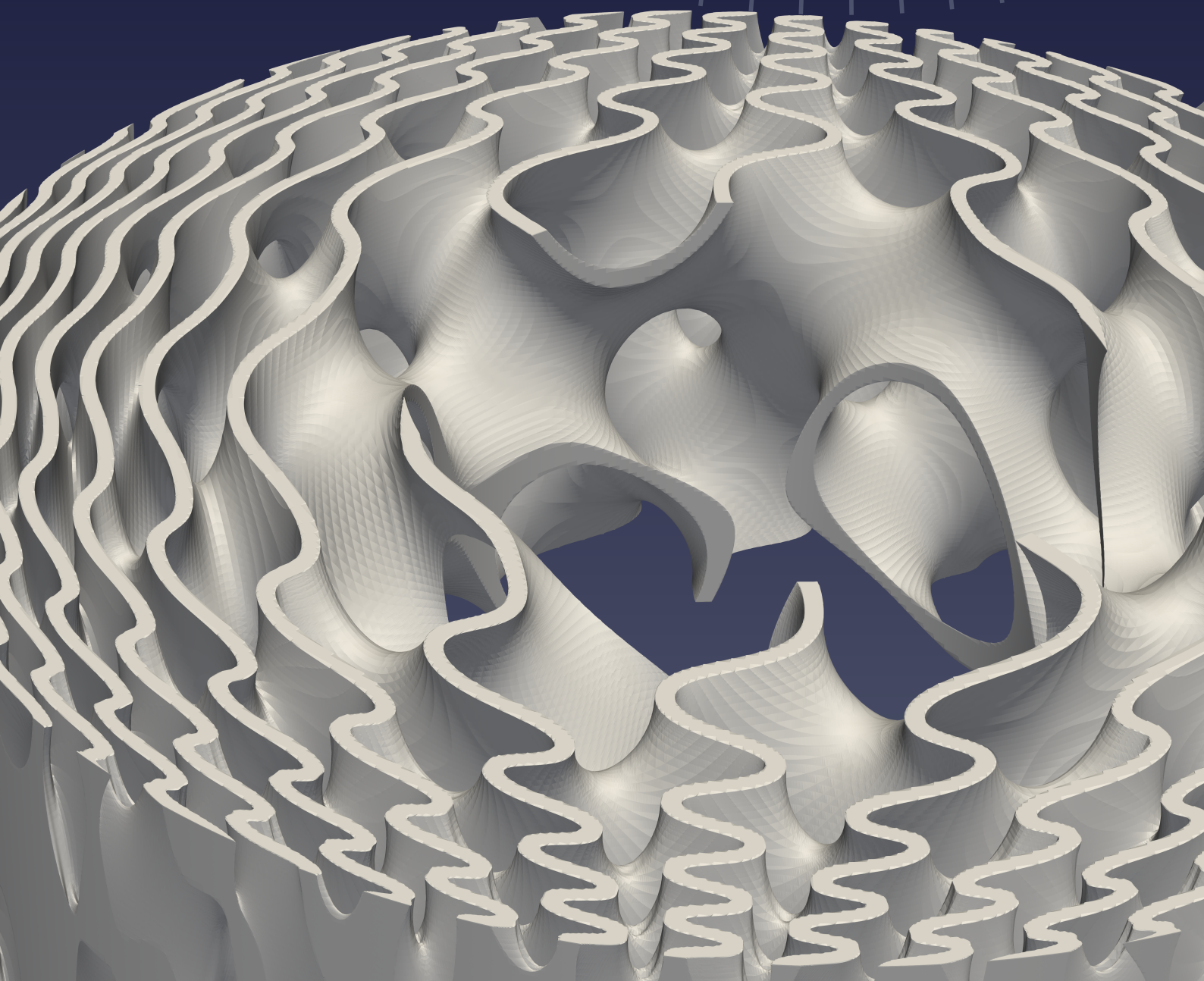
Parametric Design of Variable-Density Gyroid Lattice for a Hybrid Rocket Armoured Grain Fuel

Dan Borcea



Abstract

This project originated as a personal initiative during a trial period at SPLAB, Politecnico di Milano, while exploring potential thesis topics. It was motivated by both curiosity and the possibility that the work could later benefit Skyward Experimental Rocketry, the university's rocketry association, as well as serve as a foundation for future developments. Initially implemented in MATLAB, the project was later revisited after graduation, refined, and fully rewritten in Python.

The work brings together two personal passions: programming and 3D printing. The main objective is to generate a gyroid lattice—a type of periodic minimal surface—entirely through Python code, without relying on external modeling software. This approach enables rapid export of the generated structure directly into a slicer for 3D printing.

Unlike traditional slicers, which often offer gyroid infill as a fixed-density option, this implementation allows for a radially variable infill density, opening up new possibilities for material optimization and structural performance. The project not only provided a hands-on opportunity to deepen algorithmic and geometric modeling skills, but also aimed to create a reusable tool for prototyping and design in engineering applications.

Contents

List of Figures

1 Introduction

The armoured grain is a novel type of propellant for hybrid rocket engines, developed by SPLAB at Politecnico di Milano and later tested and implemented by Skyward Experimental Rocketry in their Lyra project, which participated in the EUROC 2024 competition. As a member of Lyra's propulsion team, I was directly involved in this development.

To provide some background, hybrid rocket motors operate by combining a solid fuel—typically shaped as a cylindrical grain—with a liquid oxidizer injected at high pressure into the central port of the grain. The oxidizer reacts with the combustible material of the grain, which burns from the inside out, producing thrust. Solid fuels used for this purpose are generally made of either plastic (e.g., ABS) or paraffin wax.

Each material offers trade-offs: plastic-based fuels provide high mechanical strength and combustion efficiency but suffer from a low burning rate, resulting in reduced thrust. In contrast, paraffin wax offers a significantly higher thrust thanks to its surface regression behavior—melting and being stripped away before fully evaporating and reacting—but it is mechanically fragile and structurally unstable under stress.

The armoured grain addresses this limitation by combining both materials: it consists of an internal structural lattice, typically made of ABS, embedded within

a volume of paraffin wax that is cast around it. The lattice not only reinforces the grain mechanically but also maintains combustion performance.

Among the various possible lattice structures, the gyroid has emerged as a particularly effective choice. Its triply periodic minimal surface (TPMS) geometry distributes mechanical stress efficiently and, crucially, features a single continuous surface with no self-intersections, simplifying the paraffin casting process. Fig. 1 shows the structure of one of Skyward's armoured grains.

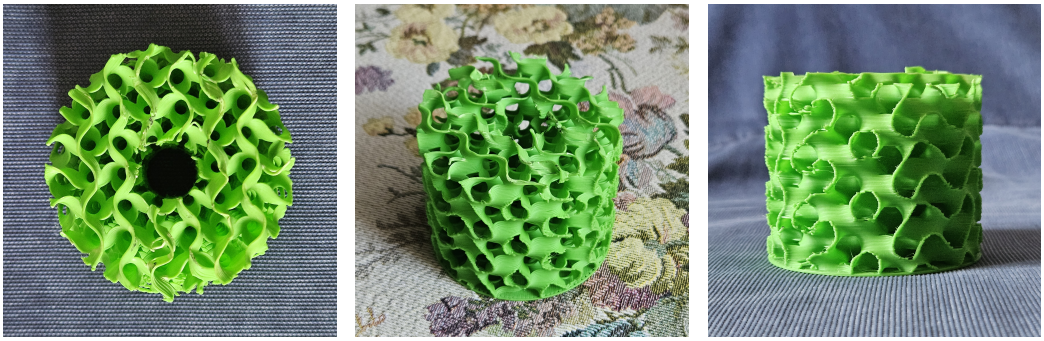


Figure 1: Some views of a 10% infill 3D printed gyroid.

By varying the local infill density—the proportion of lattice material within a given volume—it is possible to optimize the grain's properties, balancing structural support, fuel mass distribution, and burning rate. This introduces an additional design degree of freedom, allowing engineers to tailor performance based on mission requirements.

This context motivated the development of a custom Python tool, capable of generating gyroid lattices with radially variable infill density, designed specifically for 3D printing applications without relying on external CAD or mesh software.

2 Code description

The code was designed to be highly configurable yet easy to use. Its structure allows both quick experimentation and deeper customization, depending on the user's level of familiarity with the algorithm. In most cases, only the main script needs to be edited, as it contains all the essential input parameters required to generate a specific geometry. Additional configuration options, mainly related to algorithmic details and secondary behaviors, are stored in a separate script `config.py` located in the same directory. This design keeps the workflow simple and minimizes the risk of unintentional modifications to the underlying com-

putational logic.

The code offers extensive configurability over the geometry of the grain. Users can define key parameters such as the inner and outer radius, total length, and wall thickness. Moreover, the gyroid's local infill density can be configured to vary linearly along the radius, between user-specified initial and final values. This variation allows the designer to control the distribution of solid material within the propellant and therefore tune the structural stiffness and burning characteristics. It is also possible to impose a constant-density region at the beginning of the grain—both the extent and the density value are fully configurable—so that the gyroid maintains uniform infill over a given initial portion before transitioning to a variable pattern.

Beyond the internal lattice, the final geometry can include additional optional features such as inner and outer walls or an annular base attached to one end of the grain. These features can be individually enabled or disabled, allowing the user to adapt the model to specific structural or manufacturing constraints. Wall thickness and base dimensions are configurable as well, ensuring full control over the external shape of the final component.

```
7
8   # --- INPUTS - Geometry ---
9   GEO["radius"]           = 50                        # External radius
10  GEO["port"]             = 20                         # Internal radius
11  GEO["offset"]           = 0.15                      # Percentage value, until radial position gets
12                                                      #   higher, the infill is set constant.
13  GEO["originalPort"]     = GEO["port"]               # If extensions are added, the previous
14  GEO["originalRadius"]   = GEO["radius"]             # three parameters will be changed and need to be saved
15
16
17  # --- INPUTS - 3D printing parameters ---sour
18  INPUT["infill"]                 = 0.1               # Internal volumetric infill
19  INPUT["externalInfill"]         = 0.6               # External volumetric infill
20
21  INPUT["thickness"]              =   {
22                                          "gyroid":   0.5,    # Main gyroidal isosurface
23                                          "top":      0.4,    # Base surface
24                                          "internal": 0.8,    # Internal wall
25                                          "external": 0.8,    # External wall
26                                      }
27
28  # --- FLAG ---
29  # Optional components. The external surface is recommended  #
30  # to avoid the generation of suspended volumes              #
31  FLAG["extensions"] = {}
32  FLAG["extensions"]["internalWall"]          = 1         # Optional internal cylindric wall
33  FLAG["extensions"]["externalWall"]          = 1         # Optional external cylindric wall
34  FLAG["extensions"]["topWall"]               = 1         # Optional base surface at one of two ends
35
36
37
```

Figure 2: Main inputs to be configured in the main script.

To enhance efficiency and readability, all variables are organized into structured dictionaries (equivalent to MATLAB structures in the original implementation). This organization simplifies function calls and ensures a consistent interface across the entire codebase.

These data containers—identified by uppercase names—encapsulate the various groups of parameters:

- INPUT contains the primary geometric and physical parameters that define the grain;
- PARAM includes secondary configuration values related to algorithmic behavior and computational accuracy;
- FLAGS manages feature toggles, allowing specific functions or export options to be enabled or disabled at runtime.

In Fig. 2 the geometrical and printing inputs, with the flags are shown. Overall, the code was written with the dual goal of flexibility and usability. It allows researchers and engineers to quickly modify design parameters, test different geometric configurations, and export the resulting 3D structures directly to a slicer for printing. The internal modularity, inherited from the original MATLAB version but enhanced in Python, makes the system easy to maintain, extend, and integrate with other design or simulation tools.

2.1 The structure

2.2 Main script

2.3 Configurations

2.4 Mesh calibration tool

2.4.1 The algorithm

2.4.2 Montecarlo iterator

The function infillIteration() is a Monte Carlo–based evaluation routine designed to estimate the effective infill density generated by a given gyroid configuration. Rather than performing a full optimization itself, this function serves as a measurement tool that, for a specific combination of geometric parameters, computes the actual infill percentage obtained within a radial section of the grain. This value can then be used externally by an iterative calibration or optimization process to adjust the gyroid wavelength and match the desired infill distribution.

At each call, the function receives the current wavelength value WL, the previous one prevWL, and several configuration dictionaries (INPUT, SEC, GEO,

and PARAM) containing the global geometric and algorithmic parameters. From these inputs, it defines a temporary local section sec that represents the radial portion of the grain being analyzed. The inner and outer radii of this section are determined based on the port radius and on the wavelength value, scaled by a Monte Carlo divider parameter (PARAM["monteCarlo"]["divider"]). The function also computes auxiliary geometric quantities such as the section's center, length, and the corresponding target infill. This target is derived through a linear interpolation between the internal and external infill values, INPUT["infill"] and INPUT["externalInfill"], ensuring that the intended density smoothly varies along the radius. The local wavelength rate, stored in sec["rate"], quantifies how the field wavelength changes from one section to the next.

When the goal argument is set to "mesh", the function calls the routine meshConstruction() to generate the scalar gyroid field f for the current section, using the current wavelength and geometric parameters. The actual infill percentage is then estimated through a Monte Carlo approach: the values of f are sampled across the grid, and the proportion of points belonging to the solid phase is computed using the smoothed indicator function

2.5 Mesh construction

2.5.1 The concept of isosurface

At the core of the algorithm lies the definition of a scalar field $f(x)$, which implicitly describes the geometry of the gyroid lattice. The function f is constructed as a trigonometric combination of periodic terms along the three spatial directions, producing a triply periodic minimal surface (TPMS). The field is defined as:

$$\begin{aligned} f = &+ \cos\left(\frac{2\pi\, y}{\lambda_W}\right) \sin\left(\frac{2\pi\, x}{\lambda_W}\right) \\ &+ \cos\left(\frac{2\pi\, z}{\lambda_z}\right) \sin\left(\frac{2\pi\, y}{\lambda_W}\right) \\ &+ \cos\left(\frac{2\pi\, x}{\lambda_W}\right) \sin\left(\frac{2\pi\, z}{\lambda_z}\right) \end{aligned} \tag{1}$$

where λ_W denotes the characteristic wavelength along the planar directions (x,y), and λ_z (defined in the code as `GEO["waveLength"]`) represents the axial wavelength along z. By controlling these two parameters, it is possible to tune the periodicity and aspect ratio of the gyroid unit cells: λ_W determines how rapidly the field oscillates in the transverse plane, whereas the latter defines the repetition rate along the grain axis.

The function $f(x)$ does not directly define a solid object; instead, it encodes a continuous scalar field whose isosurfaces correspond to different levels of the function. An isosurface is defined as the locus of points satisfying:

$$f(\mathbf{x}) = threshold, \tag{2}$$

where `threshold` is a chosen constant value. By selecting a specific isovalue, the algorithm extracts the corresponding geometric surface embedded in the three-dimensional grid. The classic gyroid shape corresponds to the isosurface at $f(x) = 0$, which separates regions of positive and negative field values. In this work, this concept is extended by considering two symmetric isosurfaces enclosing a thin volumetric region that represents the printable solid material of the lattice.

To numerically construct the gyroid, the code first generates a structured 3D grid over cylindrical coordinates, mapping each point (r, θ, z) to its Cartesian counterpart through:

$$x = r\cos\theta, \qquad y = r\sin\theta, \qquad z = z. \tag{3}$$

The scalar field f is then evaluated at each node of this grid. Once computed, the field serves as a volumetric dataset from which the desired isosurface can be extracted using a polygonization algorithm such as Marching Cubes or an equivalent implementation.

This approach offers significant flexibility: by simply modifying the parameters $lambda_W$, λ_z, and the isovalue (the chosen threshold), one can control the unit-cell size, overall density, and wall thickness of the generated structure. Furthermore, because the entire geometry is mathematically defined, the resolution and precision of the final mesh depend only on the discretization of the computational grid, rather than on external CAD modeling tools. This makes the process computationally efficient and perfectly suited for automated design pipelines where the gyroid geometry must adapt to specific physical or mechanical requirements.

2.5.2 Threshold calibration process

In order to control the physical thickness of the gyroid walls, it is necessary to establish a mathematical relationship between the offset of the level set function, defined by a threshold value, and the actual geometric thickness of the structure. Let $f(x)$ be the scalar field defining the implicit gyroid surface, with the reference surface given by $f(x) = 0$. Any nearby surface can be represented as an

isosurface of f at level $f(x) = \Delta f$. To relate this variation in function value to a physical displacement, we can expand f along the local normal direction using a first-order Taylor approximation. Considering a point x_0 lying on the reference surface $f(x_0) = 0$, and moving a small distance δ along the outward normal unit vector $\mathbf{n} = \nabla f / \|\nabla f\|$, the Taylor expansion gives:

$$f(\mathbf{x}_0 + \delta \mathbf{n}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\delta \mathbf{n}) = \|\nabla f(\mathbf{x}_0)\| \, \delta. \tag{4}$$

This shows that the change in the scalar field value Δ is approximately proportional to the spatial displacement δ, according to the relation:

$$\delta \approx \frac{\Delta f}{\|\nabla f\|}. \tag{5}$$

In the implemented algorithm, the gyroid wall is obtained as the region enclosed between two symmetric isosurfaces defined by $f = +threshold$ and $f = -threshold$. The total distance T between these two surfaces can therefore be approximated as:

$$T \approx (\delta_+ + \delta_-) = \frac{2 \ \text{threshold}}{\|\nabla f\|}. \tag{6}$$

This expression ensures that, when symmetric isosurfaces are used, the generated wall thickness corresponds approximately to the target value imposed by the user. The validity of this relation relies on several assumptions: the displacement δ must be small compared to the local curvature radius of the surface, ensuring that the first-order Taylor expansion remains valid; the gradient magnitude $\|\nabla f\|$ should remain approximately constant along the normal direction between the two isosurfaces; and the scalar field $f(x)$ must be sufficiently smooth and differentiable, without significant numerical noise. Under these conditions, the resulting wall thickness is accurate to first order, while higher-order curvature effects introduce corrections of order $O(k\delta^2)$, where k denotes the local mean curvature of the surface. These terms are generally negligible when the offset distance δ is small relative to the curvature scale, but can become relevant in regions of high curvature or coarse numerical resolution.

2.5.3 Switch flags

2.6 Python libraries

3 Results

3.1 Considered subcases

3.2 Mesh quality and slicer capabilities

3.3 Required computational time

3.4 Weigth of STL products

4 Conclusion

Qui inizia il contenuto del documento principale.