MODELING THE GYROID IN GEOGEBRA

Lingguo Bu

Southern Illinois University Carbondale

Abstract

The gyroid is a triply periodic minimal surface (TPMS) with intriguing geometric and aesthetic appeal. However, it is a challenging structure to model in GeoGebra. Starting with the $G*$ implicit equation, we first derive a parametric equation to model the gyroid unit surface patch in light of its intrinsic symmetries. Next, we demonstrate how JavaScript codes and GeoGebra Sequences can be used to create 3D scatterplots of the gyroid unit cell. Through this process, we showcase the interplay between GeoGebra's affordances and the computational complexity involved in gyroid modeling.

Keywords: Gyroid, 3D Modeling, Scripting, Sequences, Scatterplot

1 INTRODUCTION

The gyroid (G) is a triply periodic minimal surface (TPMS) discovered by Alan Schoen in 1968 and detailed in 1970 in his NASA technical report (Schoen, 1970, nd, 2012). Schoen's journey through gyroid mathematics is an engaging story for STEM educators – unexpected encounters, personal struggles, interdisciplinary collaborations, visualizations, and multimodal modeling. In recent years, the gyroid has found widespread applications in nature, art, architecture, manufacturing, and, most noticeably, 3D design and printing. Figure 1 shows two views of a five-foot gyroid assembled in August 2022 by a group of middle and high school students and mounted at Morris Library of Southern Illinois University Carbondale.

The gyroid is mathematically intriguing and aesthetically appealing. However, it is rather elusive to students because of its inherent geometric complexity. As is proven, the gyroid surface is the set of points (x, y, z) in the 3D Cartesian space that correspond to the real parts of its Enneper–Weierstrass representation (Sharma, 2013; Schoen, 1970, nd). Although it is possible to generate the exact surface of the gyroid using its Enneper-Weierstrass complex integrals (Gandy and Klinowski, 2000), it is pedagogically acceptable to use the level surface $G*$:

$$\cos(x)\sin(y) + \cos(y)\sin(z) + \cos(z)\sin(x) = 0 \tag{1}$$

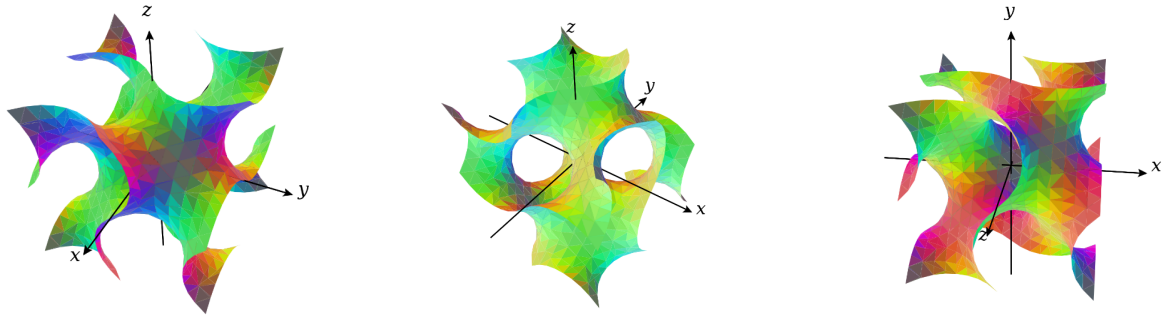
as a very close approximation of the gyroid G for educational and practical applications (Schoen, nd, Fig. E1.31).

(a) **(b)**

Figure 1. Two views of a five-foot spherical gyroid at Morris Library of Southern Illinois University Carbondale.

Unfortunately, using its usual tools, we cannot he $G*$ surface in GeoGebra. It remains an interesting challenge to model the basic structure of the gyroid in GeoGebra at the time of this article. It is also an opportunity for us to experiment with scripting using JavaScript and some foundational methods. For an immediate glimpse of the gyroid surface, we can alternatively enter the $G*$ equation as an `implicit surface` into `CalcPlot3D` at `https://c3d.libretexts.org/CalcPlot3D/index.html`. Figure 2 shows three different views of the gyroid surface generated by `CalcPlot3D`.

To model the gyroid in GeoGebra, we start with some pedagogical preliminaries and then discuss three different approaches and their characteristics in the following sections. We first derive a partial parameterization of the $G*$ equation and use it with GeoGebra `Surface()` to create an elementary patch of the gyroid. Then, we experiment with JavaScript coding and nested GeoGebra `Sequences` to construct a 3D scatterplot of the gyroid surface. In the process, we illustrate how the unique geometry of the gyroid structure aligns with the computational tools available in GeoGebra.



(a) **(b)** **(c)**

Figure 2. Three distinctive views of the gyroid $G*$ unit cell generated by CalcPlot3D.

2 PEDAGOGICAL CONSIDERATIONS

The gyroid is a geometrically complex shape. In the classroom, it can be initially challenging for students to grasp its twists and turns. This section discusses a few strategies for student engagement and mathematical accessibility, including the stories of the gyroid and its explorers, the integration of physical model building, and an overview of related GeoGebra commands and scripting techniques.

2.1 Stories about the Gyroid

Schoen's discovery of the gyroid exemplifies academic pursuits in science, technology, engineering, mathematics (STEM), and arts. To prepare students for gyroid-related explorations, we can ask them to read briefly about Schoen's lifelong journey into the study of the gyroid (`https://schoengeometry.com/e-tpms.html`). Schoen's site features diverse visual aids and personal reflections. Among the numerous lessons for STEM students and educators are his playful experiments with soap films, his initial struggles and conjectures, his extensive model building, his close interactions with mathematicians in various fields of mathematics and science, and his embrace of digital modeling and simulation technologies. After Schoen's passing in 2023, Hyde and Schroder-Turk (2024) published an essay celebrating Schoen's contributions to the STEM fields, offering a poignant look at the mathematical art of the gyroid and its stories. With a preliminary understanding of the gyroid and its five-decade-long emergence, we can further have students build a physical model using 3D-printed gyroid unit patches.

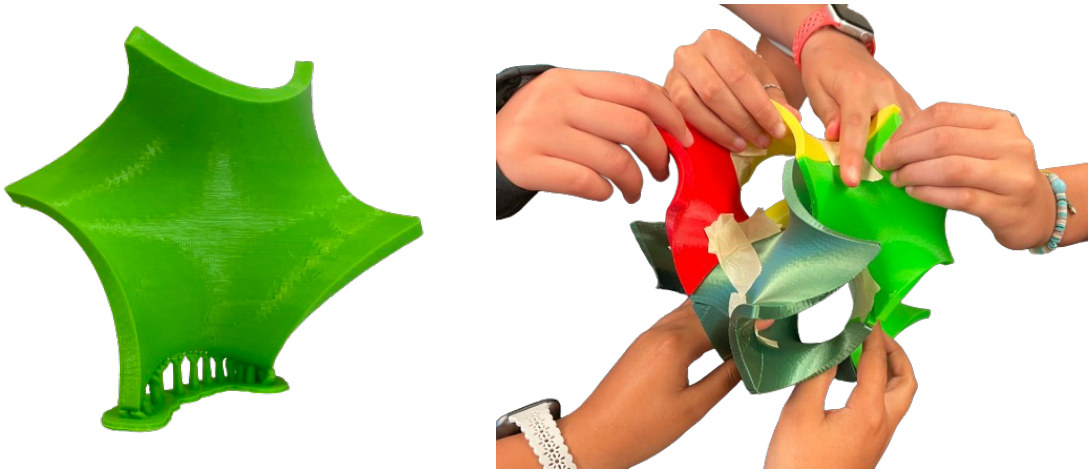
2.2 Physical Models

Physical model building provides an engaging and accessible entry into the mathematics and art of the gyroid. Using 3D printers, we can make gyroid unit patches (`https://www.thingiverse.com/thing:5394636`) for students to get a feel for the gyroid surfaces and further attempt a gyroid unit cell. These patches can indeed be exported from GeoGebra or generated using computing utilities that can handle equation (1) before GeoGebra modeling. Figure 3 shows a 3D-printed patch and a group of STEM campers assembling a gyroid unit out of eight patches. With adhesive tapes, students can usually follow the curved edges of the patches and build a gyroid unit cell with little assistance. Figure 4 presents three distinctive views of a gyroid unit cell assembled from eight unit patches.

2.3 GeoGebra Scripting

GeoGebra has flexible and powerful scripting capabilities that allow users to control the flow of computational events. Scripts come in two forms: GeoGebra Script and JavaScript, which can be attached to the `Scripting` tab of an object's properties in GeoGebra. The object that triggers a script, through `On Click` or `On Update`, can be a button or any other object in GeoGebra with a `Scripting` tab. Any GeoGebra command can be added to a GeoGebra Script, carried out in a sequence when triggered by the corresponding event. JavaScript codes are made up of general JavaScript commands and special GeoGebra API or methods (`https://wiki.geogebra.org/en/Reference:GeoGebra_Apps_API`).

Within GeoGebra, JavaScript methods can be called by `ggbApplet.method_name` with related parameters. Not all GeoGebra commands have corresponding JavaScript methods. Further, when a



(a) **(b)**

Figure 3. A 3D printed gyroid patch and student team work in building a gyroid unit cell.

(a) **(b)** **(c)**

Figure 4. Three views of a gyroid unit cell assembled out of eight unit patches.

GeoGebra command does have a corresponding JavaScript method, the syntax is slightly different in terms of spelling and parameters. Fortunately, GeoGebra commands can be passed to GeoGebra using the `ggbApplet.evalCommand()` method by JavaScript.

Figure 5 shows two buttons with some scripts. The button "Random Color and Style" has a GeoGebra Script, which sets the circle `c` to a random color, line style, and a random filling level. All these commands can be entered individually at GeoGebra's `Input` line. In the example, they are entered as a script to be triggered by a click on the button.

By contrast, the button "Remove Filling and Translate" has a JavaScript, which calls the method `setLineStyle` of `ggbApplet` to set the line style of circle `c` to 0, then calls the method `evalCommand` to pass the GeoGebra command `SetFilling` to remove the filling of the circle and finally runs a JavaScript `for` loop to call `evalCommand` five times and pass the `Translate` command to GeoGebra. Note that `evalCommand` takes a string which represents the GeoGebra commands.

Both GeoGebra Script and JavaScript codes can be utilized to perform complex computations. Ge-

oGebra scripts are executed sequentially, while JavaScript codes allow flexible control flow, including iterations and branches. The choice depends on the modeling task and other instructional objectives in a classroom setting.

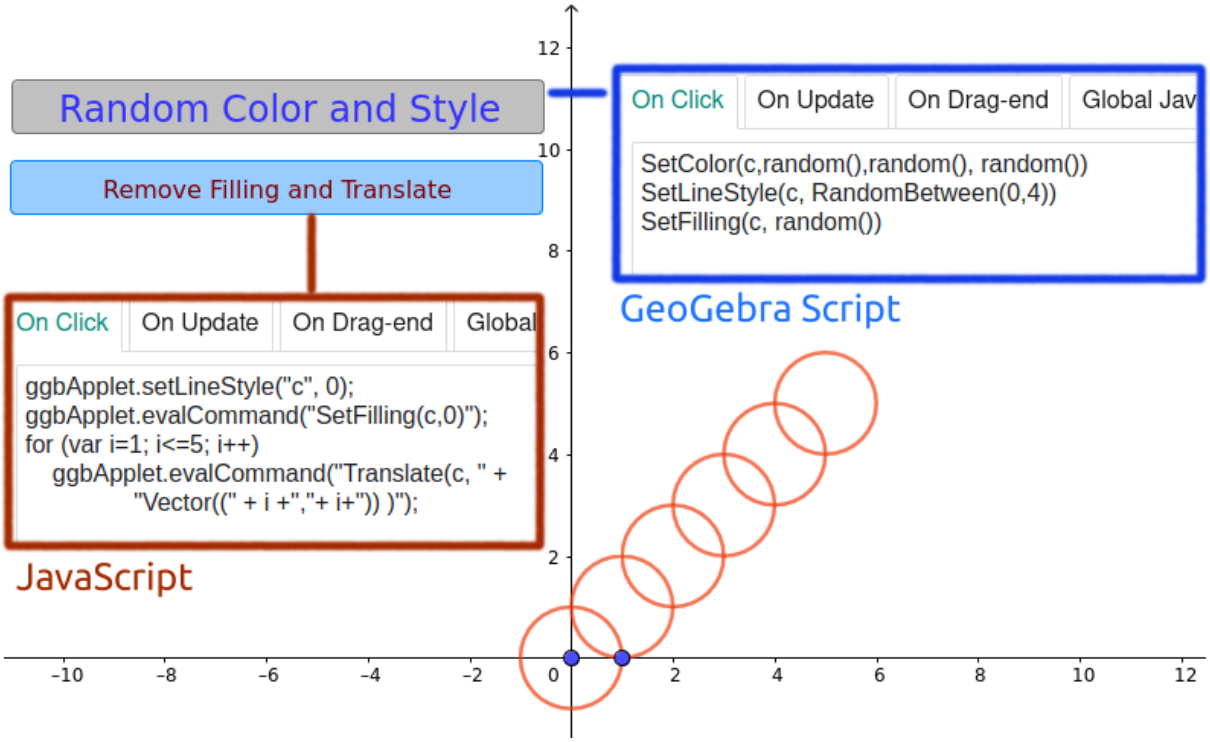


Figure 5. Both GeoGebra Script and JavaScript can be attached to the `Scripting` tab of an object.

2.4 Plotting Surfaces in GeoGebra

A surface in the 3D space can be generally described by an implicit equation $F(x, y, z) = 0$. For example, $x^2 + y^2 + z^2 = 1$ represents a unit sphere. GeoGebra can handle various implicit surfaces. If a surface can be described in an explicit equation in the form of $z = f(x, y)$, GeoGebra can always graph it when it is well-defined. In advanced mathematics, we frequently encounter surfaces that do not have an explicit form or have an implicit form that is technically difficult to plot. We then attempt to derive a parametric form:

$$\vec{r}(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}$$

where u and v are parameters.

For example, a unit sphere centered at the system origin $(0, 0, 0)$ can be parameterized as:

$$\vec{r}(u, v) = \begin{pmatrix} \cos(u)\sin(v) \\ \sin(u)\sin(v) \\ \cos(v) \end{pmatrix}$$

where $0 \leq u \leq 2\pi$ and $0 \leq v \leq \pi$.

In GeoGebra, parametric equations can be entered as a `Surface()` command. The unit sphere above can be plotted by entering the following command:

```
Surface(cos(u)sin(v),sin(u)sin(v),cos(v),u,0,2pi,v,0,pi)
```

In general, once we have a parametric form for a 3D surface, obtaining a graph in GeoGebra is relatively easy. For example, the Enneper minimal surface can be plotted in GeoGebra using its parametric form (`https://www.geogebra.org/classic/bjbkbqvy`). However, it can be challenging to derive a parameterization for a complex surface, as is the case for the gyroid.

3 Partial G* Parameterization

With a preliminary understanding of the gyroid and GeoGebra tools, we now consider the implicit equation (1), which can indeed be partially parameterized for plotting in GeoGebra over part of the (x, y) domain. Starting with

$$\cos(x)\sin(y) + \cos(y)\sin(z) + \cos(z)\sin(x) = 0,$$

we first collect all the terms containing z:

$$\cos(y)\sin(z) + \cos(z)\sin(x) = -\cos(x)\sin(y).$$

Assuming $\cos(y) \neq 0$, we can then divide the above by $\cos(y)$ for the following:

$$\sin(z) + \frac{\cos(z)\sin(x)}{\cos(y)} = -\frac{\cos(x)\sin(y)}{\cos(y)}. \tag{2}$$

For simplicity, let

$$A = \frac{\sin(x)}{\cos(y)}, \quad B = \frac{\cos(x)\sin(y)}{\cos(y)}.$$

Then, equation (2) becomes

$$\sin(z) + A\cos(z) = -B$$

or

$$\sin(z) = -B - A\cos(z),$$

which yields, upon squaring,

$$\sin^2(z) = (B + A\cos(z))^2.$$

Further, applying the identity $\sin^2(z) + \cos^2(z) = 1$, we have

$$1 - \cos^2(z) = (B + A\cos(z))^2,$$

which is a quadratic equation in $\cos(z)$. If we let $w = \cos(z)$ for readability, then we have

$$\begin{aligned} 1 - w^2 &= (B + Aw)^2 \quad \text{or} \\ (A^2 + 1)w^2 + (2AB)w + (B^2 - 1) &= 0, \end{aligned} \tag{3}$$

where A and B are constants, relatively speaking. Solving equation (3) above for w using the quadratic formula, we get:

$$\begin{aligned} w &= \frac{-2AB \pm \sqrt{(2AB)^2 - 4(A^2+1)(B^2-1)}}{2(A^2+1)} \\ &= \frac{-AB \pm \sqrt{A^2 - B^2 + 1}}{A^2+1}. \end{aligned} \tag{4}$$

Substituting $A = \frac{\sin(x)}{\cos(y)}$ and $B = \frac{\cos(x)\sin(y)}{\cos(y)}$ into equation (4), we get:

$$\begin{aligned} w &= \frac{-\left(\frac{\sin(x)}{\cos(y)}\right)\left(\frac{\cos(x)\sin(y)}{\cos(y)}\right) \pm \sqrt{\left(\frac{\sin(x)}{\cos(y)}\right)^2 - \left(\frac{\cos(x)\sin(y)}{\cos(y)}\right)^2 + 1}}{\left(\frac{\sin(x)}{\cos(y)}\right)^2 + 1} \\ &= \frac{-\sin(x)\cos(x)\sin(y) \pm \sqrt{T}}{\sin^2(x) + \cos^2(y)}, \end{aligned} \tag{5}$$

where $T = \sin^2(x)\cos^2(y) - \cos^2(x)\sin^2(y)\cos^2(y) + \cos^4(y)$.
As $w = \cos(z)$, we can find z from equation (5) under the initial assumption:

$$z = \cos^{-1}\left(\frac{-\sin(x)\cos(x)\sin(y) \pm \sqrt{T}}{\sin^2(x) + \cos^2(y)}\right). \tag{6}$$

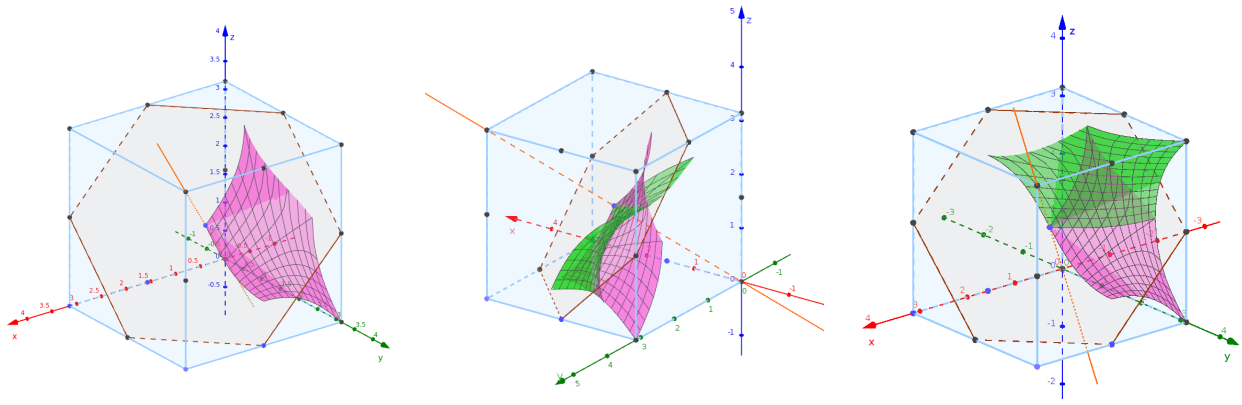
Equation (6), with either $+$ or $-$ for the square root, can be used with the GeoGebra `Surface()` command. However, it does not yield a complete gyroid surface because of the constraints imposed on it. Upon close analysis, the author realized that we can plot part of the gyroid surface over a selected part of the domain, and then take advantage of its inherent symmetry to construct a unit patch. Given adequate computing power, the unit patch can be used to build a gyroid unit cell.

We can start with a π-cube in the first octant of the 3D Cartesian System (Figure 6), then run the GeoGebra `Surface()` command over $0 < u < \frac{\pi}{2}$ and $\frac{\pi}{2} < v < \pi$, as shown in Listing (1), to create part of the gyroid surface (Figure 6a), which we refer to as the 'elementary patch' or EP. The surface in Figure 6a is part of the solution set over the chosen $(x,\, y)$ domain. We do not need the section above the plane $z = \frac{\pi}{2}$. However, since there is no easy way to remove it, we can keep it for subsequent geometric transformations, leading to some visualizations overlapping, as shown in Figure 6c.

```
Surface(u,v,arccos((-sin(u)cos(u)sin(v)+sqrt((sin(u))^2(cos(v))^2-
(cos(u))^2(sin(v))^2(cos(v))^2+(cos(v))^4))/
((sin(u))^2+(cos(v))^2)),u,0,pi/2,v,pi/2,pi)
```

Listing 1. Surface() command for GeoGebra input.

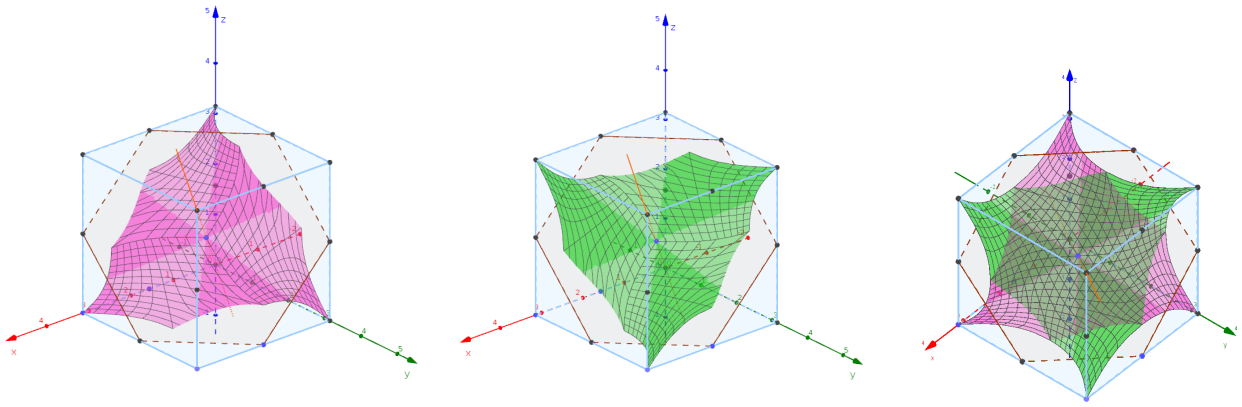
Next, we need to find the mirror image $(EP)'$ of the elementary patch EP to the hexagonal plane containing the six midpoints of the π-cube edges, which is shown in Figure 6b. The mirror image $(EP)'$ can then be rotated 60° to get into position on the gyroid surface, as shown in Figure 6c. The overlapping part is due to the extra section produced in Figure 6a.



(a) **(b)** **(c)**

Figure 6. The elementary G* patch and related geometric transformations.

To construct the gyroid unit patch, which is housed in the π-cube, we can rotate the EP and $(EP)'$ pair in Figure 6c $120°$ and then $240°$ around the vector $< \pi, \pi, \pi >$. Alternatively, we could rotate EP and $(EP)'$ separately to construct the gyroid unit patch. Figure 7a shows three copies of the EP patch. Figure 7b shows three copies of the $(EP)'$ patch. Figure 7c shows the gyroid unit patch composed of six copies of the elementary patch up to rotation and reflection.



(a) **(b)** **(c)**

Figure 7. The gyroid unit patch comprises six EP copies up to reflection and rotation.

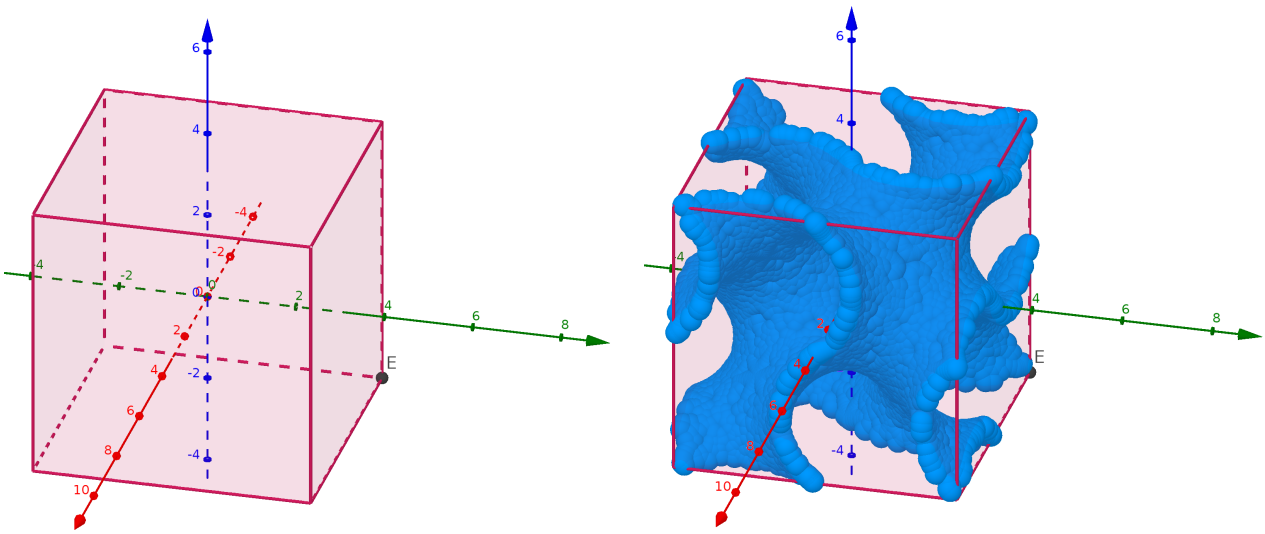
It runs very slowly because of the computation involved to generate the unit patch in Figure 7c, GeoGebra 3D Graphics runs very slowly. Theoretically, eight copies of the gyroid unit patch can be assembled, up to translation and rotation, for a gyroid unit cell as shown in Figure 2. At the time of this article, Java runs too slow on an average computer for such an attempt. For a thought experiment, we could imagine eight unit patches, each housed in a π-cube. Then, we could stack up the eight π-cubes, with translation and rotation, to build a gyroid unit cell housed in a 2π-cube.

4 Scatterplot of G* Using JavaScript

The $G*$ surface represents the solution set of Equation (1) in the 3D Cartesian System. Without an efficient and complete parametric equation, we can resort to a brutal-force method: systematically

scanning a chosen domain to look for points close to the gyroid surface. The $G*$ surface is triply periodic along all three dimensions with a period of 2π in each dimension. Thus, we can focus, for example, on a 2π-cube centered at the system origin $(0,\ 0,\ 0)$ to construct a cubic unit cell of the gyroid. Figure 8a shows the centered 2π-cube that contains a gyroid unit cell to be discovered.

To scan the 2π-cube, we must write some JavaScript code in GeoGebra to implement three nested loops at a chosen resolution. The code can be attached to the `Object Properties` of a `Button` in GeoGebra Graphics under `[Scripting, On Click]`. Figure 11 in Appendix B illustrates the user interface in GeoGebra Classic.



(a) **(b)**

Figure 8. The centered 2π-cube and the scatterplot of a gyroid unit cell.

As shown in code Listing 2, we use three `for` loops to iterate along the three dimensions at a `stepValue` determined by the `resolution`. For each candidate $(i,\ j,\ k)$, we call the `gyroidQ()` function, which calculates a `gValue` and compares it with a certain tolerance (0.001 in the example). If the `gValue` is greater than the tolerance, `gyroidQ()` returns `false` and allows the script to proceed to the next candidate. On the other hand, if the `gValue` is less than the tolerance, `gyroidQ()` returns `true`, indicating a point on the gyroid surface.

Then, the `ggbApplet.evalCommand()` is invoked to pass the point $(x,\ y,\ z)$ to GeoGebra 3D graphics for a visual point. In the code shown in Listing 2, `maxNum` defines the lower and upper bounds of the loops; it is adjusted by 0.001 to avoid some unwanted boundary behaviors of the `evalCommnd()`. Any other small number may work just as well. Of course, the user can also adjust the tolerance. At a `resolution` of 100 over $(0, \pi)$, the `stepValue` is $\frac{\pi}{100}$. About eight million candidates are to be tested in the 2π-cube, yielding 6084 points close to the $G*$ surface, as shown in Figure 8b.

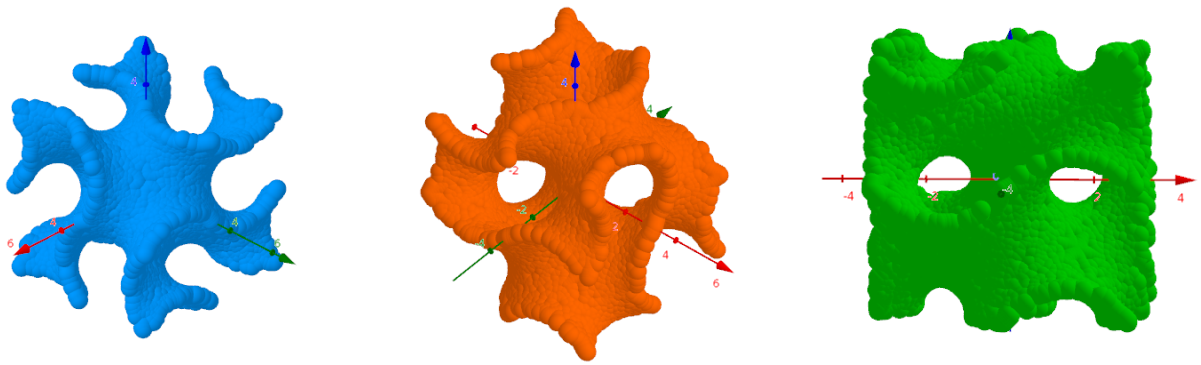
GeoGebra slows significantly at a `resolution` of 100 or higher because of the amount of computation involved. A lower resolution of less than 100 should be used for initial experiments. Once a

3D scatterplot is completed within the 2π-cube, we have a gyroid unit cell, which can be used theoretically to tessellate the infinite surface of the gyroid. Within GeoGebra 3D Graphics, the scatterplot can be rotated for some distinctive views of the gyroid (Figure 9).

```
// Return true if given point (x, y, z) is close to the G surface.
function gyroidQ(x, y, z) {
    var gValue = Math.cos(x) * Math.sin(y) + Math.cos(y) * Math.sin(z) +
        Math.cos(z) * Math.sin(x);
    if (Math.abs(gValue) <= 0.001) {
        return true;
    } else {
        return false;
    }
}

// Test all the given points along x, y, z in
// (-PI, PI) based on the given resolution.
var count = 0;
var numOfCycle = 1;
var resolution = 100;
var maxNum = Math.PI * numOfCycle - 0.001;
var stepValue = Math.PI / resolution;
for (var i = -maxNum; i < maxNum; i += stepValue) {
    for (var j = -maxNum; j < maxNum; j += stepValue) {
        for (var k = -maxNum; k < maxNum; k += stepValue) {
            if (gyroidQ(i, j, k)) {
                count++;
                ggbApplet.evalCommand("A_{" + count + "}=(" + i + "," + j +
                    "," + k + ")");
            }
        }
    }
}
```

Listing 2. JavaScript code to find points close to the gyroid surface.



(a) **(b)** **(c)**

Figure 9. Three views of the 3D scatterplot of the gyroid unit cell generated using JavaScript.

5 SCATTERPLOT OF G* USING SEQUENCES

Alternatively, we can use the GeoGebra `Sequence()` command (North American GeoGebra Journal Staff, 2013) recursively to scan a 2π-cube for points that are close to the gyroid surface at a chosen tolerance. The algorithm is similar to the JavaScript loops. However, GeoGebra slows dramatically because of the amount of intermediate data points generated. Therefore, we should start with a very low resolution.

As shown in Listing 3, we start with a resolution of 16 data points over $(0, \pi)$ and a tolerance of 0.1. During initial explorations, a low resolution, such as 3 or 5, could be used for prompt system responses. The user-defined function `gyroid(x, y, z)` returns the value of the triple-variable $G*$ function, which filters the points. The list `pointList` contains a nested list of preliminary points in a reference cube determined by `resolution`. All the triplets in `pointList` are then mapped to a 2π-cube and flattened to a new list of points stored in `pointListPi`. All the points in `pointListPi` are subsequently tested using `gyroid()` against the `tolerance`.

```
resolution = 16
tolerance = 0.1
gyroid(x,y,z) = cos(x)sin(y) +cos(y)sin(z) +cos(z)sin(x)

pointList = Sequence(Sequence(Sequence((i,j,k),i,-resolution, resolution),j,
    -resolution,resolution), k,-resolution,resolution)

pointListPi = Flatten(pointList(pi - 0.1) /resolution)

gListTemp = Sequence(If(abs(gyroid(x(Element(pointListPi,k)), y(Element(
   pointListPi,k)), z(Element(pointListPi, k)))) < tolerance, Element(
   pointListPi, k)), k, 1, Length(pointListPi))

gList = RemoveUndefined(gListTemp)
```

Listing 3. GeoGebra commands for input at command line.

If a point is a candidate for the gyroid surface, it is directly recorded in a new list named `gListTemp`; otherwise, an `undefined` point is recorded by GeoGebra. Then, we can collect all the qualifying points to another list `gList` after running `RemoveUndefined()` in `gListTemp`. If we hide all the other lists of points except for `gList`, we obtain a 3D scatterplot for the gyroid surface, as shown in Figure 10. It can be manipulated in GeoGebra 3D Graphics for various views.

With `resolution = 16` and `tolerance = 0.1`, there are about 2305 qualifying points found in the centered 2π-cube. As shown in the GeoGebra command listing, we also adjusted the lower and upper bounds by 0.1 as in "$(\texttt{pi} - 0.1)/\texttt{resolution}$," to remove a few points on the boundaries for aesthetic reasons.

6 CONCLUSION

The gyroid is a mathematically intriguing and aesthetically pleasing structure (Nervous System, nd) that has gained relevance in STEM research and development. It can be approached using the $G*$ implicit equation for practical applications and school mathematics. Using a partial parametric equation

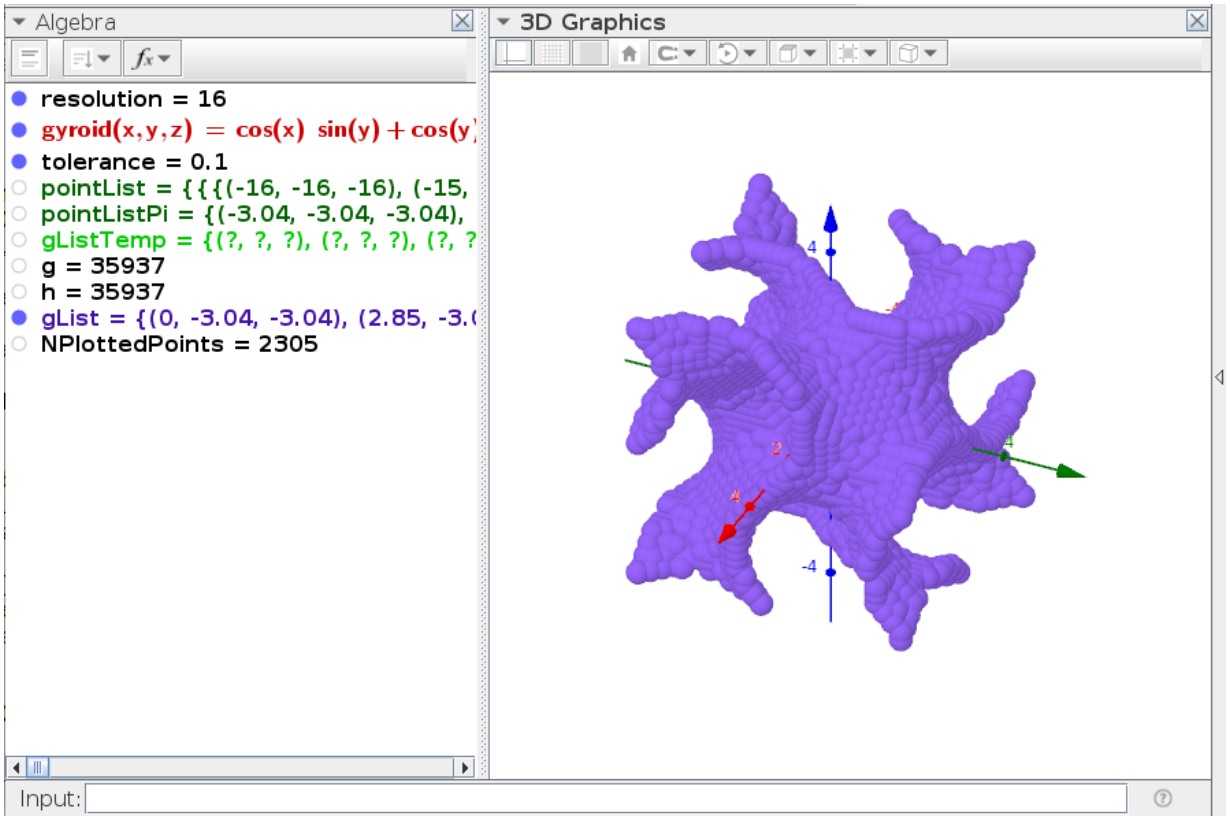


Figure 10. Scatterplot of the gyroid unit cell generated using GeoGebra sequences.

of $G*$, we can graph an elementary patch of the gyroid surface and, taking advantage of its intrinsic symmetries, construct a unit patch in a 2π-cube. With adequate computing power, the unit patch can be subsequently used to tessellate the infinite triply periodic minimal surface (TPMS) of the gyroid.

Alternatively, we can use JavaScript or GeoGebra's `Sequence()` iterations to scan a 2π-cube for qualifying points, creating a 3D scatterplot to approach the gyroid unit cell. The 3D scatterplot is visually similar to the beaded model for the gyroid surface (Chuang et al., 2012). Both JavaScript codes and GeoGebra `Sequence()` tend to slow down at high resolutions due to the significant computational demand of the brutal-force algorithm. However, at lower resolutions, either method can effectively generate a complete gyroid unit cell for visual exploration. Meanwhile, we have demonstrated that GeoGebra has powerful 3D graphic tools and scripting capabilities, which can be leveraged to implement both sophisticated mathematical modeling and computational reasoning.

As GeoGebra evolves with more computational resources available, we anticipate being able to generate the exact gyroid surface using the complex integrals of its Enneper–Weierstrass representation (Sharma, 2013), as described in (Gandy and Klinowski, 2000). This will allow us to delve deeper into the rich mathematics of the gyroid and explore its implications for design and real-world applications. For immediate access to the gyroid in GeoGebra, readers can refer to Appendix A for URL links to the online modules discussed in the article.

REFERENCES

Chuang, C., Jin, B. Y., Wei, W. C., and Tsoo, C. C. (2012). Beaded realization of canonical p, d, and g triply periodic minimal surfaces. In *Bridges Conference Proceedings*, pages 503–506, Towson, MD.

Gandy, P. J. F. and Klinowski, J. (2000). Exact computation of the triply periodic g (gyroid) minimal surface. *Chemical Physics Letters*, 321(5-6):363–371.

Hyde, S. T. and Schroeder-Turk, G. E. (2024). Alan hugh schoen. *Physics Today*.

Nervous System (n.d). Gyroid.

North American GeoGebra Journal Staff (2013). Lists and sequences. *North American GeoGebra Journal*, 9(1):25–30.

Schoen, A. H. (1970). Infinite periodic minimal surfaces without self-intersections. Technical Report TN D-5541, NASA.

Schoen, A. H. (2012). Reflections concerning triply-periodic minimal surfaces. *Interface Focus*, 2(5):658–668.

Schoen, A. H. (n.d.). Triply periodic minimal surfaces.

Sharma, R. (2013). The weierstrass representation always gives a minimal surface. *Rose-Hulman Undergraduate Mathematics Journal*, 14(1).

Lingguo Bu lgbu@siu.edu, is a Professor of Mathematics Education at Southern Illinois University Carbondale in the School of Education. He is interested in multimodal STEM modeling and simulations in the context of K–16 mathematics education and teacher development, including integrating GeoGebra and 3D design and printing.

APPENDIX A - URLS TO ONLINE MODULES

1. GeoGebra Scripting Demonstration:
   `https://www.geogebra.org/classic/dadazupv`
2. Gyroid as a Parametric Surface:
   `https://www.geogebra.org/classic/upkph74c`
3. Gyroid Using JavaScript:
   `https://www.geogebra.org/classic/xryxm5ur`
4. Gyroid Using Recursive Sequences:
   `https://www.geogebra.org/classic/ygpm5dx4`
5. Plotting the Enneper Surface:
   `https://www.geogebra.org/classic/bjbkbqvy`

APPENDIX B - USING JAVASCRIPT IN GEOGEBRA

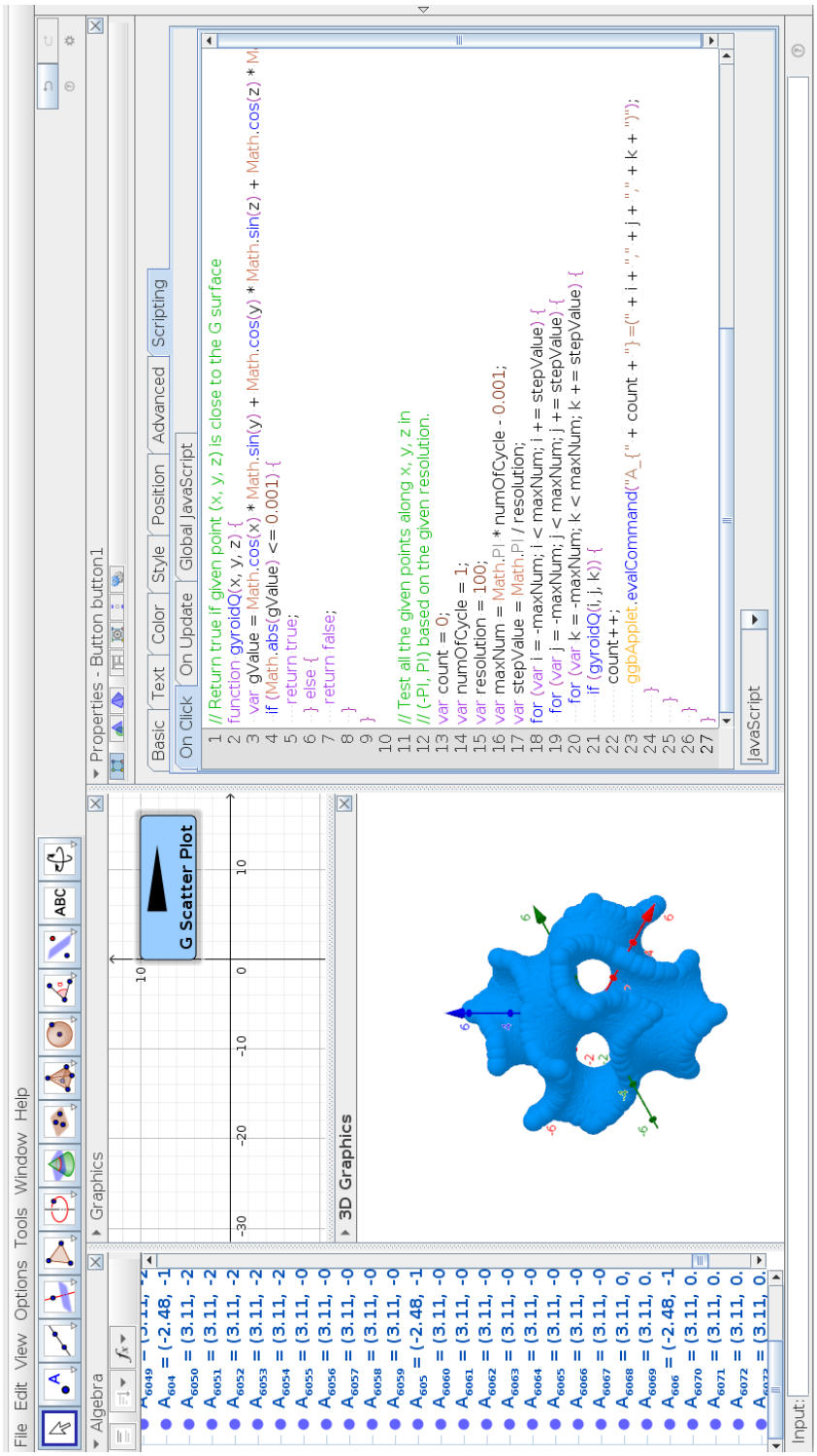


Figure 11. The JavaScript code is attached to the `Scripting` tab of a `Button` to implement the nested iterations and search for qualifying points on the gyroid surface.