Springer.

Piegl, L. (1991). On nurbs: a survey. *IEEE Computer Graphics and Applications*, 11(1):55–71.

Schoenberg, I. J. (1946). Contributions to the problem of approximation of equidistant data by analytic functions. part b. on the problem of osculatory interpolation. a second class of analytic approximation formulae. *Quarterly of Applied Mathematics*, 4(2):112–141.

Shuai, C., Junhao, X., and Lu, H. (2014). Real-time obstacle avoidance using subtargets and cubic b-spline for mobile robots. pages 634–639.