Les Cahiers du GERAD ISSN: 0711–2440

Cesogen: Cellular solid generator

P. A. Patience, C. Audet, B. Blais

G–2024–15

February 2024

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2024
– Bibliothèque et Archives Canada, 2024

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2024
– Library and Archives Canada, 2024

Cesogen: Cellular solid generator

Paul A. Patience [a]
Charles Audet [a]
Bruno Blais [b]

[a] *GERAD, & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4*

[b] *CHAOS & Department of Chemical Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4*

paul-a.patience@polymtl.ca
charles.audet@polymtl.ca
bruno.blais@polymtl.ca

February 2024
Les Cahiers du GERAD
G–2024–15

Abstract : Cellular solids are structures which have applications in mechanical engineering to make lightweight structures and heat exchangers, in biomedical engineering to make tissue scaffolds, and in chemical engineering to make catalysts. A subset of these, triply periodic minimal surface–like cellular solids, are seeing growing adoption with recent advances in additive manufacturing. Here we present a program, Cesogen, which interprets a novel domain-specific language (DSL) for specifying signed distance functions (SDFs) to generate cellular solid meshes, and which is designed to be paired with blackbox optimizers in order to spur more efficient research into cellular solids. It converts input meshes to SDFs before transforming and combining them with operations such as translation, scaling and intersection, which allows Cesogen to robustly generate hierarchical cellular solids. Finally, Cesogen contours the combined SDF via marching cubes to produce a resulting mesh which is suitable as input to a physics simulator.

Keywords : Cellular solids, triply periodic minimal surfaces, signed distance functions, additive manufacturing, blackbox optimization

Acknowledgements: The authors would like to thank Marc-Étienne Lamarche-Gagnon from NRC for constructive discussions. This project was funded by NSERC CRD grant 538319-2018 with NRC and by the discovery grants 239436–01 (Audet) and RGPIN-2020-04510 (Blais).

Data availability statement: The data that supports the findings of this study is available from the corresponding author upon reasonable request. However, it may be generated by running the example scripts provided in Cesogen's repository, specifically `examples/hierarchical.lisp` for the hierarchical experiment, `examples/roundtrip.lisp` for the roundtrip experiment and `examples/benchmark.lisp` for the benchmark.

Conflict of interest: The authors declare no conflict of interest.

Nomenclature

Abbreviations

API	application programming interface
CLI	command-line interface
DSL	domain-specific language
GUI	graphical user interface
SDF	signed distance function
TPMS	triply periodic minimal surface
TPSf	triply periodic surface
TPcS	triply periodic eccentric surface
TPnS	triply periodic endoskeleton
TPxS	triply periodic exoskeleton

Symbols

d	distance; $d\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$
f_Ω	SDF of Ω; $f_\Omega\colon \mathbb{R}^n \to \mathbb{R}$
$\tilde{f}_\Omega$	approximate SDF of Ω; $\tilde{f}_\Omega\colon \mathbb{R}^n \to \mathbb{R}$
$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geq 0}$	set of non-negative real numbers
x	point; $x \in \mathbb{R}^n$
x, y, z	coordinates of point; $x, y, z \in \mathbb{R}$
ξ	isovalue; $\xi \in \mathbb{R}$
π	circle constant
τ	full circle constant $(2\,\pi)$
Ω	feasible set or geometric object
$\partial\Omega$	boundary of Ω

1 Introduction

Cellular solids are porous structures, sometimes periodic, sometimes stochastic, composed of cells made up of solid struts, plates and surfaces [13]. They can be classified into honeycombs, foams, and lattice structures, the last of which consists of strut-based and triply periodic minimal surface (TPMS)–like cellular solids.

Strut-based cellular solids have applications in many fields, including for thermal insulation, packaging, and lightweight structures [13]. Advances in additive manufacturing [40] have prompted more research into TPMS-like cellular solids because they could be manufactured more easily. TPMS-like cellular solids are now used in many applications, including mechanical (for energy and impact absorption, lightweight structures), thermal (as heat exchangers), biological (for tissue engineering scaffolds), and chemical (as batteries, catalysts, water-absorbing films) [9]. Sandwich panels are an example of lightweighting, where the inside is composed of a cellular solid to reduce the weight of the part while maintaining its structural integrity. The advantage of using cellular solids for heat transfer is that they have a high surface area, much higher than a solid block of equivalent dimensions. In biological applications, e.g., bone implants, TPMS-like cellular solids are preferred to strut-based cellular solids because the former are more similar to structures found in the body, and cells attach to them more easily.

Many aspects of cellular solids warrant and need more investigation, including their use in heat and mass transfer, and graded, heterogeneous and multiscale cellular solids [9]. Mathematical optimization of cellular solids, and cellular solids in general, is one aspect for which there is a dearth in the literature.

This article presents the cellular solid generator Cesogen; it is a program that can take a definition of a cellular solid, e.g., its name and properties, or the signed distance function (SDF) defining it, and generate a mesh suitable for simulation and manufacturing.

Cellular solid generators are a critical element in any study of cellular solids. Their features include a combination of:

- a user interface, be it command-line or graphical;
- an extensive library of cellular solids, including graded, heterogeneous and multiscale cellular solids;
- the ability to fill arbitrary geometries with a cellular solid;
- configurable contouring algorithms; and
- performance (ideally the generation time should be negligible compared to the simulation time).

Cesogen is not the first of its kind; there exist many cellular solid generators, whether freely available and open source, freely available and closed source, or commercial and proprietary. The freely available generators, from earliest to latest publication date of the supporting article, are:

- ScaffoldStructures [8], the first such software to get published, written in Python;
- MSLattice [1], written in MATLAB and can generate graded cellular solids;
- MiniSurf [15, 16], written in MATLAB;
- Scaffolder [17], written in C++ and Python, with a CLI (command-line interface), and can fill arbitrary solids;
- TPMS Designer [18, 19], written in MATLAB and can translate, scale and rotate cellular solids;
- FLatt Pack [25], written in MATLAB, has an extensive library of cellular solids, can fill arbitrary solids and generate graded cellular solids;
- ASLI [30], written in C++, with a CLI, and can fill arbitrary solids and generate graded and heterogeneous cellular solids;
- MaSMaker [39], written in MATLAB and can fill arbitrary solids;
- Lattice_Karak [34], written in MATLAB and can generate graded, heterogeneous and hierarchical cellular solids; and
- MD-TPMS [22], written in MATLAB and can generate graded cellular solids.

Commercial cellular solid generators are usually a smaller part of a more general design tool, and include Iamsamang and Naiyanetr [17], Raju and Onkar [34]: Optistruct (Altair), Netfabb and Within (Autodesk), Sulis (Gen3D), 3-Matic (Materialise), nTop (nTopology) [26], Creo Parametric (PTC), Grasshopper (Rhino 3D), and Simpleware (Synopsys). Freely available cellular solid generators forming a smaller part of a more general design tool include the Add Mesh Extra Objects plugin for Blender, and K3DSurf [34].

The particularity of Cesogen is that it is specifically adapted to computer-guided optimization. The primary features an optimization-focused generator needs in order to be usable are the ability to generate cellular solids and a way to operate it from the optimizer, e.g., a CLI.

The contribution of this article is to introduce and present Cesogen, a tool for generating cellular solids that researchers can use in their studies of these materials.

This article begins by presenting the theory required to understand the operation of Cesogen — starting with a description of SDFs and cellular solids — and continues by presenting the user interface of Cesogen and some details of how it works. Finally, it presents some examples of Cesogen in use and ends with a conclusion.

2 Theory

Two theoretical concepts are required to explain the functioning of Cesogen: SDFs and cellular solids.

2.1 Signed distance functions

In a metric space $(\mathbb{R}^n, d)$, the SDF f corresponding to an object $\Omega \subseteq \mathbb{R}^n$ is

$$f(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{if } x \in \mathbb{R}^n \setminus \Omega \end{cases} \tag{1}$$

where $\partial\Omega$ is the boundary of Ω and

$$d(x, \partial\Omega) := \inf_{y \in \partial\Omega} d(x, y). \tag{2}$$

In other words, f is a function taking negative values within Ω, positive values without, and the value zero on its boundary.

The convention of negative for inside and positive for outside is not universal; it is adopted by some authors [23], but others use the opposite convention [5, 29].

SDFs of basic objects are generally derived mathematically [32]. The SDFs of geometric objects represented as meshes are brute forced by calculating the shortest distance from x to the polygons in the mesh, possibly sped up with spatial query structures such as a bounding volume hierarchy. The sign of the distance can be resolved in many ways, one of which is described by Bærentzen and Aanæs [3].

The mathematical transformations morphological dilation and erosion, and taking the shell, are particularly useful when combining SDFSs:

- Dilation by offset $\xi \in \mathbb{R}_{\geq 0}$:
$$f'(x) = f(x) - \xi \tag{3}$$
- Erosion by offset $\xi \in \mathbb{R}_{\geq 0}$:
$$f'(x) = f(x) + \xi \tag{4}$$
- Shell of thickness $2\xi \in \mathbb{R}_{\geq 0}$:
$$f'(x) = |f(x)| - \xi \tag{5}$$

The usual mathematical transformations translation, scaling and rotation are also applicable:

- Translation by offset $\beta \in \mathbb{R}^3$:
$$f'(x) = f(x - \beta) \tag{6}$$
- Scaling by factor $\alpha \in \{\mathbb{R} \setminus 0\}^4$:
$$f'(x) = \alpha_0 f(x') \tag{7}$$
where $\alpha_0 = \min\{|\alpha_1|, |\alpha_2|, |\alpha_3|\}$ and $x'_i = x_i/\alpha_i$ for $i \in \{1, 2, 3\}$
- Rotation by angle $\theta \in \mathbb{R}$ around axis $v \in \mathbb{R}^3$:
$$f'(x) = f\big((A + B)x\big) \tag{8}$$
where $A_{ij} = v_i v_j (1 - \cos\theta)$ and
$$B = \begin{pmatrix} \cos\theta & -v_3 \sin\theta & v_2 \sin\theta \\ v_3 \sin\theta & \cos\theta & -v_1 \sin\theta \\ -v_2 \sin\theta & v_1 \sin\theta & \cos\theta \end{pmatrix}$$

SDFs are practical for the relative simplicity with which they can be combined to form new shapes. The basic implementation of set-theoretic operations for SDFs is the following [29] though they are merely approximations, hence the tilde:

- Union:
$$\tilde{f}_{\Omega \cup \Lambda} = \min(f_\Omega, f_\Lambda) \tag{9}$$

- Intersection:
$$\tilde{f}_{\Omega\cap\Lambda} = \max(f_\Omega, f_\Lambda) \tag{10}$$

- Difference:
$$\tilde{f}_{\Omega\setminus\Lambda} = \tilde{f}_{\Omega\cap(\mathbb{R}^n\setminus\Lambda)} \tag{11}$$

Note that the implementations of union and intersection are swapped if the SDF sign convention is inverted. Unfortunately, these operations do not always result in a proper SDF; they may result in a signed distance bound, where the boundary is correct but the inner and outer values are not [33].

SDFs are ultimately useful herein when converted, or contoured, to polygon meshes so that simulations can be performed on them. Various methods exist for the contouring of SDFs, including marching cubes [6, 7, 14, 20, 21] (the predecessor of them all), dual marching cubes [27], and more advanced methods [24]. The methods differ in terms of number of polygons generated, performance, and kinds of shapes supported, though no research has been done to determine which methods would be ideal for cellular solids in particular.

2.2 Cellular solids

Gibson and Ashby [13] describe cellular solids as “an interconnected network of solid struts or plates which form the edges and faces of cells”, classifying them into honeycombs, which are 2D structures extruded into the third dimension, and foams, which are 3D stochastic structures. Recent literature has introduced a third class, made more usable thanks to additive manufacturing, called lattice structures, which could also be stochastic according to the design mechanism [4, 28, 38].

The fundamental parameters of a cellular solid are the cell size and relative density. The cell size, when discussing cubic cellular solids, is the length of the unit cell’s edges; there exist also orthorhombic cellular solids, where the edges of the unit cell have different lengths. The relative density, or volume fraction or solid fraction, of a cellular solid is ρ^*/ρ_s, where ρ^* is the cellular solid’s density and ρ_s the density of the solid composing the cellular solid [13].

Lattice structures are further divided into two kinds: strut-based and TPMS-like. Strut-based cellular solids have been studied for longer; TPMS-like cellular solids have grown in use thanks to additive manufacturing. This article studies only the latter.

The name lattice can lead to an unfortunate confusion when considering that many strut-based lattice structures are named after Bravais lattices from the field of crystallography [43], e.g., body-centered cubic. However, lattices are not limited to crystallography, and in fact one of the primary definitions of the word “lattice” is “an open framework made of strips of metal, wood, or similar material overlapped or overlaid in a regular, usually crisscrossed pattern” [2]. The terms from crystallography and mathematics are likely inspired from this definition.

TPMS-like cellular solids are based on the equations of implicit surfaces, and thus are also known as surface- or equation-based. They include TPMSs and other triply periodic (non-minimal) surfaces. TPMS-like cellular solids can be approximated by the periodic nodal surfaces of the Fourier series representing those cellular solids [41, 42].

TPMSs have no convenient exact representation for use in engineering applications, and so they are normally approximated by the first few terms of their Fourier series [12]. Sample equations are presented in Section 4.1.

Different isosurfaces of a fundamental TPMS can be obtained by varying the isovalue $\xi \in \mathbb{R}$ and eccentricity $\epsilon \in \mathbb{R}$ in the following equations [11]:

- Triply periodic endoskeleton (TPnS):
$$f_\Omega(x) \leq \xi \tag{12}$$

- Triply periodic exoskeleton (TPxS):

$$\xi \leq f_\Omega(x) \tag{13}$$

- Triply periodic surface (TPSf):

$$-|\xi| \leq f_\Omega(x) \leq |\xi| \tag{14}$$

- Triply periodic eccentric surface (TPcS):

$$-|\xi| \leq f_\Omega(x) - \epsilon \leq |\xi| \tag{15}$$

The isovalue and eccentricity control the relative density of the cellular solid, and also the shape of the walls in the case of TPSfs and TPcSs.

Certain transformations can be applied to cellular solids and TPMSs in particular. Arbitrary meshes can be filled with cellular solids by computing the SDF of the mesh and intersecting it with the cellular solid's SDF. Graded cellular solids are obtained by replacing constant transformations, e.g., translation, scaling, rotation, by functions [1, 9]. Heterogeneous cellular solids consist of fusions of fundamental cellular solids, and can be obtained via a sigmoid function [1]. Multiscale, or hierarchical, cellular solids are cellular solids within cellular solids [9], and can be obtained by intersecting cellular solids with different unit cell sizes.

3 Cellular solid generator

The cellular solid generator introduced in this article is called Cesogen, which is a contraction of "cellular solid generator". First we provide a summary of Cesogen and its features, then how it may be used.

3.1 Overview

Cesogen is written in Common Lisp, a programming language featuring a good balance of interactivity and performance. It is developed first and foremost as a software library, in order that it may be used interactively from a read-eval-print loop, thus allowing more systematic exploration of cellular solids than a CLI or GUI (graphical user interface) can provide. Another advantage of being a library is that the program logic is decoupled from the CLI, which allows third-party applications, including, but not limited to GUIs, to benefit from a more expressive interface than is possible via a CLI. However, in order accommodate users of all kinds, including optimizers, Cesogen is also available as a CLI.

In spite of its name, Cesogen functions as a full-fledged SDF processor. Its ability to generate cellular solids is a byproduct of cellular solids being representable as SDFs. The name remains, though, because Cesogen is primarily geared towards generating cellular solids. And it is catchy.

Cesogen's mode of operation can be seen as consisting of three phases (Figure 1). In the first phase, it converts any meshes provided to their corresponding SDFs. The second phase consists of combining the SDFs according to the operation requested. This results in a single SDF, which is then contoured to produce the resulting mesh. In reality, Cesogen proceeds along these phases eagerly, i.e., it converts meshes and combines them as soon as it can, so the phases are actually interleaved.

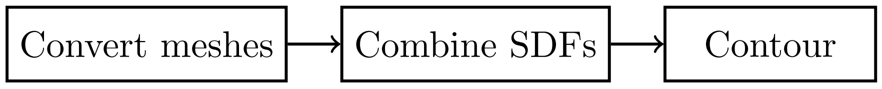

Figure 1: Phases of Cesogen.

Cesogen accepts as input arbitrary solids represented as meshes. It supports various formats, including OBJ, OFF, PLY and STL. It converts these meshes to SDFs by computing an intermediate sphere-based bounding volume hierarchy for speeding up proximity queries based on a port of TriangleMeshDistance [10] to Common Lisp. The signs of the computed distances are resolved via the algorithm described by Bærentzen and Aanæs [3].

Cesogen also boasts an extensive library of cellular solids, currently including all the TPMS-likes described by Fisher et al. [11] (Table 1). Users may define their own cellular solids by providing equations directly to Cesogen instead of meshes, or saving the equations in a file which is then provided, and more TPMS-like and other equations will be added as they are discovered.

In the SDF combination phase, Cesogen supports various operations on the SDFs, including translation, scaling, rotation, dilation, erosion, shell and the set-theoretic operations complement, union, intersection and difference (Section 2). It cannot yet create graded or heterogeneous cellular solids, but it can create multiscale, or hierarchical, cellular solids by intersecting cellular solids of differing unit cell sizes.

The contouring phase consists of applying marching cubes to the combined SDF to produce the resulting mesh. The choice of contouring algorithm has an impact on the number of mesh elements composing it, which in turn has an impact on the running time of the physics solver. Cesogen currently uses VTK's [36] implementation of marching cubes, but further algorithms will be added which offer different tradeoffs.

Performance is important only insofar as the cellular solid generation step takes a negligible amount of time compared to the rest of the current optimization step. Cesogen has been thus developed with a focus on performance.

Finally, Cesogen is freely available, open source software licensed under the MIT (more specifically, the Expat) license [37], a permissive license which allows commercial use with few restrictions. Its homepage is `https://git.sr.ht/~paulapatience/cesogen`.

Cesogen is multiplatform; it runs on various flavors of Linux and also FreeBSD, OpenBSD, NetBSD, macOS and Windows. Precompiled binaries are available for each release on the homepage, but it may also be built if one has the SBCL [35] compiler. Specific installation instructions are described in the manual.

3.2 Usage

This section briefly describes the usage of Cesogen. The Cesogen manual goes into more detail.

Cesogen's CLI is aimed at being comprehensive enough to expose all the functionality that clients, usually users and optimizers, may require. Optimizers are expected to be provided with a script that will launch Cesogen with the appropriate command-line arguments. Therefore, the most important part of Cesogen's API, for the average client, is its CLI.

Cesogen's CLI is a mostly stack-based, domain-specific language (DSL) which dispenses with the need to have nested parentheses on the command-line. Its command-line arguments can be one of four kinds: command-line options, SDF names, unary SDF operations, and binary SDF operations. The command-line options are discussed at the end of this section.

SDF names must be either the name of an SDF as recognized by Cesogen, the name of a mesh file with an extension recognized by Cesogen, or an infix equation containing the variables `x`, `y`, `z` and some basic mathematical constants and operations, e.g., `pi`, trigonometric functions. These inputs are disambiguated in the listed order, i.e., SDF names take precedence over filenames, which take precedence over equations.

The non-TPMS SDFs supported by Cesogen are currently:

- `sphere`, the unit sphere centered at the origin;

- `cylinder`, the unit cylinder centered at the origin, oriented along the z axis; and
- `box`, the unit box, i.e., with a half-width of 1, centered at the origin.

The TPMS-like SDFs supported by Cesogen are many, and documented in the manual. Cesogen extracts the SDF of any mesh provided on the command-line, and automatically calculates its bounds. Equations look like '`x+y+z-1`'; point coordinates are represented by x, y, z because it is more convenient than x_1, x_2, x_3, and there is no possible ambiguity. The bounds of arbitrary equations are assumed to be infinity.

To generate a sphere, run:

```
cesogen sphere
```

When an SDF is specified on the command-line, it is pushed onto a stack of SDFs. This stack is operated upon by the unary SDF and binary SDF operations. Unary SDF operations pop the SDF stack once, modify the SDF, and push it back onto the stack. Binary SDF operations pop the stack twice, combine the SDFs accordingly, and push the result back onto the stack. Unary SDF operations may also take non-SDF arguments, which are specified separately as pairs of names and values in the following arguments on the command-line. The terms unary and binary in the context of SDF operations refer to the number of SDFs taken by the operations.

The unary SDF operations and the arguments they take are:

- `complement`, none
- `dilate`, offset $\xi \in \mathbb{R}_{\geq 0}$
- `erode`, offset $\xi \in \mathbb{R}_{\geq 0}$
- `shell`, thickness $2\xi \in \mathbb{R}_{\geq 0}$
- `translate`, offset $\beta \in \mathbb{R}^3$
- `scale`, factor $\alpha \in \mathbb{R}^3$
- `rotate`, angle $\theta \in \mathbb{R}$, axis $v \in \mathbb{R}^3$

The components of arguments in $\mathbb{R}^3$ are listed in one command-line argument and separated by commas. Numeric arguments may also be equations, though without any point coordinates. To generate a sphere with radius 2, run:

```
cesogen sphere scale 2,2,2
```

Finally, the binary SDF operations are `union`, `intersection` and `difference`. Cesogen's command-line always implicitly ends in an n- ary intersection. To generate a unit cylinder with a height of 2:

```
cesogen cylinder box
```

A more complex SDF combination is (Figure 2):

```
cesogen                           \
mesh1.ply gyroid    intersection \
mesh2.ply p-surface intersection \
union
```

Cesogen detects the bounding box of the resulting mesh automatically. This bounding box must be finite for Cesogen to be able to produce a result, so when meshing unbounded objects, e.g., a cylinder or TPMSs, at least one additional SDF must be provided, e.g., `sphere`, `cylinder`, `box` or some mesh.

The usual command-line options all start with the hyphen ('`-`') and are described in the manual and also by the `--help` option. Cesogen also supports the `@`*file* option, which it replaces with the contents of the specified file, which should consist of whitespace-separated command-line arguments.

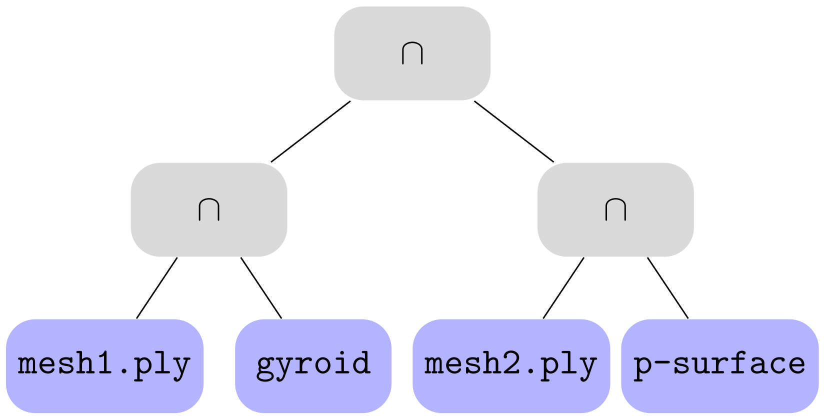

Figure 2: Tree representation of the operations Cesogen performs when invoked as '`cesogen mesh1.ply gyroid intersection mesh2.ply p-surface intersection union`'. It evaluates the SDFs in a depth-first, post-order manner. In fact, the command-line arguments are a flattened version of the tree with the nodes visited in depth-first post-order.

This option acts as an ad hoc configuration file mechanism; users can specify their SDFs, or even operations on SDFs, in files which they later include via the `@`*file* option. It is particularly apt for specifying custom equations.

The `-s` option specifies the spacing of the grid used during contouring. It may consist of one or three components; in the former case, the spacing is the same along all dimensions. The `-o` option specifies the output file, overriding the default of `out.ply`; the output mesh format is detected from the extension.

The TPMS-like–specific parameters isovalue $\xi \in \mathbb{R}$ and eccentricity $\epsilon \in \mathbb{R}$ may be reproduced via dilation, erosion, and shelling. For a TPnS S with equation $f_S(x) \leq \xi$, run:

`cesogen S dilate` ξ `box`

For a TPxS S with equation $-f_S(x) \leq -\xi$, run:

`cesogen S complement erode` ξ `box`

For a TPSf S with equation $-|\xi| \leq f_S(x) - \epsilon \leq |\xi|$, run:

`cesogen S dilate` ϵ `shell '2*abs(`ξ`)' box`

4 Results and discussion

This section contains a series of examples of the kinds of cellular solids that Cesogen can generate, and also an evaluation of the performance and limitations of Cesogen.

4.1 Examples of TPMS-like cellular solids

Cesogen supports many TPMS-like cellular solids; they are listed in Table 1, where

$$S_{f(x,y,z)} = \sin(\tau\, f(x,y,z)) \tag{16}$$
$$C_{f(x,y,z)} = \cos(\tau\, f(x,y,z)) \tag{17}$$

and $\tau = 2\,\pi$. Point coordinates are represented by x, y, z rather than x_i to conform to Cesogen's equation syntax.

A selection of these TPMS-like cellular solids is presented in Figures 3 and 4.

Table 1: Equations of TPMS-like cellular solids described by [11].

Name	SDF equation
Gyroid	$C_xS_y + C_yS_z + C_zS_x$
D	$S_xS_yS_z + C_xS_yC_z + C_yS_zC_x + C_zS_xC_y$
P	$C_x + C_y + C_z$
IWP	$2(C_xC_y + C_yC_z + C_zC_x) - (C_{2x} + C_{2y} + C_{2z})$
Neovius	$4C_xC_yC_z + 3(C_x + C_y + C_z)$
C(Y)	$-S_xS_yS_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x - C_xC_yC_z + C_xS_{2y} + C_yS_{2z} + C_zS_{2x}$
Lidinoid	$S_{2x}C_yS_z + S_{2y}C_zS_x + S_{2z}C_xS_y - (C_{2x}C_{2y} + C_{2y}C_{2z} + C_{2z}C_{2x}) + 0.3$
OCTO	$0.6(C_xC_y + C_yC_z + C_zC_x) - 0.4(C_x + C_y + C_z) + 0.25$
FRD	$8C_xC_yC_z + C_{2x}C_{2y}C_{2z} - (C_{2x}C_{2y} + C_{2y}C_{2z} + C_{2z}C_{2x})$
S	$C_{2x}S_yC_z + C_{2y}S_zC_x + C_{2z}S_xC_y$
P+C(P)	$0.3C_xC_yC_z + 0.1C_{2x}C_{2y}C_{2z} + 0.2(C_x + C_y + C_z) + 0.1(C_{2x} + C_{2y} + C_{2z}) + 0.05(C_{3x} + C_{3y} + C_{3z}) + 0.1(C_xC_y + C_yC_z + C_zC_x)$
Split P	$1.1(S_{2x}C_yS_z + S_{2y}C_zS_x + S_{2z}C_xS_y) - 0.2(C_{2x}C_{2y} + C_{2y}C_{2z} + C_{2z}C_{2x}) - 0.4(C_{2x} + C_{2y} + C_{2z})$
F	$C_xC_yC_z$
C(D)	$C_{3x+y}C_z - S_{3x-y}S_z + C_{x+3y}C_z + S_{x-3y}S_z + C_{x-y}C_{3z} - S_{x+y}S_{3z}$
G′	$S_{2x}C_yS_z + S_{2y}C_zS_x + S_{2z}C_xS_y + 0.32$
G′$_2$	$5(S_{2x}C_yS_z + S_{2y}C_zS_x + S_{2z}C_xS_y) + C_{2x}C_{2y} + C_{2y}C_{2z} + C_{2z}C_{2x}$
D′	$0.5(C_xC_yC_z + S_xC_yS_z + S_yC_zS_x + S_zC_xS_y) - 0.5(S_{2x}S_{2y} + S_{2y}S_{2z} + S_{2z}S_{2x}) - 0.2$
K	$0.3(C_x + C_y + C_z + C_xC_y + C_yC_z + C_zC_x) - 0.4(C_{2x} + C_{2y} + C_{2z}) + 0.2$
C(S)	$C_{2x} + C_{2y} + C_{2z} + 2(S_{2x}C_yS_{3z} + S_{2y}C_zS_{3x} + S_{2z}C_xS_{3y}) + 2(S_{2x}C_{3y}S_z + S_{2y}C_{3z}S_x + S_{2z}C_{3x}S_y)$
Y	$S_xS_yS_z + C_xS_{2y} + C_yS_{2z} + C_zS_{2x} + C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
$\pm Y$	$2C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
$C(\pm Y)$	$-2C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
$C(I_2 - Y^{**})$	$2(S_{2x}C_yS_z + S_{2y}C_zS_x + S_{2z}C_xS_y) + C_{2x}C_{2y} + C_{2y}C_{2z} + C_{2z}C_{2x}$
W	$C_{2x}C_y + C_{2y}C_z + C_{2z}C_x - (C_xC_{2y} + C_yC_{2z} + C_zC_{2x})$
Q*	$(C_x - 2C_y)C_z - \sqrt{3}S_z(C_{x-y} - C_x) + C_{x-y}C_z$
C(G)	$3(S_xC_y + S_yC_z + S_zC_x) + 2(S_{3x}C_y + S_{3y}C_z + S_{3z}C_x) - 2(S_xC_{3y} + S_yC_{3z} + S_zC_{3x})$
Slotted P	$-2(C_xC_y + C_yC_z + C_zC_x) - 2(C_{2x} + C_{2y} + C_{2z}) + C_{2x}C_y + C_{2y}C_z + C_{2z}C_x - (C_xC_{2y} + C_yC_{2z} + C_zC_{2x})$

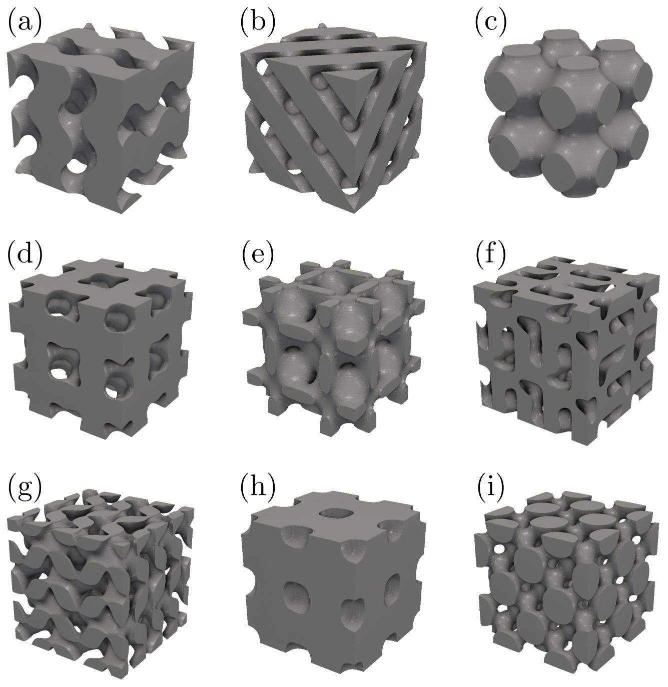

Figure 3: Showcase of various endoskeletal TPMS-likes generated by the command '`cesogen -s 2/m box S dilate` ξ' with by default $m = 60$ and $\xi = 0.35$. (a) Gyroid, (b) D, (c) P, (d) IWP ($m = 85$), (e) Neovius, (f) C(Y) ($m = 85$), (g) Lidinoid ($m = 85$), (h) OCTO ($\xi = 0.1$), (i) FRD ($m = 95$, $\xi = 0.55$).

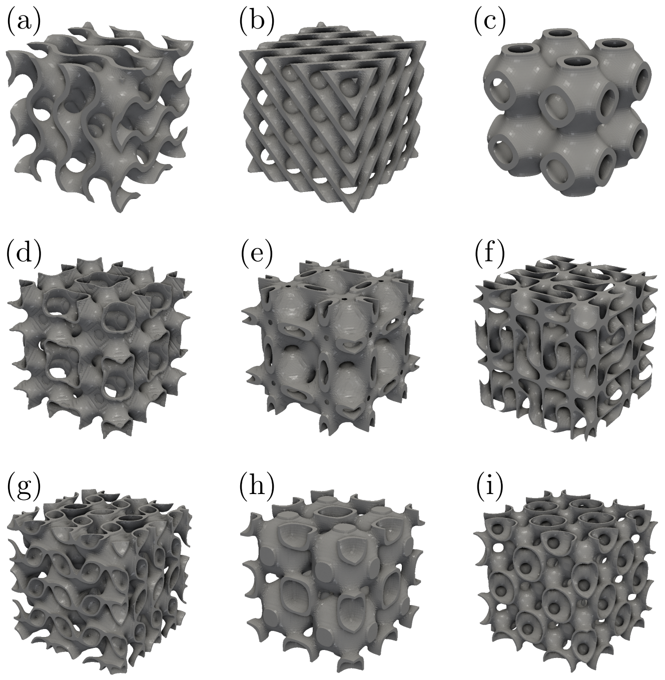

Figure 4: Showcase of various shell TPMS-likes generated by the command '`cesogen -s 2/m box S shell '2*`ξ`'`' with by default $m = 60$ and $\xi = 0.35$. (a) Gyroid, (b) D, (c) P, (d) IWP ($m = 85$), (e) Neovius, (f) C(Y) ($m = 85$), (g) Lidinoid ($m = 85$), (h) OCTO ($\xi = 0.1$), (i) FRD ($m = 95$, $\xi = 0.55$).

4.2 Generating hierarchical cellular solids

One aspect of Cesogen which distinguishes it from the other freely available cellular solid generators is its ability to robustly generate hierarchical cellular solids (Figure 5).

Cesogen generates hierarchical meshes by combining the constituent SDFs all at once at runtime, and computing one resulting SDF; hereafter this method is called "direct". Cellular solid generators which are unable to combine multiple SDFs are limited to computing one intersection, writing it to disk, and repeating the process for each additional SDF; hereafter this method is called "sequential". The sequential method incurs a performance cost — the writing and reading of each intermediate SDF — but also sets an upper bound on the number of combinations possible. For instance, Cesogen is currently unable to generate the hierarchical cellular solid in Figure 5g via the sequential method.

Indeed, the extraction of SDFs from triangle meshes containing narrow triangles, via a proximity algorithm, is fraught with floating point errors, because computing the distance from a point to a skewed triangle is numerically unstable. Whereas the direct SDF computation needs to extract a single SDF, the sequential computation extracts each of the intermediate SDFs. Furthermore, when the meshes result from a contouring algorithm such as marching cubes without a subsequent smoothing algorithm, narrow triangles abound.

The rest of this section demonstrates the issue with two examples. The first example is an expansion of the hierarchical cellular solid example above. The second is a series of roundtrip serialization–deserialization steps, where the serialization consists of writing an SDF to disk and the deserialization of extracting it via a proximity algorithm. In the ideal case, roundtrip conversions should be repeatable ad infinitum, but this is not the case in practice.

To aid in the analysis of the results, we measure four kinds of errors: the L^2 error of the distance from the mesh points and cell centers to the reference SDF, and the maximum distance from the points and cell centers to the reference SDF. These are all relative to the length of the diagonal of the grid.

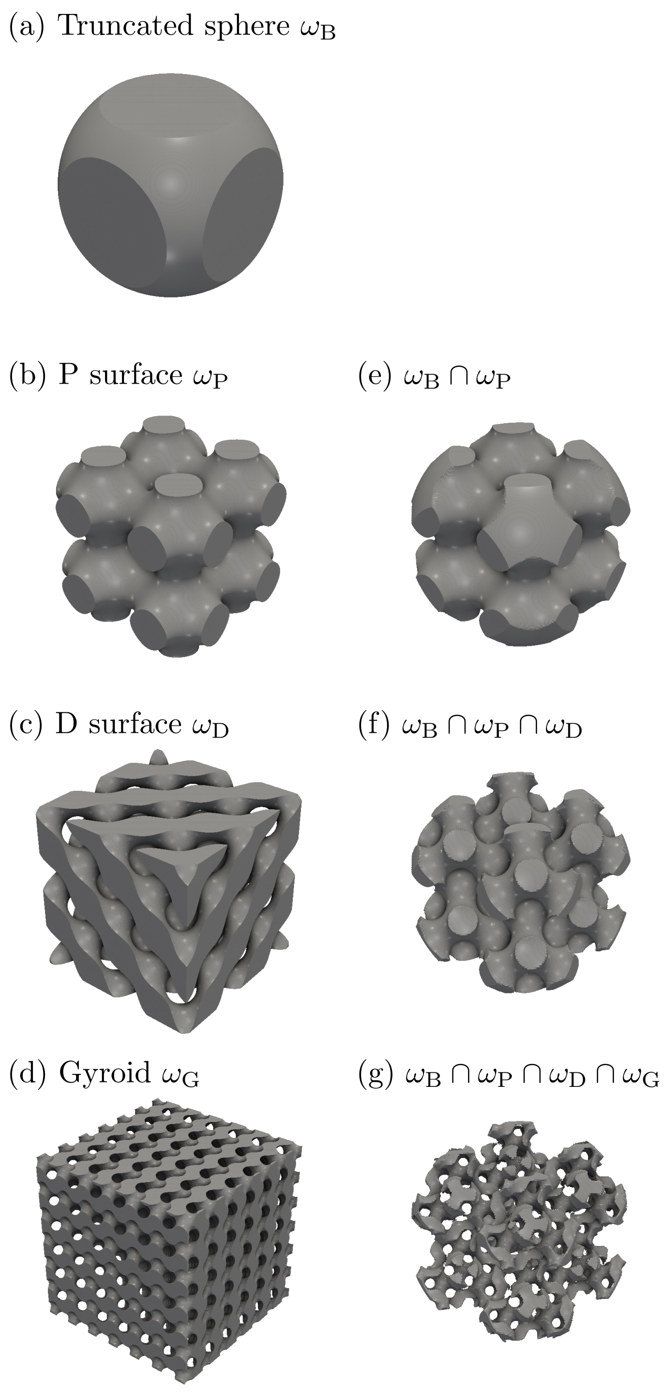
(a) Truncated sphere ω_{B}
(b) P surface ω_{P}
(e) $\omega_{\mathrm{B}} \cap \omega_{\mathrm{P}}$
(c) D surface ω_{D}
(f) $\omega_{\mathrm{B}} \cap \omega_{\mathrm{P}} \cap \omega_{\mathrm{D}}$
(d) Gyroid ω_{G}
(g) $\omega_{\mathrm{B}} \cap \omega_{\mathrm{P}} \cap \omega_{\mathrm{D}} \cap \omega_{\mathrm{G}}$

Figure 5: Hierarchical cellular solid consisting of the intersection of (a) a truncated sphere, (b) a P surface, (c) a finer D surface, and (d) an even finer gyroid. The figures at the left show the original models and those at the right the progressively intersected cellular solid. The command which generates the final result is ‘`cesogen -s 4/130 box scale 4,4,4 sphere scale 5,5,5 p-surface scale 4,4,4 d-surface scale 3.75,3.75,3.75 gyroid scale 1.25,1.25,1.25`’.

The hierarchical experiment’s errors are displayed in Figures 6 and 7. The zeroth and first steps are the same in the direct and sequential cases since they consist of the same operation. In the second step, the error for the sequential meshes is greater than that of the direct meshes in all cases. The third step is missing the sequential results because the generation fails for trying to divide by zero.

We can also see that there is more error along the edges of the sequential version of $\Omega_B \cap \Omega_P \cap \Omega_D$ (Figure 8).

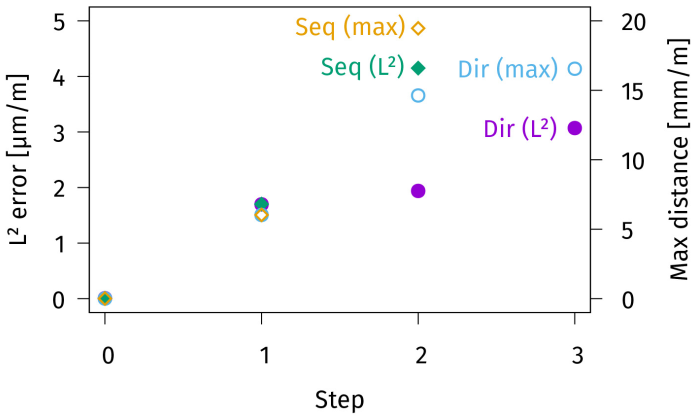

Figure 6: Direct and sequential hierarchical L^2 errors and max distances for the points. Direct errors are represented by circles, sequential errors by diamonds. L^2 errors are represented by filled shapes, max distances by hollow shapes. The labels are located beside actual points; there is no separate legend.

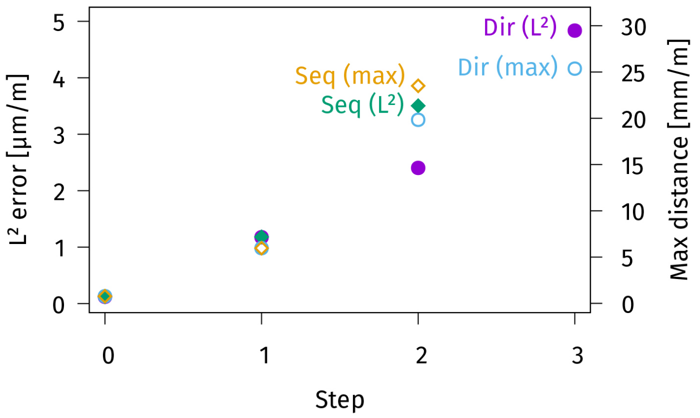

Figure 7: Direct and sequential hierarchical L^2 errors and max distances for the cells. Direct errors are represented by circles, sequential errors by diamonds. L^2 errors are represented by filled shapes, max distances by hollow shapes. The labels are located beside actual points; there is no separate legend.

The roundtrip experiment's errors are displayed in Figure 9. The data stops at the 15th roundtrip because the 16th results in a division by zero. The L^2 errors increase monotonously starting from the second and first roundtrips for points and cells, respectively; the maximum distances decrease and increase and decrease again throughout the roundtrips. However, the quality of the final mesh is severely degraded compared to the first (Figure 10).

We conducted the hierarchical experiment with ASLI and obtained similar results, i.e., ASLI crashes after a few combinations.

The results become even worse when starting with a more complex initial mesh, e.g., the Stanford bunny rather than a sphere. In some cases, artifacts begin to appear, i.e., triangles appearing outside of the objects.

The problems inherent to the sequential method can be mitigated by using more robust algorithms for computing the point to triangle distance, but those incur a performance cost and are thus warranted only when the initial mesh contains problematic features; for generating hierarchical cellular solids, the direct method should be preferred.

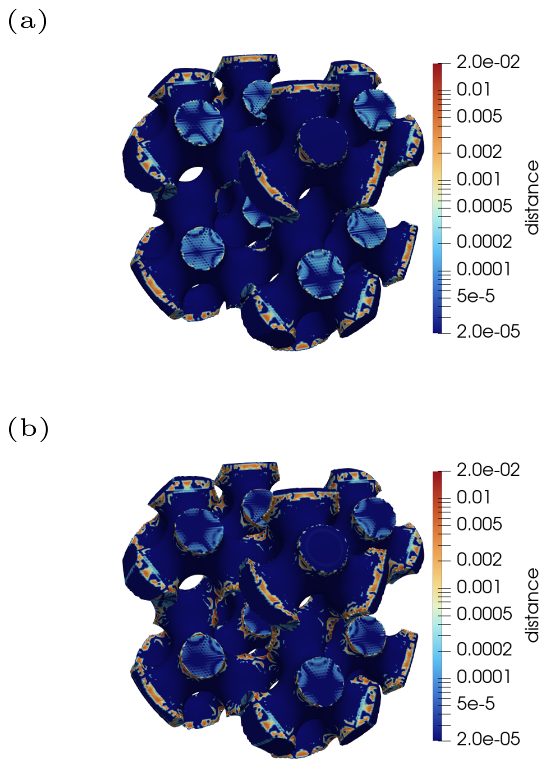

Figure 8: Colored meshes corresponding to $\Omega_\mathbf{B} \cap \Omega_\mathbf{P} \cap \Omega_\mathbf{D}$; (a) direct, (b) sequential. The log-scale coloring corresponds to the distance from the points to the reference SDF.

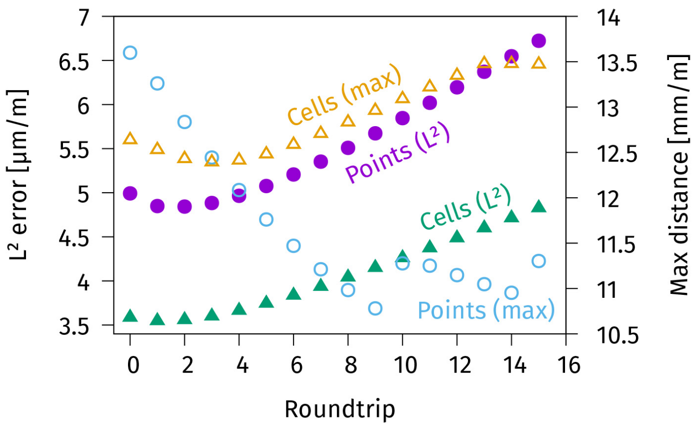

Figure 9: Roundtrip L^2 errors and max distances. Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.

4.3 Performance and limitations

This section examines the performance of Cesogen — how fast can it generate cellular solids, and how big can they be — and also some of its limitations.

We ran Cesogen via hyperfine [31] and generated a 2 unit cell by 2 unit cell by 2 unit cell gyroid with progressively finer contouring grids, i.e., grids of widths 2×2×2 and dimensions iii with $i \in \{26, 51, 101, 201, 301, \ldots, 801\}$ on a machine with the following specifications:

- OS: Chimera Linux x86_64
- Host: 20UH000CUS ThinkPad T14s Gen 1
- Kernel: 6.6.8-0-generic
- CPU: AMD Ryzen 7 PRO 4750U with Radeon Graphics (16) @ 1.700GHz
- Memory: 15220MiB

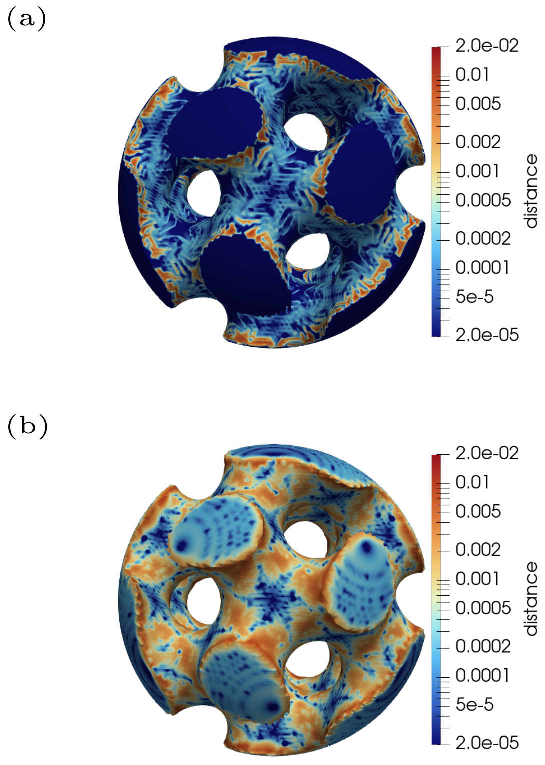

Figure 10: Meshes (a) before the first and (b) after the last successful roundtrips. The log-scale coloring corresponds to the distance from the points to the reference SDF. The command which generates the initial result is '`cesogen -s 4/100 box scale 4,4,4 sphere scale 4,4,4 gyroid scale 4,4,4`'.

The mesh generated from an 801801801 grid contained 16 million cells and took 7.4 min (Figure 11). Larger meshes than that required more than 16 GiB of memory.

Further limitations of Cesogen include the following:

- The cellular solid generation is not parallelized. However, this can be mitigated in some contexts by launching several instances of Cesogen as long as enough memory is available.
- Generating hierarchical cellular solids requires a fine contouring grid in order to capture all the fine features, which increases computation time.
- Cesogen requires an explicit contouring grid be given, unlike ASLI. This means that users of Cesogen must estimate the dimensions of the grid required to reach mesh convergence.
- The algorithm for computing mesh SDFs is not robust. As demonstrated in the previous section, it fails for some meshes.

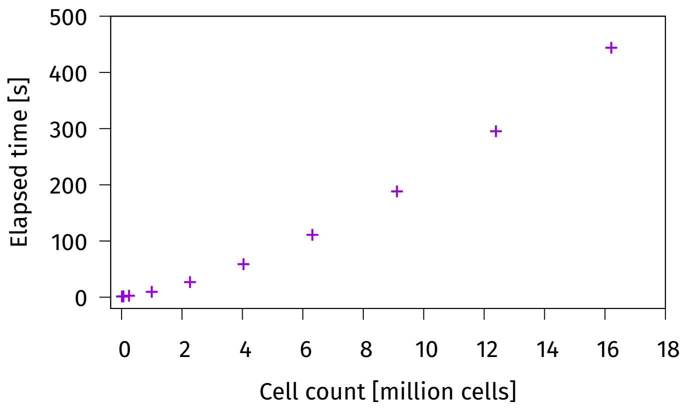

Figure 11: Mean time taken for Cesogen to generate a 2 unit cell by 2 unit cell by 2 unit cell gyroid with progressively finer contouring grids, i.e., grids of widths 2×2×2 and dimensions iii with $i \in \{ 26, 51, 101, 201, 301, \ldots, 801 \}$. Grid sizes beyond 801801801 required more memory than what was available on the test machine.

5 Conclusion

Cesogen is a general SDF processor which specializes in generating cellular solids in a manner suitable for computer-guided optimization. It has an extensive library of TPMS-like cellular solids and can generate hierarchical cellular solids. Its CLI is flexible enough to cover a wide range of use-cases and is targeted at any user wanting to study and optimize cellular solids.

Cesogen has some limitations, which include requiring an explicit contouring grid be provided, its SDF-computation algorithm being sensitive to narrow triangles in the input, and generating hierarchical cellular solids requiring a fine grid, which non-negligibly increases computation time.

Cesogen is under active development and its roadmap includes the following features:

- more cellular solids, in particular strut-based cellular solids;
- more contouring algorithms [7, 14], possibly parallel ones;
- a graphical user interface for real-time interactive exploration of cellular solids;
- heuristically saturating maximum distances when computing the SDFs of meshes containing narrow triangles;
- possibly built-in mesh adaptation, to make reading generated meshes more robust; and
- possibly more robust SDF computation algorithms.

One important future feature is providing a universal interface for cellular solid generation. The most compelling feature of Cesogen is its DSL for describing SDFs. We believe it can represent any cellular solid that the other freely available cellular solid generators can produce. The goal is to make Cesogen a universal cellular solid generator which can hook into any generator which has a CLI or is exposed as a library, e.g., ASLI, allowing Cesogen to benefit from any advancements to other cellular solids generators by leveraging them itself. Cesogen would then present a universal interface for cellular solid generation in order to facilitate the application of mathematical optimization techniques to cellular solid design.

In addition to continuing the development of Cesogen, further research could be done on determining which contouring algorithms are best suited for cellular solids.

References

[1] Oraib Al-Ketan and Rashid K. Abu Al-Rub. MSLattice: A free software for generating uniform and graded lattices based on triply periodic minimal surfaces. Material Design & Processing Communications, 3(6), 2020. doi: 10.1002/mdp2.205.

[2] American Heritage Dictionaries. Lattice. In American Heritage Dictionaries, editors, The American Heritage Dictionary of the English Language. HarperCollins Publishers, New York City, NY, United States, 5th edition, October 2018. URL https://ahdictionary.com/word/search.html?q=lattice.

[3] J. Andreas Bærentzen and Henrik Aanæs. Generating signed distance fields from triangle meshes. IMM Technical Report IMM-TR-2002-21, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2002. URL http://www2.imm.dtu.dk/pubdb/pubs/1289-full.html.

[4] Dhruv Bhate. Four questions in cellular material design. Materials, 12(7):1060, 2019. doi: 10.3390/ma12071060.

[5] Tony Chan and Wei Zhu. Level set based shape prior segmentation. In CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), page 1164–1170. IEEE Computer Society, June 2005. doi: 10.1109/cvpr.2005.212.

[6] Lis Custodio, Tiago Etiene, Sinesio Pesco, and Claudio Silva. Practical considerations on Marching Cubes 33 topological correctness. Computers & Graphics, 37(7):840–850, 2013. doi: 10.1016/j.cag.2013.04.004. URL http://www.sci.utah.edu/~etiene/pdf/mc33.pdf.

[7] Lis Custodio, Sinesio Pesco, and Claudio Silva. An extended triangulation to the Marching Cubes 33 algorithm. Journal of the Brazilian Computer Society, 25(6), 2019. doi: 10.1186/s13173-019-0086-6.

[8] J. C. Dinis, T. F. Morais, P. H. J. Amorim, R. B. Ruben, H. A. Almeida, P. N. Inforçati, P. J. Bártolo, and J. V. L. Silva. Open source software for the automatic design of scaffold structures for tissue engineering applications. Procedia Technology, 16:1542–1547, 2014. doi: 10.1016/j.protcy.2014.10.176.

[9] Jiawei Feng, Jianzhong Fu, Xinhua Yao, and Yong He. Triply periodic minimal surface (TPMS) porous structures: from multi-scale design, precise additive manufacturing to multidisciplinary applications. International Journal of Extreme Manufacturing, 4:022001, 2022. doi: 10.1088/2631-7990/ac5be6.

[10] José Antonio Fernández-Fernández. TriangleMeshDistance, 2021. URL https://github.com/InteractiveComputerGraphics/TriangleMeshDistance. Online; accessed 2023-12-26.

[11] Joseph W. Fisher, Simon W. Miller, Joseph Bartolai, Timothy W. Simpson, and Michael A. Yukish. Catalog of triply periodic minimal surfaces, equation-based lattice structures, and their homogenized property data. Data in Brief, 49:109311, 2023. doi: 10.1016/j.dib.2023.109311.

[12] Paul J. F. Gandy, Sonny Bardhan, Alan L. Mackay, and Jacek Klinowski. Nodal surface approximations to the P, G, D and I-WP triply periodic minimal surfaces. Chemical Physics Letters, 336(3–4):187–195, 2001. doi: 10.1016/s0009-2614(00)01418-4.

[13] Lorna J. Gibson and Michael F. Ashby. Cellular solids: Structure and properties. Cambridge Solid State Science Series. Cambridge University Press, 2 edition, 1999.

[14] Roberto Grosso. Construction of topologically correct and manifold isosurfaces. Computer Graphics Forum, 35(5):187–196, 2016. doi: 10.1111/cgf.12975.

[15] Meng-Ting Hsieh and Lorenzo Valdevit. Minisurf – a minimal surface generator for finite element modeling and additive manufacturing. Software Impacts, 6:100026, 2020. doi: 10.1016/j.simpa.2020.100026.

[16] Meng-Ting Hsieh and Lorenzo Valdevit. Update (2.0) to MiniSurf—a minimal surface generator for finite element modeling and additive manufacturing. Software Impacts, 6:100035. doi: 10.1016/j.simpa.2020.100035.

[17] Jirawat Iamsamang and Phornphop Naiyanetr. Computational method and program for generating a porous scaffold based on implicit surfaces. Computer Methods and Programs in Biomedicine, 205:106088, 2021. doi: 10.1016/j.cmpb.2021.106088.

[18] Alistair Jones, Martin Leary, Stuart Bateman, and Mark Easton. TPMS Designer: A tool for generating and analyzing triply periodic minimal surfaces. Software Impacts, 10:100167, 2021. doi: 10.1016/j.simpa.2021.100167.

[19] Alistair David Jones. Design and additive manufacturing of TPMS-like cellular structures. Phd thesis, RMIT University, 2022. URL https://researchrepository.rmit.edu.au/esploro/outputs/9922159113301341.

[20] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. Journal of Graphics Tools, 8(2):1–15, 2003. doi: 10.

1080/10867651.2003.10487582. URL https://www.ks.uiuc.edu/Research/vmd/projects/ece498/surf/lewiner.pdf.

[21] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, page 163–169. Association for Computing Machinery, August 1987. doi: 10.1145/37401.37422.

[22] Chenxi Lu, Luthfan Adhy Lesmana, Fei Chen, and Muhammad Aziz. MD-TPMS: Multi-dimensional gradient minimal surface generator. Software Impacts, 17:100527, 2023. doi: 10.1016/j.simpa.2023.100527.

[23] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: a level set approach. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(2):158–175, 1995. doi: 10.1109/34.368173.

[24] Josiah Manson and Scott Schaefer. Isosurfaces over simplicial partitions of multiresolution grids. Computer Graphics Forum, 29(2):377–385, 2010. doi: 10.1111/j.1467-8659.2009.01607.x. URL https://people.engr.tamu.edu/schaefer/research/iso_simplicial.pdf.

[25] I. Maskery, L. A. Parry, D. Padrão, R. J. M. Hague, and I. A. Ashcroft. FLatt Pack: A research-focussed lattice design program. Additive Manufacturing, 49:102510, 2022. doi: 10.1016/j.addma.2021.102510.

[26] Liz Nickels. Software toolkits for architected materials, lightweighting, and more. Metal Powder Report, 75(4):203–206, 2020. doi: 10.1016/j.mprp.2020.04.003.

[27] Gregory M. Nielson. Dual marching cubes. In VIS '04: Proceedings of the conference on Visualization '04, page 489–496. IEEE Computer Society, October 2004. doi: 10.1109/visual.2004.28.

[28] Chen Pan, Yafeng Han, and Jiping Lu. Design and optimization of lattice structures: A review. Applied Sciences, 10(18):6374, 2020. doi: 10.3390/app10186374.

[29] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. The Visual Computer, 11(8):429–446, 1995. doi: 10.1007/bf02464333.

[30] Fernando Perez-Boerema, Mojtaba Barzegari, and Liesbet Geris. A flexible and easy-to-use open-source tool for designing functionally graded 3D porous structures. Virtual and Physical Prototyping, 17(3):682–699, 2022. doi: 10.1080/17452759.2022.2048956.

[31] David Peter. hyperfine. URL https://github.com/sharkdp/hyperfine. Online; accessed 2023-03-21.

[32] Inigo Quilez. distance functions. URL https://iquilezles.org/articles/distfunctions/. Online; accessed 2023-12-22.

[33] Inigo Quilez. interior SDFs, 2020. URL https://iquilezles.org/articles/interiordistance/. Online; accessed 2023-12-22.

[34] S. Kamal Krishnam Raju and Prasad S. Onkar. Lattice_Karak: Lattice structure generator for tissue engineering, lightweighting and heat exchanger applications. Software Impacts, 14:100425, 2022. doi: 10.1016/j.simpa.2022.100425.

[35] SBCL Authors. Steel Bank Common Lisp. URL https://www.sbcl.org/. Online; accessed 2023-12-26.

[36] Will Schroeder, Ken Martin, and Bill Lorensen. The Visualization Toolkit. An Object-Oriented Approach to 3D Graphics. Kitware, 4th edition, 2006.

[37] SPDX Workgroup. MIT License. URL https://spdx.org/licenses/MIT.html. Online; accessed 2023-12-26.

[38] Wenjin Tao and Ming C. Leu. Design of lattice structure for additive manufacturing. In 2016 International Symposium on Flexible Automation (ISFA), page 325–332. IEEE, August 2016. doi: 10.1109/isfa.2016.7790182.

[39] Mauricio Ivan Tenorio-Suárez, Arturo Gómez-Ortega, Horacio Canales, Saul Piedra, and James Pérez-Barrera. MaSMaker: An open-source, portable software to create and integrate maze-like surfaces into arbitrary geometries. SoftwareX, 19:101203, 2022. doi: 10.1016/j.softx.2022.101203.

[40] Mary Kathryn Thompson, Giovanni Moroni, Tom Vaneker, Georges Fadel, R. Ian Campbell, Ian Gibson, Alain Bernard, Joachim Schulz, Patricia Graf, Bhrigu Ahuja, and Filomeno Martina. Design for additive manufacturing: Trends, opportunities, considerations, and constraints. CIRP Annals, 65(2):737–760, 2016. doi: 10.1016/j.cirp.2016.05.004.

[41] H. G. von Schnering and R. Nesper. Nodal surfaces of Fourier series: Fundamental invariants of structured matter. Zeitschrift für Physik B Condensed Matter, 83(3):407–412, 1991. doi: 10.1007/bf01313411.

[42] Meinhard Wohlgemuth, Nataliya Yufa, James Hoffman, and Edwin L. Thomas. Triply periodic bicontinuous cubic microdomain morphologies by symmetries. Macromolecules, 34(17):6083–6089, 2001. doi: 10.1021/ma0019499.

[43] Frank W. Zok, Ryan M. Latture, and Matthew R. Begley. Periodic truss structures. Journal of the Mechanics and Physics of Solids, 96:184–203, 2016. doi: 10.1016/j.jmps.2016.07.007.