Generation and Visualization of NURBS Surfaces Derived from Controlled NURBS Curves

Ruben Teodoro Urbina Guzman(✉), Yheff Alexander Castillo Maza, and Ronald Paul Santamaria Silupu

Universidad Nacional de Piura, Urb. Miraflores s/n, Piura, Peru
rurbinag@unp.edu.pe, 1302018004@alumnos.unp.edu.pe, rsantamarias@egresados.unp.edu.pe

Abstract. This study presents the generation and visualization of Non-Uniform Rational B-Spline surfaces derived from controlled curves, implemented in Python within the Jupyter Notebook environment. Three main approaches are examined: translation surfaces, revolution surfaces, and tensor product surfaces constructed from B-spline curves.

Matrix-based algorithms and object-oriented implementations enable the precise modeling of parametric surfaces through control points, weights, and knot vectors. Among the approaches, revolution surfaces exhibit the highest geometric accuracy and smoothness when applied to symmetric base curves, such as circles and ellipses. Translation surfaces are computationally efficient but limited to linear or parallel curve interactions, while tensor product surfaces demonstrate greater versatility for complex free-form modeling, albeit with increased computational cost.

Experimental results show that the methods achieve high visual fidelity and parametric continuity for both regular and irregular shapes, including cylinders, cones, hyperboloids, and user-defined geometries. In addition to its computational value, the study offers an educational perspective: the use of Python as a programming and visualization tool facilitates the comprehension of abstract mathematical concepts in undergraduate-level geometry and modeling courses. Its readable syntax and interactive environment foster experimentation, allowing students to engage with core ideas in computational geometry through practical implementations.

These findings highlight the adaptability and pedagogical potential of NURBS surfaces in both scientific applications and mathematics education.

Keywords: NURBS curves · NURBS surfaces · Surface generation · Python modeling · Geometric design

1 Introduction

The Non-Uniform Rational B-Spline (NURBS) representation constitutes a fundamental standard in computational geometry and is widely used in Computer-

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15648, pp. 3–20, 2025.
https://doi.org/10.1007/978-3-031-97000-9_1

Aided Geometric Design (CAGD) for the precise modeling of curves and surfaces [1]. It supports the construction of both simple entities, such as lines and circles, and complex surfaces in $\mathbb{E}^2 \to \mathbb{E}^2$ and $\mathbb{E}^3 \to \mathbb{E}^3$. Advancements in NURBS have enhanced their generation, approximation, and manipulation in geometric modeling and CAD. Techniques based on curve reparameterization have enabled the construction of developable surfaces using algebraic formulations [1], while generalizations of NURBS provide improved control over complex geometries such as helicoids and ruled surfaces [2]. Algorithms for precise surface fitting, surface generation, distance calculations, and intersection detection have been proposed, especially for applications in digital manufacturing and computer graphics [3,4]. Tejada's study [5] demonstrated the construction of revolution spheres and tori by rotating base curves, though this approach is constrained to single-axis rotations. Meanwhile, Cuartas [6] optimized NURBS surfaces using 3D scanned data and the Levenberg-Marquardt algorithm, achieving high accuracy but requiring specialized acquisition hardware. Python, a high-level general-purpose programming language, is increasingly employed for its readable syntax and accessible learning curve [7]. Several libraries enable the manipulation and visualization of NURBS geometry, although they may lack low-level control for detailed modeling tasks. Recent research has expanded Python's capabilities in this area. For instance, [8] introduces algorithms that convert 3D images into CAD models through control networks and optimization, while [9] presents a GPU-accelerated differentiable module in PyTorch for computing derivatives and enhancing CAD-related operations.

In addition to its computational utility, Python also offers pedagogical advantages. Its flexible syntax and support for multiple paradigms—structured, object-oriented, functional, and imperative—foster student engagement in programming and promote deeper understanding of mathematical structures [10,11]. Interactive environments like Jupyter Notebook further support exploratory learning by enabling real-time manipulation of control points, weights, and knots.

This work proposes a computational framework for constructing and visualizing NURBS curves and surfaces using Python. The foundation is the mathematical definition of B-spline basis functions, implemented through a set of Python classes and functions. The methodology supports three construction strategies: (1) translation of a generating curve along a guiding curve; (2) generation of revolution surfaces via matrix products; and (3) formation of tensor product surfaces from two or more B-spline curves. These approaches have been optimized for compatibility across Python environments.

The resulting implementations validate the effectiveness of the proposed methods, yielding precise representations of NURBS geometries and extending the modeling capacity for complex surfaces via generalized matrix operations.

2 Mathematical Preliminaries

This section outlines the foundational mathematical tools used in constructing and visualizing NURBS surfaces, including B-spline basis functions, rational curves, and matrix transformations.

2.1 Bases B-Spline

Let $U = \{u_0, \ldots, u_m\}$ be a non-decreasing sequence of real numbers, i.e., $u_i \leq u_{i+1},\ i = 0, \ldots, m-1$. The u_i are called *knots*, and U is the *knot vector*. The i-th B-spline basis function of degree p (order $p+1$), denoted as $N_{i,p}(u)$, is defined as:

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

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u).$$

– $N_{i,0}(u)$ is a step function, equal to zero everywhere except in the half-open interval $u \in [u_i, u_{i+1})$.

2.2 NURBS Curves

A p-degree NURBS curve is defined by

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

where

– The P_i are the control points (forming a control polygon). These influence the shape of the curve, and not all points necessarily lie on the curve.
– The w_i are the weights, which are associated with each control point P_i.
– The $N_{i,p}(u)$ are the p-degree B-spline basis functions defined on a non-periodic (and non-uniform) knot vector.

$$U = \{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\}$$

Unless otherwise stated, we assume that $a = 0$, $b = 1$, and that $w_i > 0$ for all i.

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

This allows us to rewrite the equation in the form:

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

The $R_{i,p}(u)$ are the rational basis functions; they are piecewise rational functions defined on $u \in [0, 1]$.

2.3 Superficies NURBS

The definitions stated in this subsection are taken from [10]. A NURBS surface of degree p in the u direction and degree q in the v direction is a piecewise rational bivariate and vector-valued function of the form:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}}, \quad 0 \leq u, v \leq 1$$

The $P_{i,j}$ form a bidirectional control mesh, the $w_{i,j}$ are the weights, and the $N_{i,p}(u)$ and $N_{j,q}(v)$ are the non-rational B-spline basis functions.

Introducing the piecewise-defined rational basis functions.

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}}$$

Thus, the equation of the surface can be written as follows:

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)P_{i,j}$$

It is worth mentioning that the generation of the weights $w_{i,j}$ is derived from the weights w_i and w_j through an internal operation, which is generally convenient to use as the product. Similarly, the control points $P_{i,j}$ are generated from the control points P_i and P_j through an internal operation (such as addition, a cross product, among others).

2.4 Rotation Matrices in 2D and 3D

According to Dunn [19], there are two types of rotations. The first refers to transforming each point of the object embedded in a space to another point within the same space, while the second refers to transforming the coordinate space itself. The commonly known rotation matrices usually correspond to the first type of transformation, meaning they transform each point of the object.

When rotating in the two-dimensional plane by an angle θ, it is done with reference to a point, which for simplicity is taken as the origin of the system.

$$\mathbf{R}(\theta) = \begin{bmatrix} p' \\ q' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix}$$

When the rotation occurs in three-dimensional space, it is performed with reference to an axis. Thus, there are three fundamental rotation matrices, each corresponding to the coordinate axes (the x-axis, y-axis, and z-axis).

After constructing a matrix from the rotated basis vectors, we obtain:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} p' \\ q' \\ r' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} p' \\ q' \\ r' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} p' \\ q' \\ r' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.5 Translation Surfaces

According to [20], translation surfaces are a method developed using a guiding curve $\Gamma_1 : \boldsymbol{R}(u) = (x(u), y(u), z(u))$ and a generating curve $\Gamma_2 : \boldsymbol{R}^*(v) = (x^*(v), y^*(v), z^*(v))$. These surfaces consist of all the curves obtained by translating the generating curve parallel to itself as a point $P_0 = (x_0, y_0, z_0)$ moves along the guiding curve. It is important to note that P_0 is a common point between the curves Γ_1 and Γ_2 (Fig. 1).

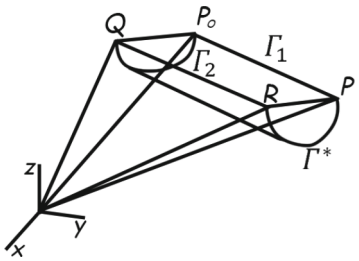

Fig. 1. Translation surfaces, obtained from [20].

The parametric equation of these translation surfaces is:

$$\begin{cases} x = x(u) + x^*(v) - x_0 \\ y = y(u) + y^*(v) - y_0 \\ z = z(u) + z^*(v) - z_0 \end{cases}$$

3 Construction of NURBS Circles

3.1 Generation of Control Points, Weights, and Knots on n-Sided Polygons

To generate NURBS circles inscribed in n-sided polygons, specific control points, weights, and parameters are used. For this purpose, a class is introduced that, given the parameter n (number of polygon sides) and r (radius of the inscribed circle), generates the NURBS circle using subcommands previously implemented within the class `Construction`:

– `Polygon` is an attribute of the `Construction` class and returns a list of two-dimensional points, which are the control points of the circle.

Code 1

```
A = Construccion(n=num_lado, r=radio)
A.polygon()
```

– `Peso` is another attribute of the `Construction` class and returns a list of numbers that represent the weights associated with each control point.

Code 2

```
A = Construccion(n=num_lado, r=radio)
A.peso()
```

– Lastly, `open1` is an attribute that returns a list of numbers representing the nodes of the circle, i.e., the interval.

Code 3

```
A = Construccion(n=num_lado, r=radio)
A.open1()
```

3.2 NURBS Circles Through Control Points Obtained from Polygons

The construction of circles is initiated by inscribed within an n-sided polygon, using an approach based on strategically placed control points on the polygons. These control points define NURBS curves that allow precise modeling of the inscribed circle in the polygon. The following class enables the visualization of a circle with radius "r" inscribed in an "n"-sided polygon with "p" degrees of freedom.

Code 4

```
C3 = Nurbs_curve2D_graph(n=num_lados, r=radio, p=grado
    de la curva)
C3.grafica(puntos=True)
```

The implementation is carried out through the class `Nurbs_curve2D_graph`, which receives three parameters: number of sides (n), radius (r), and degree of freedom ($p = 2$) corresponding to the circle. Additionally, the attribute `gráfica` includes a boolean parameter (`puntos`) that allows visualizing the circle with or without the control points used. This design ensures flexibility and clarity in the representation of the model.

4 Generation of NURBS Surfaces

Three approaches for surface generation are presented. The first focuses on translation surfaces. The second involves revolution surfaces based on matrix multiplication, described in 4.2. Finally, the third approach consists of tensor product surfaces, which are constructed using a set of points defined by two or more curves, allowing the formation of surfaces bounded by them.

4.1 Construction of Translation Surfaces

The following describes the class that constructs translation surfaces. This class is computationally similar to surfaces obtained through rotation, with the difference that only the generation of the control points P_{ij} has been modified, as described in Sect. 2.5.

Having the NURBS control points for both the generating and guiding curves, we construct the mesh for the points P_{ij} as follows.

$$P_{ij} = pdir_i + pgen_j - P_0$$

Here is the class that allows graphing this type of surface.

Code 5

```
mi_superficie = Nurbs_superficie3D_graph2(pts1=pdir,
    pts2=pgen, P=punto, pes1=w1, pes2=w2, nod1=x1,
    nod2=x2, p1=2, p2=2)
mi_superficie.grafica(pdir=False, pgen=False,
    c_pdir=True, c_pgen=True, enmaller=True,
    pts_ctrl=True, superficie=True)
```

As mentioned in previous lines, this class is very similar to the class for plotting revolution surfaces. The parameters do not change significantly; in fact, they remain the same, except for the addition of a single parameter $P = puntos$, which represents the common point of the two guiding and generating curves.

4.2 NURBS Revolution Surfaces

From 2D and 3D rotation, the concept is generalized to the rotation of points around other points.

Let the points P_{ij} of the generating curve in the XY plane be:

$$\begin{aligned} P_i &= (P_{ix}, 0, P_{iz}) \quad \text{in plane } XZ \\ R_j &= (R_{jx}, R_{jy}, 0) \quad \text{in plane } XY \end{aligned}$$

Graphical representation of P_i and R_j in a three-dimensional space with coordinates x, y, z.

Generating Curve:

$$[P_{ix}, 0, P_{iz}]$$

Rotation matrix defined by R_j:

$$\begin{bmatrix} R_{ix} & R_{jy} & 0 \\ -R_{jy} & R_{jx} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Generalized Rotation:

$$P_{ij} = \begin{bmatrix} P_{ix} & 0 & P_{iz} \end{bmatrix} \begin{bmatrix} R_{ix} & R_{jy} & 0 \\ -R_{jy} & R_{jx} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Resulting:

$$P_{ij} = (P_{ix}R_{jx}, P_{ix}R_{jy}, P_{iz})$$

Based on the curve product described previously, we will incorporate the class, considering the following aspects:

- The guiding curve (circle) rests in the XY plane, while the generating curve (arbitrary curve) lies on the XZ plane. For example, taking the guiding curve as the circle inscribed in an equilateral triangle and the generating curve as a semicircle inscribed in a hexagon, the rotation results in a sphere.
- A NURBS circle is defined on the basis of the XY plane and a NURBS semicircle in the YZ plane. Using the control points that generate these circles, the matrix product is applied to generate described in Sect. 4.2 to obtain a list of points forming a mesh. From these generated points, the NURBS surface is constructed from the resulting mesh.

Code 6

```
mi_superficie = Nurbs_superficie3D_graph(pts1=dir,
    pts2=gen, pes1=w1, pes2=w2, nod1=n1, nod2=n2, p1=2,
    p2=2)
mi_superficie.grafica(pdir=True, pgen=True, c_pdir=True,
     c_pgen=True, enmaller=True, pts_ctrl=True,
         superficie=True)
```

The class `Nurbs_superficie3D_graph` allows plotting surfaces generated by the rotation of two curves: a generating curve and a guiding curve. In the first line, the instance is initialized where the arguments `pts1` is a list of control points of the guiding curve; `pts2` is a list of control points of the generating curve; `pes1` and `pes2` are the respective weights of the two curves; the nodes of the respective curves are represented by `nod1` and `nod2`; and finally, the degrees of freedom of the curves, `p1` and `p2`.

In the second line, it allows for plotting the surface where some "features" can be controlled. For example:

- `pdir=True` indicates that the line passing through the control point list of the guiding curve is drawn.
- `pgen=True` indicates that the generating curve is drawn.
- `c_pdir=True` indicates that the guiding curve is drawn.
- `enmaller=True` and `pts_ctrl=True` specify that the mesh will be added along with all control points.
- Finally, `superficie=True` specifies that the surface will be added to the plot.

4.3 Tensor Product Surfaces with NURBS Curves

In Python, the class `Nurbs_superficie3D_graph1` has been created to plot NURBS surfaces, which instead of taking two NURBS curves as input, receives a matrix where each column represents a list of control points of a curve. In the first line, the instance is initialized where the argument `data` is the input matrix; `pes1` and `pes2` are the respective weights, one at the row level and the other at the column level; `nod1` and `nod2` are the nodes associated with the curves at the row and column level; `p1` and `p2` are the degrees of freedom of the curves at the row and column level. Finally, the range for the parameters `u` and `v` of the surface to be plotted is specified.

Code 7

```
mi_superficie = Nurbs_superficie3D_graph1(data=mi_data,
    pes1=w1, pes2=w2, nod1=n1, nod2=n2, p1=2, p2=2,
    ran_u=(u1, u2, ran_v=(v1, v2))
mi_superficie.grafica(enmaller=True, pts_ctrl=True,
    superficie=True, curv=True)
```

In the second line, the method `grafica` allows:

- `enmaller=True`: Add the point mesh.
- `pts_ctrl=True`: Draw the control points.
- `superficie=True`: Add the surface to the plot.
- `curv=True`: Draw the generated curves.

5 Results

The curves and surfaces obtained are based on the classes mentioned in the development section. In Fig. 2, NURBS-type circles constructed through control points obtained from polygons are shown. These circles have been generated using the class `Nurbs_curve2D_graph`.

5.1 Translation Surfaces in Python

The first approach is visualized using the class to build NURBS translation surfaces `Nurbs_superficie3D_graph2`. In this case, the hyperbolic paraboloid was obtained, as shown in Fig. 3. This result was achieved using two NURBS parabolas, where one is translated along the other.

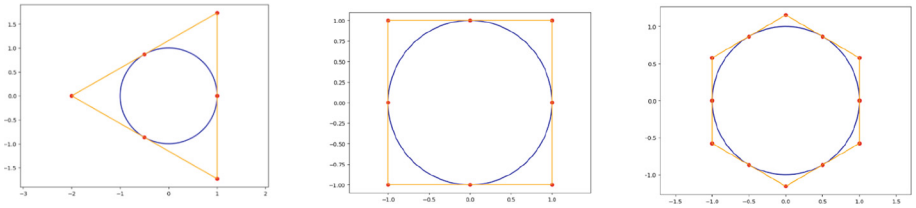

Fig. 2. Circles obtained from different polygons.

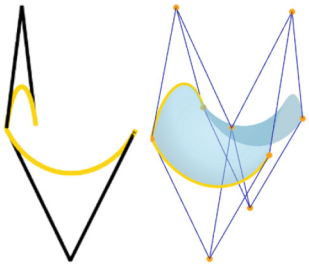

Fig. 3. Saddle surface

5.2 Revolution Surfaces in Python

In this approach, several NURBS surfaces are constructed based on rotational transformations.

In Fig. 4, a sphere is generated by rotating a semicircular NURBS curve, defined in the YZ plane, around a guiding NURBS curve inscribed in the XY plane.

Figure 5 illustrates the construction of a unit NURBS sphere using different base polygons. The resulting mesh configurations vary according to the selected polygon, confirming the dependency of surface topology on control geometry.

Figure 6 presents a dome formed by combining two semicircles inscribed in hexagons. The generated mesh demonstrates coherence and precision in the modeling process.

In Fig. 7, a similar procedure is applied; however, the resulting surface does not correspond to a dome, despite the similarity in the mesh structure. This suggests that the geometric outcome is sensitive to the specific combination of generating and guiding curves, enabling the construction of free-form surfaces through appropriate parametric selection.

Based on the overall modeling outcomes, additional surfaces are generated by systematically modifying the guiding and generating curves. Representative examples of these configurations are shown in Figs. 8, 9, 10, 11, 12, 13, 14 and 15.

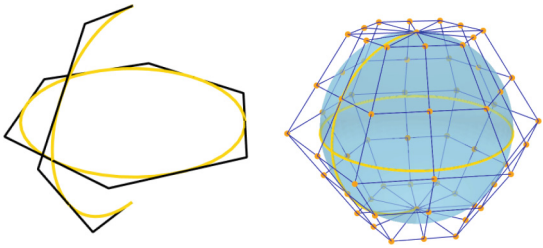

Fig. 4. GGenerating Curve: Circle inscribed in a hexagon and Guiding Curve: Semicircle inscribed in a hexagon.

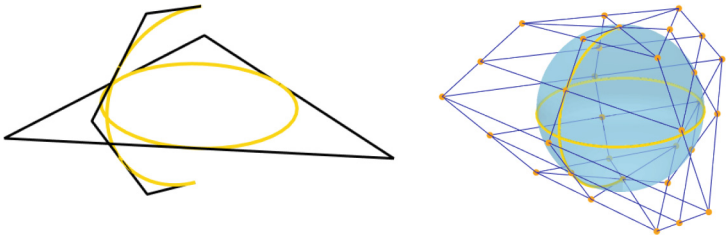

Fig. 5. Visualization of the unit NURBS sphere.

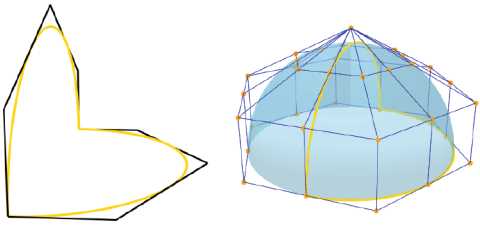

Fig. 6. Generating Curve: Semicircle with hexagon. Guiding Curve: Semicircle with hexagon.

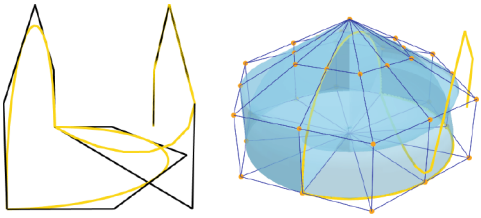

Fig. 7. Generation of Dome.

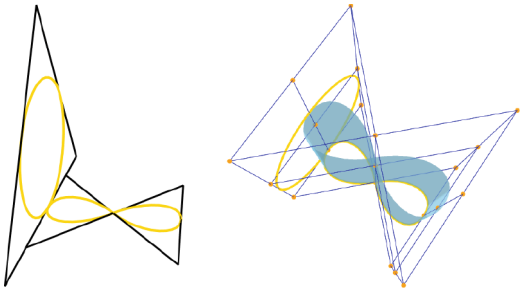

Fig. 8. Generating Curve: Lemniscate and Guiding Curve: Circle with Triangle.

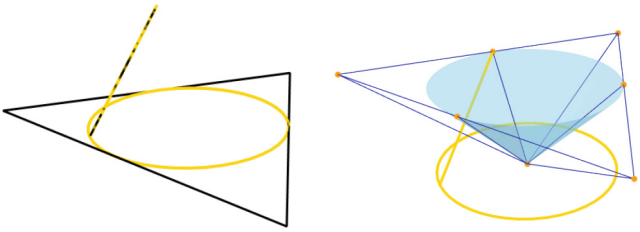

Fig. 9. Generating Curve: Circle Triangle and Guiding Curve: Line.

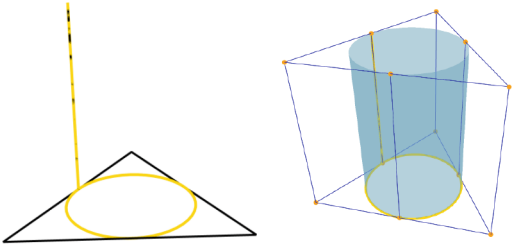

Fig. 10. Generating Curve: Circle Triangle and Guiding Curve: Line.

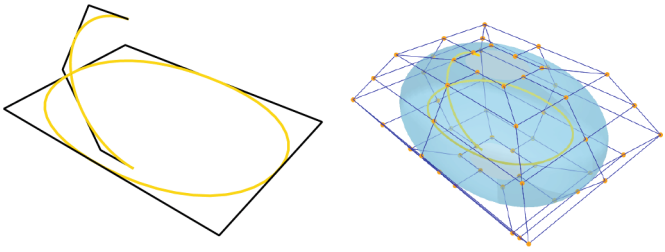

Fig. 11. Ellipsoid: Guiding Curve - an ellipse with 9 points. Generating Curve - a semi-ellipse from the hexagon.

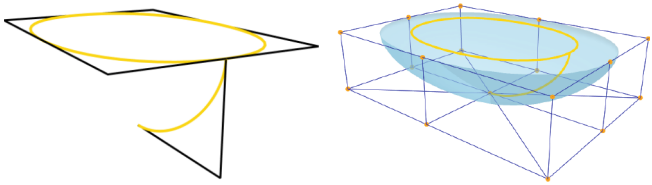

Fig. 12. Paraboloid: Guiding Curve - an ellipse. Generating Curve - a parabola with 3 control points.

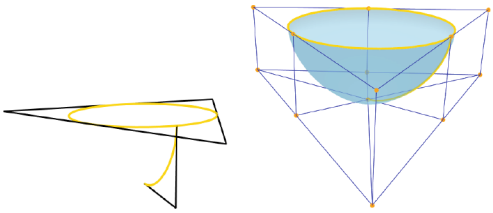

Fig. 13. Another Paraboloid: Same as the previous one, except the guiding curve is a circle derived from the triangle.

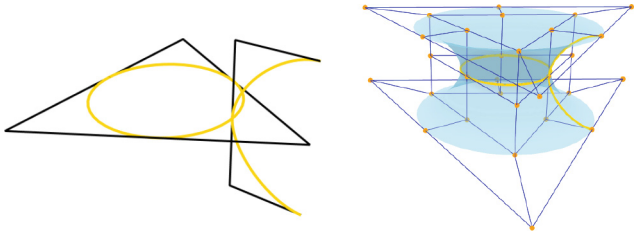

Fig. 14. Hyperboloid of one sheet.

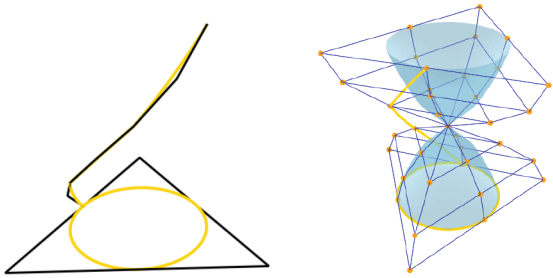

Fig. 15. Hourglass.

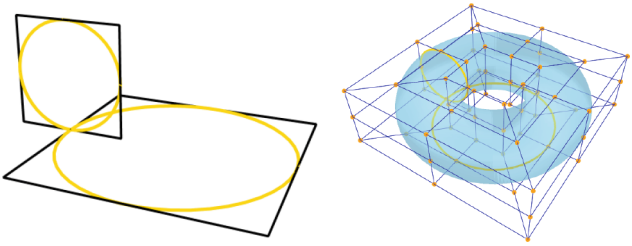

Fig. 16. Generating Curve: Circle with Square and Guiding Curve: Circle with Square.

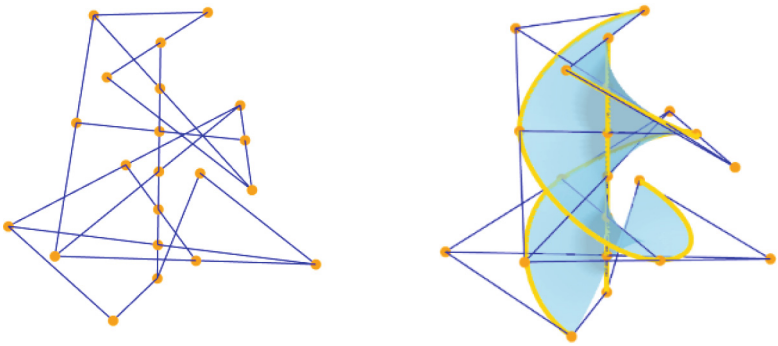

Fig. 17. Circular Helix.

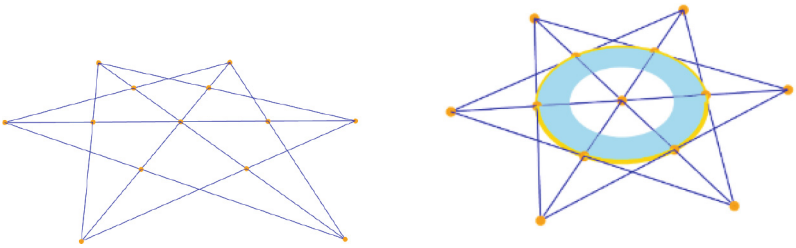

Fig. 18. Circle.

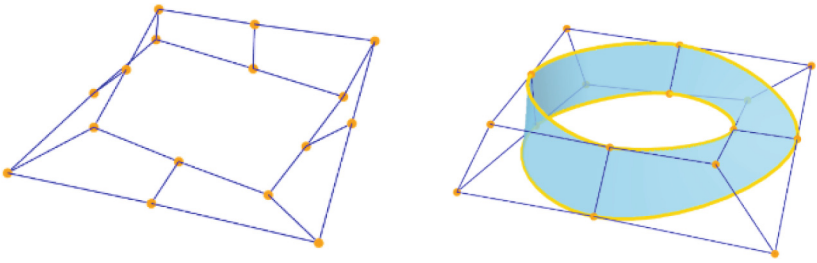

Fig. 19. Möbius Strip.

These results correspond to NURBS surfaces such as a cylinder, a cone, an ellipsoid, a paraboloid, and a hyperboloid of one sheet. From these constructions, most of the quadric surfaces have been generated.

Finally, a NURBS revolution torus will be visualized, based on its mathematical definition as the product of two circles. In this case, the construction process is similar to that of the previous NURBS surfaces. The result can be seen in Fig. 16.

5.3 Tensor Product Surfaces with NURBS Curves in Python

In this final approach, we will focus on surfaces bounded by NURBS curves, as explained in Sect. 4.3. This method allows for surface generation using the tensor product, creating a three-dimensional mesh of control points based on the base curve data. This ensures high precision and flexibility in geometric design. The implementation in Python, using the `Nurbs_superficie3D_graph1` class, facilitates the configuration of key parameters such as weights, nodes, and degrees of freedom of the curves. For this, a set of control points is needed to construct the NURBS surfaces, as shown in Figs. 17, 18 and 19.

6 Discussion and Perspectives

The results of this study demonstrate that Python, combined with a structured implementation of NURBS geometry, provides an effective environment for constructing, manipulating, and visualizing parametric surfaces. The three approaches explored—translation, revolution, and tensor product surfaces—offer distinct advantages depending on the geometric features of the problem.

From a performance standpoint, revolution surfaces yield the greatest precision and continuity when applied to symmetric and closed generating curves, making them suitable for classical geometries such as spheres, domes, and tori. Translation surfaces, though limited in variety, are computationally efficient and offer intuitive mesh control, making them especially appropriate for educational purposes and ruled surface generation. Tensor product surfaces, while more demanding computationally, enable the modeling of complex, free-form geometries, extending the versatility of the methodology.

A notable strength of this work lies in its pedagogical dimension. The Python-based implementation, integrated into interactive Jupyter Notebooks, supports conceptual understanding through real-time manipulation of control points, weights, and knot vectors. This fosters experiential learning and facilitates engagement with core topics in B-spline theory, linear algebra, and differential geometry. Additionally, the modular, object-oriented code structure encourages good programming practices consistent with contemporary computational mathematics instruction.

Looking ahead, this framework opens pathways for both educational innovation and further research. Potential extensions include the incorporation of real-time parameter tuning, integration with physical simulation engines, and quantitative evaluation of surface continuity. The methodology can also be adapted

to early undergraduate or secondary education, introducing geometric modeling concepts through interactive environments. Furthermore, combining this framework with symbolic computation or machine learning could enhance applications in shape optimization and surface fitting.

In comparison with existing literature, this implementation stands out for its integration of computational generality and instructional clarity. Unlike prior studies such as Tejada (2021), which focused on fixed-axis revolution surfaces, this work generalizes surface generation by supporting arbitrary generating and guiding curves. Similarly, in contrast to Cuartas and Prieto (2015), whose approach relies on 3D scanned data and optimization routines, the present methodology remains independent of specialized hardware and is fully synthetic and customizable, broadening both its practical and educational utility.

All implementations run efficiently on standard personal computers with at least 4 GB RAM, using Python 3.10 and the Jupyter Notebook environment.

Overall, the proposed framework contributes to bridging the gap between theoretical geometry, computational modeling, and educational practice, in alignment with contemporary needs in mathematics, design, and interdisciplinary education.

7 Conclusions and Future Work

This study demonstrated the feasibility and effectiveness of generating and visualizing Non-Uniform Rational B-Spline (NURBS) surfaces using three construction methods—translation, revolution, and tensor product—implemented in Python through the Jupyter Notebook environment. The proposed object-oriented framework, grounded in matrix operations and B-spline basis functions, enables precise geometric control and supports the creation of both classical and free-form surfaces with high flexibility.

Among the techniques evaluated, revolution surfaces exhibited superior performance in terms of smoothness and geometric continuity, especially for closed and symmetric base curves. Translation surfaces offered computational simplicity and intuitive mesh structure, making them suitable for educational and ruled-surface contexts. Tensor product surfaces, while more computationally intensive, facilitated the construction of intricate and compound geometries. The comparative use of these methods highlights trade-offs among accuracy, adaptability, and computational demand, providing a versatile toolkit for surface modeling.

From a pedagogical standpoint, the Python-based implementation proved highly effective in supporting the teaching and learning of advanced mathematical concepts. Its visual and interactive nature encourages intuitive understanding of rational parametrization, matrix transformations, and spline theory. By promoting hands-on experimentation, the framework fosters an inquiry-based learning environment that bridges theoretical content with computational practice—particularly valuable in undergraduate courses in geometry, computer graphics, and applied mathematics.

As a continuation of this work, future research may focus on several directions:

- Quantitative evaluation of surface continuity (e.g., G1/G2 smoothness) and computational performance under varying curve configurations;
- Integration of optimization algorithms, such as energy minimization or least-squares fitting, to refine control point placement;
- Expansion of the pedagogical toolkit, including auto-evaluation modules, parameter sliders, and multilingual interfaces to support international educational contexts;
- Extension to higher-dimensional or time-varying surfaces, enabling applications in animation, scientific visualization, and real-time CAD environments.

References

1. Fernández-Jambrina, L., Pérez-Arribas, F.: Developable surface patches bounded by NURBS curves. J. Comput. Math. **38**(5), 693–709 (2020). https://doi.org/10.4208/JCM.1904-M2018-0209
2. Taheri, A.H., Suresh, K.: Surface approximations using generalized NURBS. Eng. Comput. 1–19 (2021). https://doi.org/10.1007/s00366-021-01483-8
3. Guo, Z., Sun, W.: Study on a NURBS surface fitting algorithm. 35–40 (2022). https://doi.org/10.1109/AHPCAI57455.2022.10087689
4. Watts, A.: Algorithms for geometrical operations with NURBS surfaces (2022). https://doi.org/10.48550/arxiv.2210.13160
5. Tejada, P.: Construcción de esferas y toros de revolución mediante el uso de superficies B-Spline racionales. Universidad Nacional de Piura, Facultad de Ciencias, Escuela Profesional de Matemática, Trabajo de investigación (2021)
6. Cuartas, E., Prieto, F.: Optimización de la representación con superficies NURBS de imágenes en tiempo real. Revista Energía y Computación **15**(2), 31–37 (2015)
7. Cuervo, L., Cuervo, N., Cuervo, J.: Iniciando a programar con Python: Guía básica de programación. Editorial de la Universidad Pedagógica y Tecnológica de Colombia - UPTC, Bogotá (2018)
8. Perney, A., Bordas, S., Kerfriden, P.: NURBS-based surface generation from 3D images: spectral construction and data-driven model selection. J. Comput. Des. Eng. **10**, 1856–1867 (2023). https://doi.org/10.1093/jcde/qwad082
9. Prasad, A., Balu, A., Shah, H.S., Sarkar, S., Hegde, C., Krishnamurthy, A.: NURBS-diff: a differentiable programming module for NURBS. Comput. Aided Des. **146**, 103199 (2021)
10. Johansson, R.: Introduction to Computing with Python, pp. 1–24. Apress, Berkeley (2015). https://doi.org/10.1007/978-1-4842-0553-2_1
11. Lvov, M., Kruglyk, V.: Teaching algorithmization and programming using Python language. **20**, 13–23 (2014). https://doi.org/10.14308/ITE000493
12. Piegl, L., Tiller, W.: The NURBS Book. 2nd edn. Springer, Berlin (1997). https://doi.org/10.1007/978-3-642-59223-2
13. Cordero, J., Cortés, J.: Curvas y superficies para modelado geométrico. Ed. ilustrada. Ra-Ma, Librería y Editorial Microinformática (2002)
14. Prautzsch, H., Paluszny, M., Böhm, W.: Métodos de Bézier y B-splines. KIT Scientific Publishing, Karlsruhe (2005)
15. Lancaster, P., Salkauskas, K.: Curve and Surface Fitting: An Introduction. Academic Press, New York (1986)

16. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide, 4th edn. Academic Press, San Diego (1996)
17. Salomon, D.: Curves and Surfaces for Computer Graphics. Springer-Verlag, London (2006). https://doi.org/10.1007/0-387-28452-4
18. Bingol, O.R., Krishnamurthy, A.: NURBS-python: an open-source object-oriented NURBS modeling framework in Python. SoftwareX **9**, 85–94 (2019). https://doi.org/10.1016/j.softx.2018.12.005
19. Dunn, F., Parberry, I.: 3D Math Primer for Graphics and Game Development. Wordware Publishing Inc, Plano (2002)
20. López, A., Villa, A.: Geometría Diferencial. Editorial Clásica, Madrid (1994)
21. Cordero, L.A., Fernández, M., Gray, A.: Geometría diferencial de curvas y superficies con matemática. Addison-Wesley Iberoamericana, Wilmington (1998)