Taylor & Francis
Taylor & Francis Group

Science and Technology of Advanced Materials: Methods

ISSN: 2766-0400 (Online) Journal homepage: www.tandfonline.com/journals/tstm20

Cesogen: cellular solid generator

Paul A. Patience, Charles Audet & Bruno Blais

To cite this article: Paul A. Patience, Charles Audet & Bruno Blais (10 Oct 2025): Cesogen: cellular solid generator, Science and Technology of Advanced Materials: Methods, DOI: 10.1080/27660400.2025.2570116

To link to this article: https://doi.org/10.1080/27660400.2025.2570116

Publisher: Taylor & Francis & The Author(s). Published by National Institute for Materials Science in partnership with Taylor & Francis Group
Journal: *Science and Technology of Advanced Materials: Methods*
DOI: 10.1080/27660400.2025.2570116

This paper presents a cellular solid generator whose peerless, CLI-based DSL, specifically compatible with blackbox optimizers, can greatly promote the study, and blackbox-optimization–based design, of cellular solids.

Cesogen: Cellular solid generator

Paul A. Patience*, Charles Audet*, Bruno Blais[†]

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 can be fed to a physics simulator.

Keywords

Cellular solids; triply periodic minimal surfaces; signed distance functions; additive manufacturing; blackbox optimization.

*GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada.
Email: mailto:paul-a.patience@polymtl.capaul-a.patience@polymtl.ca, mailto:charles.audet@polymtl.cacharles.audet@polymtl.ca.

†CHAOS and Department of Chemical Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada.
Email: mailto:bruno.blais@polymtl.cabruno.blais@polymtl.ca.

Nomenclature

Abbreviations

CLI command-line interface
DSL domain-specific language
FEM finite element method
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

$\mathcal{C}$ blackbox configuration
c constraints; $c : X \to \bar{\mathbb{R}}^m$
d distance; $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$
e_k unit vector along dimension k; $e_k \in \mathbb{R}^n$
f objective function; $f : X \to \bar{\mathbb{R}}$
f_Ω SDF of Ω; $f_\Omega : \mathbb{R}^n \to \mathbb{R}$
$\tilde{f}_\Omega$ approximate SDF of Ω; $\tilde{f}_\Omega : \mathbb{R}^n \to \mathbb{R}$
I_Ω indicator function of Ω; $I_\Omega : \mathbb{R}^n \to \{0,1\}$
ℓ number of samples; $\ell \in \mathbb{Z}_{>0}$
m dimension of constraints; $m \in \mathbb{Z}_{>0}$
n dimension of point; $n \in \mathbb{Z}_{>0}$
$\mathbb{R}$ set of real numbers
$\mathbb{R}_{\geq 0}$ set of nonnegative real numbers
$\mathbb{R}_{\neq 0}$ set of nonzero real numbers
$\bar{\mathbb{R}}$ $\mathbb{R} \cup \{+\infty\}$
t time (s); $t \in \mathbb{R}_{\geq 0}$
v rotation axis; $v \in \mathbb{R}^3 \setminus \{0\}$
v orientation axis; $v \in \mathbb{R}^3 \setminus \{x \in \mathbb{R}^3 : x_1 = 0, x_2 = 0, x_3 \leq 0\}$
X domain
x point; $x \in \mathbb{R}^n$
x first coordinate of 3D point; $x \in \mathbb{R}$
y point; $y \in \mathbb{R}^n$
y second coordinate of 3D point; $y \in \mathbb{R}$
$\mathbb{Z}$ set of integers

$\mathbb{Z}_{\geq 0}$ set of strictly positive integers
z third coordinate of 3D point; $z \in \mathbb{R}$
α solid fraction; $\alpha \in [0,1]$
δ translation distance; $\delta \in \mathbb{R}^n$
θ rotation angle; $\theta \in \mathbb{R}$
κ_0 isotropic scaling factor; $\kappa_0 \in \mathbb{R}_{\neq 0}$
$\boldsymbol{\kappa}$ anisotropic scaling factor; $\boldsymbol{\kappa} \in \mathbb{R}^n_{\neq 0}$
Λ solid object; $\Lambda \subseteq \mathbb{R}^n$
Ξ shell thickness; $\Xi \in \mathbb{R}_{\geq 0}$
ξ offset radius; $\xi \in \mathbb{R}$
π half-circle constant
$\rho*$ density of cellular solid ($\mathrm{kg\,m^{-3}}$); $\rho* \in \mathbb{R}_{\geq 0}$
ρ_s density of cellular solid's constituent solid ($\mathrm{kg\,m^{-3}}$); $\rho_s \in \mathbb{R}_{\geq 0}$
τ full-circle constant (2π)
Ω solid object; $\Omega \subseteq \mathbb{R}^n$
$\partial\Omega$ 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 [1]. 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 [1]. Advances in additive manufacturing [2] 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) [3]. 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 [3]. 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 and library that takes a textual or code-based description of a cellular solid, e.g., its name and various transformations applied to it, converts it into a signed distance function (SDF), and finally generates 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;
- the ability to fill arbitrary geometries with a cellular solid, and also to generate graded, heterogeneous and multiscale cellular solids;
- 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. We have evaluated various standalone, freely available generators in order to place Cesogen among its peers, noting their language of implementation, interaction via command-line interface (CLI), graphical user interface (GUI) or library, and additional features (Table 1). Commercial cellular solid generators are usually a smaller part of a more general design tool, and include [4, 5]: Optistruct (Altair), Netfabb and Within (Autodesk), Sulis (Gen3D), 3-Matic (Materialise), nTop (nTopology) [6], 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 [5].

Table 1: Various standalone, freely available cellular solid generators and their implementation languages, whether they offer batch-like CLI, GUI or library form, their features, year of publication of supporting article, or if none, initial release, and finally reference. Languages are one or more of C++ (C), Common Lisp (L), MATLAB (M), Python (P), Mathematica (W) and unknown (U). Having a CLI presupposes it is compatible with batch processing, i.e., requiring no user input. Generators with neither CLI, GUI or library form require editing the source code before running. Features may be one or more of extensive library of cellular solids (E), filling arbitrary solids (F), generating graded (G), heterogeneous (H), hierarchical (I) and multisymmetrical (S) cellular solids, applying mathematical transformations (T) and having advanced modeling features (M). We determined the features from a cursory inspection of the papers presented, the documentation and, when those were insufficient or incomplete and the source code was available, the source code. TPMS Studio in particular may have more features than those listed.

Name	Lang	CLI	GUI	Lib	Feats	Year	Ref
ScaffoldStructures	P		●			2014	[7]
MSLattice	M		●		G	2020	[8]
MiniSurf	M		●			2020	[9, 10]
TPMS-Modeler	M					2021	[11]
Scaffolder	CP	●	●	●	F	2021	[4]
TPMS Designer	M		●		T	2021	[12, 13]
RegionTPMS	W				G	2021	[14]
FLatt Pack	M		●		EFG	2022	[15]
TPMS Scaffold	M					2022	[16]

ASLI	C	●	●	●	FGH	2022	[17]
Microgen	P			●	EFIT	2022	[18]
MaSMaker	M		●		FGH	2022	[19, 20]
Lattice_Karak	M		●		GHI	2022	[5]
TPMSgen	P		●			2023	[21]
MD-TPMS	M		●		G	2023	[22]
TPMS Studio	U		●		EM	2023	[23]
TPMS_Scaffold_Generator	M		●		FGS	2024	[24]
LattGen	M		●		EFG	2024	[25]
Cesogen	L	●		●	EFIT	2025	

The primary features an optimization-focused generator needs in order to be usable are the ability to generate cellular solids and a way for the optimizer to be able to operate it without any user intervention, e.g., via batch-like CLI or library, which eliminates all evaluated freely available generators except for Scaffolder, ASLI and Microgen. Furthermore, since Microgen is available only as a library, using it at all involves writing scripts, which may be inconvenient depending on the application.

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. Cesogen was designed from the start to be adapted to computer-guided optimization, and features a CLI-based, domain-specific language (DSL) which allows the user to generate complex geometries on the fly without the use of a separate modeling tool. In particular, like Microgen, Cesogen is able to robustly generate hierarchical cellular solids of arbitrary depth via direct intersection, whereby the underlying SDFs are combined at runtime and the resulting SDF written directly to disk. Any generator that can fill arbitrary solids is also able to generate hierarchical cellular solids via sequential intersection, whereby each successive intersection is written to disk and reread for the next, but this method incurs a cost on the performance and quality of the results.

This article begins by presenting the theory required to understand the operation of Cesogen — starting with a description of SDFs, which are how cellular solids are represented internally by Cesogen, and then of cellular solids themselves — 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 Signed distance functions

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

$$f_\Omega(x) = (2I_\Omega(x) - 1) \inf_{y \in \partial\Omega} d(x, y) \tag{1}$$

where $\partial\Omega$ is the boundary of Ω and I_Ω is the indicator function of Ω, i.e., $I_\Omega(x)$ is 1 if $x \in \Omega$ and 0 otherwise. In other words, f_Ω 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 [26, 27], but others use the opposite convention [28, 29].

The SDFs of primitive solid objects are generally derived mathematically [30]. The SDFs

of solid objects represented as meshes are brute-forced by calculating the shortest distance from x to the polygons on the surface of 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 [31].

Once we have the SDFs f_Ω and f_Λ corresponding to solid objects $\Omega \subseteq \mathbb{R}^n$ and $\Lambda \subseteq \mathbb{R}^n$, we can transform and combine Ω and Λ by manipulating their SDFs or the points provided to them. Most operations on SDFs return approximate rather than exact SDFs, where an exact SDF is one which returns an exact distance, the gradient of which is always of length 1 [32, 26, 30, 33]. These approximate SDFs, for which the boundary is correct but the inner and outer distances may not be, are marked with a tilde, e.g., $\tilde{f}_\Omega$.

We can apply the usual mathematical transformations translation, scaling and rotation to Ω by transforming the point passed to f_Ω:

• Translation by distance $\delta \in \mathbb{R}^n$:

$$f_{\Omega'}(x) = f_\Omega(x - \delta) \tag{2}$$

• Isotropic scaling by factor $\kappa_0 \in \mathbb{R}_{\neq 0}$:

$$f_{\Omega'}(x) = |\kappa_0| \, f_\Omega(x / \kappa_0) \tag{3}$$

• Anisotropic scaling by factor $\kappa \in \mathbb{R}^n_{\neq 0}$:

$$\tilde{f}_{\Omega'}(x) = \min_{i \in \{1,\ldots,n\}} |\kappa_i| \, f_\Omega(x') \tag{4}$$

where $x_{i'} = x_i / \kappa_i$ for $i \in \{1, \ldots, n\}$

• Three-dimensional rotation about axis $v \in \mathbb{R}^3 \setminus \{0\}$ by angle $\theta \in \mathbb{R}$:

$$f_{\Omega'}(x) = f_\Omega((A + B)x) \tag{5}$$

where $A_{ij} = \hat{v}_i \hat{v}_j (1 - \cos\hat{\theta})$, $\hat{v} = v / \|v\|$ and

$$B = \begin{pmatrix} \cos\theta & -\hat{v}_3 \sin\theta & \hat{v}_2 \sin\theta \\ \hat{v}_3 \sin\theta & \cos\theta & -\hat{v}_1 \sin\theta \\ -\hat{v}_2 \sin\theta & \hat{v}_1 \sin\theta & \cos\theta \end{pmatrix}$$

The three-dimensional orientation along axis $v \in \mathbb{R}^3 \setminus \{x \in \mathbb{R}^3 : x_1 = 0, x_2 = 0, x_3 \leq 0\}$ is a variant of rotation which is useful for modeling with SDFs. It consists of rotating Ω about axis $e_3 \times v$ by angle $\arccos(v_3 / \|v\|)$, where $e_3 = (0,0,1)$, such that Ω' points in the direction of v rather than e_3.

Some more niche transformations, usually seen when modeling with SDFs, can be applied by transforming f_Ω directly:

• Offset by radius $\xi \in \mathbb{R}$ (equivalent to morphological dilation if $\xi \geq 0$, morphological erosion otherwise):

$$\tilde{f}_\Omega(x) = f_\Omega(x) - \xi \tag{6}$$

• Shell of thickness $\Xi \in \mathbb{R}_{\geq 0}$:

$$\tilde{f}_{\Omega'}(x) = f_{\Omega}(x) - \tfrac{1}{2}\Xi \tag{7}$$

The basic implementation of set-theoretic operations on objects represented by SDFs is the following, though variants exist with different properties [29]:

• Complement:

$$f_{\mathbb{R}^n \setminus \Omega} = -f_{\Omega} \tag{8}$$

• 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.

SDFs are ultimately useful when converted, or contoured, to polygon or polyehdral meshes used in simulations or manufactured. Various methods exist for contouring SDFs to triangle meshes, including marching cubes [34, 35, 36, 37, 38] (the predecessor of them all), dual marching cubes [39], and more advanced methods [40, 41]. SDFs can also be contoured to tetrahedral meshes via isosurface stuffing [42]. Contouring methods differ in number and kind (e.g., triangle or tetrahedral) of cells generated and performance, though no research has been done to determine which methods would be ideal for cellular solids in particular.

3 Cellular solids

Gibson and Ashby [1] 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 [43, 44, 45].

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 $\rho^* \mathbb{R}_{\geq 0}$ is the cellular solid's density and $\rho_s \in \mathbb{R}_{\geq 0}$ the density of the solid constituting the cellular solid

[1].

Lattice structures are further divided into two kinds: strut-based and TPMS-like. Strut-based cellular solids are composed of combinations of struts and have been studied for longer because they have traditionally been easier to manufacture; TPMS-like cellular solids are based on smooth surfaces and 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 [46], 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" • [47]. The terms from crystallography and mathematics are likely inspired from this definition.

TPMS-like cellular solids are derived from the equations of triply periodic, implicit surfaces, which include TPMSs and other triply periodic (non-minimal) surfaces. Indeed, TPMSs can be generalized to non-minimal surfaces of the same family [48]. The equations of TPMSs can be approximated by taking the first few terms of the Fourier series constituting the periodic nodal surface approximations of those surfaces [49, 50, 51]. Sample equations are presented in Section 1.

TPMS-like surfaces give rise to four different kinds of TPMS-like cellular solids, depending on which side of the surfaces we treat as solid or whether we take their shell of thickness $\Xi \in \mathbb{R}_{\geq 0}$. Each of these kinds can additionally be offset by radius $\xi \in \mathbb{R}$ before making solid. The four kinds of TPMS-like cellular solids are [52] (with names and notation simplified and adapted to conform to our SDF terminology and notation, and original names in parentheses):

• Endoskeleton (triply periodic endoskeleton, TPnS):

$$f_\Omega(x) \leq \xi \tag{12}$$

• Exoskeleton (triply periodic exoskeleton, TPxS):

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

• Shell (triply periodic surface, TPSf):

$$\left| f_\Omega(x) \right| \leq \tfrac{1}{2}\Xi \tag{14}$$

• Eccentric shell (triply periodic eccentric surface, TPcS):

$$\left| f_\Omega(x) - \xi \right| \leq \tfrac{1}{2}\Xi \tag{15}$$

The offset radius is also known as the isovalue or level set value [52, 8] in the skeletal inequalities and as the eccentricity [52] in the eccentric shell inequality. The offset radius and shell thickness control the relative density of the cellular solid, and the offset also changes the shape of the shell walls.

Complex cellular solids can be composed from primitive cellular solids by manipulating their SDFs. Arbitrary meshes can be filled with cellular solids by computing the meshes' SDFs and intersecting them with the cellular solids' SDFs. Graded cellular solids are obtained by replacing constant transformations, e.g., translation, scaling, rotation, by functions [3, 8]. Heterogeneous cellular solids consist of fusions of cellular solids, and can be obtained via a

sigmoid function [8]. Multiscale, or hierarchical, cellular solids are cellular solids within cellular solids [3], and can be obtained by intersecting cellular solids having different unit cell sizes.

4 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.

4.1 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 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 — in fact, TPMS approximations are by definition SDFs, which makes representing meshes as SDFs in Cesogen the natural choice. 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 the following phases: (1) Convert meshes; (2) combine SDFs; and (3) contour. 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 the first two phases eagerly, i.e., it converts meshes and combines them as soon as it can, so the phases are actually interleaved.

Cesogen accepts as input arbitrary solids represented as meshes. Supported input formats include OBJ, OFF, PLY and STL. Cesogen converts the input meshes to SDFs by computing an intermediate sphere-based bounding volume hierarchy for speeding up proximity queries based on a port of TriangleMeshDistance [53] to Common Lisp. The signs of the computed distances are resolved via the algorithm described by Bærentzen and Aanæs [31]

Cesogen also boasts an extensive library of cellular solids, currently consisting of those derived from most of the TPMS-like surfaces described by Fisher et al. [52] (Table 2). Users may define their own cellular solids by providing expressions for their corresponding SDFs directly to Cesogen instead of meshes, or saving the expressions in files which are then provided, and more TPMS-like cellular solids will will be added as they are discovered.

In the SDF combination phase, Cesogen supports various operations on the SDFs, including the translation, scaling, rotation, orientation, offset, shell and the set-theoretic operations complement, union, intersection and difference on their corresponding objects (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 triangle mesh. Supported output formats include OBJ, OFF, PLY, STL and legacy VTK. The choice of contouring algorithm has an impact on the number of mesh cells composing it, which in turn has an impact on the running time of the physics solver. Cesogen currently uses a robust, marching cubes–like algorithm [36], but further algorithms will be added which offer different tradeoffs.

Performance is important only until the cellular solid generation step takes a negligible amount of time compared to the rest of the current optimization step. Cesogen has been developed with a focus on performance in order to reach this threshold.

Cesogen is multiplatform; it runs on various flavors of Linux and also macOS and Windows. Specific installation instructions are described in the manual.

Finally, Cesogen is freely available, open source software licensed under the MIT (more specifically, the Expat) [54], a permissive license which allows commercial use with few restrictions. Its homepage is https://git.sr.ht/ p̃ aulapatience/cesogen.

4.2 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 launches Cesogen with the appropriate command-line arguments. Therefore, the most important part of Cesogen's interface, for the average client, is its CLI.

Cesogen's CLI is a mixfix, stack-oriented DSL which dispenses with the need to have nested parentheses on the command-line. The command-line arguments can be one the following kinds: command-line options, SDF specifiers, and SDF operations and operation arguments. The SDF operations are mixfix and termed SDF-nullary, SDF-unary and SDF-binary based on how many SDFs they take on the left; some operations also take non-SDF arguments on the right, provided as separate command-line arguments. The command-line options are discussed at the end of this section.

SDF specifiers are names of solid objects whose SDFs are known to Cesogen. The primitive solids supported by Cesogen are currently:

- ball, the unit ball 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.

TPMS-like cellular solids are named after their corresponding surfaces and prefixed with tpms., so gyroid becomes tpms.gyroid, P becomes tpms.p and C(Y) becomes tpms.c-y. The full list of TPMS-like cellular solids supported by

Cesogen is documented in the manual.

To generate a ball, run: cesogen ball

cesogen ball

SDFs may also be specified by the following SDF-nullary operations:

- expression, taking a string argument on the right in the form of an infix expression corresponding to the left-hand side of the equation of an implicit surface, with right-hand side 0, which describes an SDF, e.g., 'x+y+z-1'. These expressions accept the point coordinates x, y, z,

the standard arithmetic operators, and some basic mathematical constants and operations, e.g., pi and trigonometric functions. The syntax is described in more detail in the manual.

• file, taking a string argument on the right corresponding to the name of a mesh from which Cesogen extracts the SDF. The input mesh format is detected from the extension.

When an SDF specifier is provided, the corresponding SDF is pushed onto a stack of SDFs, which is operated upon by the SDF operations. SDF-unary operations pop the stack once, modify the SDF, and push it back onto the stack. SDF-binary operations pop the stack twice, combine the SDFs accordingly, and push the result back onto the stack.

The SDF-unary operations and the arguments they take on the right are:

• complement, none
• offset by radius $\xi \in \mathbb{R}$
• shell of thickness $\Xi \in \mathbb{R}_{\geq 0}$
• move (translate) by distance $\delta \in \mathbb{R}^3$
• scale isotropically by factor $\kappa_0 \in \mathbb{R}_{\neq 0}$
• scale anisotropically by factor $\kappa \in \mathbb{R}^3_{\neq 0}$
• rotate about axis $v \in \mathbb{R}^3 \setminus \{0\}$ by angle $\theta \in \mathbb{R}$
• orient along axis $v \in \mathbb{R}^3 \setminus \{x \in \mathbb{R}^3 : x_1 = 0, x_2 = 0, x_3 \leq 0\}$

The components of three-dimensional arguments are listed in one command-line argument and separated by commas. Numeric arguments may also be expressions, though without any point coordinates.

To generate a ball of radius 2, run:

```
cesogen ball scale 2
```

The SDF-binary operations are union, intersection and difference. They take no arguments on the right.

A more complex SDF combination is (Figure 1):

```
cesogen \
tpms.gyroid file mesh1.ply intersection \
tpms.p file mesh2.ply intersection \
union
```

Figure 1: Tree representation of the operations Cesogen performs when invoked as ‘cesogen tpms.gyroid file mesh1.ply intersection tpms.p file mesh2.ply 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.

Cesogen detects the bounding box of the resulting SDF automatically, with arbitrary expressions being treated as unbounded. When Cesogen writes the SDF to disk, it maps $-\infty$ to -1 and ∞ to 1. In other words, the default bounding box of TPMS-like SDFs is the same as that

of box.

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. 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 expressions.

The -b option specifies an explicit bounding box; it consists of six comma-separated components corresponding to the three lower and three upper bounds. The -s option specifies the number of samples to take of the SDF during contouring. It may consist of one component or three comma-separated components; in the former case, the value is a total number of samples distributed as uniformly as possible along all dimensions, and in the latter, the number of samples along each dimension. The -o option specifies the output file, overriding the default of out.ply; the output mesh format is detected from the extension.

To generate endoskeletal, exoskeletal and eccentric shell variants of the TPMS-like Ω with offset radius ξ, shell thickness Ξ, and corresponding equations $f_\Omega(x) \le \xi$, $-f_\Omega(x) \le -\xi$ and $\left| f_\Omega(x) - \xi \right| \le \frac{1}{2}\Xi$, run, respectively:

cesogen Ω offset ξ
cesogen Ω offset ξ complement
cesogen Ω offset ξ shell Ξ

In other words, all TPMS-like cellular solids generated by Cesogen are by default endoskeletons.

4.3 Guiding principles

Cesogen and its CLI-based DSL were carefully designed according to a set of guiding principles which may not be immediately apparent upon reading a description of Cesogen's usage. The guiding principles are the following: (1) suitability for blackbox optimization; (2) ergonomics; and (3) composability.

The original impetus for developing Cesogen was the desire to solve cellular solid problems via blackbox optimization algorithms not based on metaheuristics. Metaheuristic algorithms can start from a small sample of pregenerated, possibly via GUI, cellular solids and estimate the properties of points proposed by the optimizer without having to generate new cellular solids. Without this estimation, the blackbox must be able to generate the meshes automatically. The principle of suitability for blackbox optimization mandates providing a text-based interface, i.e., CLI- or library-based.

The reason for the principle of ergonomics is to make Cesogen accessible to as large an audience as possible, so as to encourage more research in the field of cellular solids. Having chosen to provide a CLI, this principle mandates the syntax be word-based, to avoid the proliferation of quotes that would be necessary to escape parentheses and other protected shell characters that might be used in an imperative DSL. (This principle also favors providing a CLI over a library, though Cesogen provides both.)

The reason for the principle of composability is to make Cesogen able to interpret the textual representation of any cellular solid, so as to limit the proliferation of cellular solid

generators that support only some kinds of cellular solids. Having chosen a word-based syntax, this principle mandates the DSL be stack-oriented, to allow arbitrarily long combinations of SDFs.

Cesogen's DSL is called mixfix because operators can take arguments on the left and on the right. SDFs are always provided on the left, which allows arbitrarily deep SDF combinations via SDF-binary operators while avoiding parentheses in the syntax which would otherwise be necessary to handle operator precedence. Additional arguments are provided on the right because the resulting syntax follows command-line conventions and reads more naturally. For example, the command

```
cesogen cylinder orient 1,0,0 scale 1/2
```

can be read aloud as "generate a cylinder oriented along the x axis with a radius of one-half" • .

The stack-oriented nature of the DSL becomes apparent when invoking SDF-binary operators such as union and intersection, which pop two SDFs on the stack and push the resulting SDF back onto it. In particular, any operator that takes an SDF on the left effectively takes on the left any sequence of words that produces an SDF. Thus, we can build models piecemeal by starting with the SDFs of primitive solids and progressively transforming, duplicating, and combining them. For example, the following command produces a pair of cylinders identical to the one above but shifted along the y axis:

```
cesogen \
cylinder orient 1,0,0 scale 1/2 move 0,-2,0 \
cylinder orient 1,0,0 scale 1/2 move 0,2,0 \
union
```

The stack becomes harder to keep track of as more SDFs are combined, but this can be mitigated by storing custom SDFs in files and passing them to Cesogen via the *@file* option. Also, an operator may be added to Cesogen in the future which would name the SDF on the left, popping it from the stack and allowing further references to it by name.

Though the DSL is CLI-based insofar as that is how the Cesogen CLI takes it as input, nothing prevents its use as a general textual representation of SDFs, e.g., in configuration files of cellular solid–based blackboxes which invoke Cesogen on the provided textual representation. Indeed, the Cesogen library accepts the DSL as a string representing an SDF. The generality of the DSL makes it more convenient to model complex shapes incorporating cellular solids directly via Cesogen and results in more accurate results, because an intermediate model need not be transformed into an SDF — which transformation is inherently lossy — before intersecting it with the SDF of a cellular solid.

Finally, of the cellular solid generators listed in the introduction, only Cesogen satisfies all three of the guiding principles described here. None of the GUI-based generators are suitable for blackbox optimization, which eliminates all the commercial generators listed and all the freely available generators listed except Scaffolder, ASLI and Microgen. ASLI supports only a predetermined set of cellular solids to fill external solids with, without any way to extend its library without recompiling the application, and its input is strictly via configuration file, thus it violates the second and third principles. Scaffolder supports custom SDFs in Lua files, and

Microgen is a Python library, so both are theoretically as composable as Cesogen, but they require writing code, thus they violate the second principle.

5 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.

All experiments were performed on a machine with the following specifications:

- OS: Chimera Linux x86_64
- Host: 20UH000CUS ThinkPad T14s Gen 1
- Kernel: 6.13.4-0-generic
- CPU: AMD Ryzen 7 PRO 4750U with Radeon Graphics (16) @ 1.700GHz
- Memory: 15210MiB

5.1 Examples of TPMS-like cellular solids

Cesogen supports many TPMS-like cellular solids; the expressions of their corresponding surfaces are listed in Table 2, 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 and z rather than x_i to conform to Cesogen's equation syntax.

Table 2: Expressions of TPMS-like surfaces described by Fisher et al. fisher-2023-catal-lattic-struc. The right-hand sides of the corresponding implicit surface equations are 0.

Name	Expression
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)	$0.3C_xC_yC_z + 0.1C_{2x}C_{2y}C_{2z} + 0.2(C_x + C_y + C_z) + 0.1(C_{2x} + C_{2y} + C_{2z}) + 0.05(C_{3x} + C_{3y} + C_{3z}) + 0.1(C_xC_y$

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	$S_xS_yS_z + C_xS_{2y} + C_yS_{2z} + C_zS_{2x} + C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
±Y	$2C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
C(±Y)	$-2C_xC_yC_z + S_{2x}S_y + S_{2y}S_z + S_{2z}S_x$
C(I2-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	$-2(C_xC_y + C_yC_z + C_zC_x) - 2(C_{2x} + C_{2y} + C_{2z}) + C_{2x}C_y + C_{2y}C_z + C_{2z}C_x - (C_xC_{2y} + C_yC_{2z} + C_zC_{2x})$

A selection of TPMS-like cellular solids generated by Cesogen is presented in Figures 2 and 3.

Figure 2: Showcase of various TPMS-like endoskeletons generated by the command ‘cesogen -s ℓ Ω offset ξ ‘ with by default $\ell = 60^3$ and $\xi = 0.35$.

Figure 3: Showcase of various TPMS-like shells generated by the command ‘cesogen -s ℓ Ω shell Ξ ‘ with $\Xi = 2\xi$ and by default $\ell = 60^3$ and $\xi = 0.35$.

5.2 Filling meshes

Cesogen is able to fill existing meshes with arbitrary cellular solids via the intersection operation, invoked as ‘cesogen $\Omega\ldots$ file Λ intersection’ where $\Omega\ldots$ is a sequence of Cesogen operations representing the cellular solid with which to fill the mesh named Λ.

For example, to fill a baluster model with a Neovius endoskeleton (Figure 4), run

```
cesogen -s ℓ tpms.neovius scale κ0 \
    file baluster.obj \
```

intersection

with samples ℓ to preserve finer features and scale κ_0 to adjust the cellular solid's cell size to the baluster's size. Values for (ℓ, κ_0) could be for example $(80^3, 0.065)$ or $(120^3, 0.03)$. More samples should be taken for smaller scales to appropriately capture the features of the cellular solid. To fill an acorn model with an OCTO endoskeleton (Figure 5), run

```
cesogen -s ℓ tpms.octo scale κ0 \
        file acorn.obj \
        intersection
```

where (ℓ, k_0) could be for example $(80^3, 0.009)$ or $(120^3, 0.003)$.

Figure 4: Neovius baluster, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh [55] and (b, c) filled meshes generated by the command 'cesogen -s ℓ tpms.neovius scale κ_0 file baluster.obj intersection', which took 1.3 and 4.2 to generate, respectively.

Figure 5: OCTO acorn, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh [56] (b, c) filled meshes generated by the command 'cesogen -s ℓ tpms.octo scale κ_0 file acorn.obj intersection', which took 1.3 and 4.1 to generate, respectively.

5.3 Generating hierarchical cellular solids

Cesogen generates hierarchical cellular solids via direct intersection (Figure 6). It combines the constituent SDFs at runtime and writes the resulting SDF directly to disk. Cellular solid generators which are unable to combine multiple SDFs are limited to sequential intersection, i.e., computing one intersection, writing it to disk, and repeating the process for each additional SDF.

Figure 6: Hierarchical cellular solid consisting of the intersection of (a) a truncated ball, (b) a P endoskeleton, (c) a finer D endoskeleton, and (d) an even finer gyroid endoskeleton. 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 -b -4,-4,-4,4,4,4 -s 1303 ball scale 5 tpms.p scale 4 tpms.d scale 3.75 tpms.gyroid scale 1.25 intersection intersection intersection'.

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 solids in Figures 6f,g 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.

The hierarchical experiment's errors are displayed in Figure 7. Only the data for the direct case is presented, because the zeroth step is the same operation in the direct and sequential cases; the error in the first step is almost identical; and from the second step the sequential generation fails for trying to divide by zero. The errors increase monotonously, because each additional combination adds more detail to the resulting object, making it harder to approximate with a mesh contoured from a given grid size. The details lie along the edges of the objects; there is more error along the edges of the sequential version of $\Omega_B \cap \Omega_P$ (Figure 8).

Figure 7: Direct hierarchical L^2 point errors (●), L^2 cell errors (▲), max point distances (○) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.

Figure 8: Colored meshes corresponding to $\Omega_B \cap \Omega_P \cap \Omega_D$. The log-scale coloring corresponds to the distance from the points to the reference SDF.

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

The roundtrip experiment's errors are displayed in Figure 9. The data stops at the 27th roundtrip because the 28th 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).

Figure 9: Roundtrip L^2 point errors (●), L^2 cell errors (▲), max point distances (○) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.

Figure 10: Meshes before the first and 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 -b -4,-4,-4,4,4,4 -s 100^3 ball scale 4 tpms.gyroid scale 4 intersection’.

The roundtrip results become even worse when starting with a more complex initial mesh, e.g., the Stanford bunny rather than a ball. 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.

5.4 Performance and limitations

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

We ran Cesogen via hyperfine [57] and generated gyroids with progressively more samples by the command

cesogen -s ℓ tpms.gyroid

with $\ell \in \{25^3, 50^3, 100^3, 200^3, 300^3, \ldots, 800^3\}$. The mesh generated from 800^3 samples contained 16.2 million cells and took 79 to generate (Figure 11). Finer meshes than that required more than 16 GiB of memory.

Figure 11: Mean time taken for Cesogen to generate gyroids via ‘cesogen -s ℓ tpms.gyroid’ with $\ell \in \{25^3, 50^3, 100^3, 200^3, 300^3, \ldots, 800^3\}$. Samples beyond $800Â^3$ required more memory than that available on the test machine.

Further limitations of Cesogen include the following:

• 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.

• Generating hierarchical cellular solids requires more samples in order to capture the finer features, which increases computation time. This could be mitigated by contouring algorithms which take advantage of the nature of SDFs to evaluate less points on the grid [41].

Although Cesogen is not parallelized, this is not a limitation in the context of blackbox optimization: direct search algorithms are embarrassingly parallelizable because they can evaluate several blackbox simulations concurrently [58], and exploiting parallel resources at the

solver level is generally preferable to parallelizing individual components of the blackbox.

6 Blackbox optimization

Cesogen's raison d'être is to facilitate the blackbox optimization of cellular solids. Gradient information for the objective function and constraints of blackbox optimization problems are usually either unavailable or hard to compute, and thus dedicated algorithms exist for this class of problems [59]. Examples of such problems are computer simulations, which arise in many engineering applications [60]. In this section, we demonstrate Cesogen's capabilities by optimizing a cellular solid much more conveniently than would be possible by hand, i.e., without Cesogen.

The optimization process (Figure 12) consists of two parts: the blackbox, a program which simulates the compression of a cellular solid, and the blackbox optimizer, which launches the blackbox over and over in order to determine the best cellular solid given some constraints.

Figure 12: Block diagram of the optimization process. The generator consists of Cesogen and [61], the FEM solver is MFEM [62], and the optimizer is NOMAD [63]. The optimizer starts by feeding a point $x \in X$ to the blackbox, where X is the domain of the objective function $f : X \to \bar{\mathbb{R}}$ and constraints $c : \mathrm{X} \to \bar{\mathbb{R}}^m$, with $\bar{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$. From this point, and further informed by the configuration $\mathcal{C}$, the blackbox produces the values $f(x)$ and $c(x)$ which are inspected by the optimizer to propose the following point, and the cycle repeats until a stopping criterion is met.

The blackbox simulates the compression of a cellular solid described by the input variables $x \in X$ and a configuration $\mathcal{C}$. It invokes Cesogen to generate a triangle mesh corresponding to the cellular solid, converts it to a tetrahedral mesh with Gmsh [61], and then simulates its compression by calling MFEM [62], a finite element method (FEM) library, to solve the linear elastic equations in a way similar to MFEM's example 2. The blackbox applies a force evenly distributed over the top surface of the material and returns several values describing the resulting mesh, from which we can extract the values of the objective function $f : X \to \bar{\mathbb{R}}$ and constraints $c : X \to \bar{\mathbb{R}}^m$, where $\bar{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$. We have not done a mesh convergence analysis of the blackbox because this is a qualitative demonstration of Cesogen.

The specific optimization problem consists of minimizing the solid fraction $\alpha \in [0,1]$ of an hourglass-shaped block (Figure 13a) filled with an exoskeletal cellular solid and to which we apply a downwards vertical compression on the top surface. The cellular solid is described by the input variables cellular solid kind — one of gyroid, IWP and FRD —, offset radius $\xi \in [-1,3]$ and scale $\kappa_0[0.5,2]$. The kind variable is categorical and is handled by launching one instance of the blackbox optimizer per possible value; the other variables are continuous. Various constraints exist to ensure that the blackbox produces a physically valid mesh; they include a constraint on the average displacement of the top surface induced by the compression and a constraint on the maximum displacement of any point in the domain.

The triangle meshes corresponding to the initial and filled hourglasses can be generated

with the standalone Cesogen command

```
cesogen -s 40^3 \
cylinder move 0,-2,0 \
cylinder move 0,2,0 union \
scale 1/2 orient 1,0,0 complement \
box intersection \
[Ω offset Î¾ complement scale κ0 \
intersection]
```

where the section in brackets is omitted for the plain hourglass and kept, though without the brackets, for a TPMS-like Ω with offset radius ξ and scale κ_0. Cesogen generates the hourglasses in one step via direct intersection.

The blackbox optimizer is NOMAD [63], an open-source implementation of the MADS algorithm [64]. We left the algorithm parameters to their default values, determined the initial point from a set of 20 LHS evaluations, and set the evaluation budget (limit), which includes the LHS evaluations, to 50 per cellular solid kind.

The optimization of the three cellular solids took 7.5 in total (Table 3). The best cellular solids found were, in increasing order of solid fraction, IWP (Figure 13c), FRD (Figure 13d) and gyroid (Figure 13b).

Figure 13: Initial hourglass (a) and best solutions (b, c, d) to the optimization problem for each cellular solid kind. In the convergence plot (e), the filled shapes represent feasible solutions, the solid lines the best solution found so far, and infeasible solutions are omitted.

Table 3: Solutions to the optimization problem for each cellular solid kind, along with the time t taken for each.

Kind	ξ	κ_0	α	t / h
Gyroid	0.037	1.3	0.38	1.8
IWP	0.52	0.50	0.31	2.9
FRD	0.56	1.1	0.31	2.9

The convergence plot for each cellular solid kind (Figure 13e) illustrates the feasible solutions found by the optimizer and the evolution of the solid fraction throughout the optimization. Infeasible solutions are omitted from the graph, and they can account for more than 50% of the total evaluations; they are the reason the best-solution lines in the convergence plot do not start at blackbox evaluation 1. There are two main causes for infeasible solutions: non-physical simulations, i.e., the mesh would not be manufacturable, and mesh generation errors. Both are handled by NOMAD.

This example illustrates how Cesogen can be employed to perform a complex optimization task. A GUI-only generator would be inappropriate, as the user would have to generate each mesh by hand. Here the blackbox optimizer tested about 150 different meshes, but

introducing additional cellular solid kinds and allowing more blackbox evaluations would increase this number to the thousands.

7 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 via direct intersection. 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. Furthermore, it cannot generate tetrahedral meshes, which means a third-party tool is required in order to run FEM simulations on Cesogen output.

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

- more cellular solids, in particular strut-based cellular solids;
- more triangle-based contouring algorithms [40, 41, 39], possibly parallel ones;
- a tetrahedron-based contouring algorithm [42];
- 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.

Acknowledgments

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 (Audet) 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 openly available at https://doi.org/10.5281/zenodo.15270544 [65], or directly from its repository https://git.sr.ht/ p̃aulapatience/cellul-solid-gener-results.

CRediT statement

- Conceptualization: PAP, CA, BB
- Data curation: PAP
- Formal analysis: PAP
- Funding acquisition: CA, BB
- Investigation: PAP
- Methodology: PAP
- Project administration: CA, BB
- Software: PAP
- Resources: CA, BB
- Supervision: CA, BB
- Validation: PAP
- Visualization: PAP
- Writing — original draft: PAP
- Writing — review & editing: PAP, CA, BB

Conflict of interest

The authors declare no conflict of interest.

References

[1] L. J. Gibson and M. F. Ashby. *Cellular solids: Structure and properties*. 2nd ed. Cambridge Solid State Science Series. Cambridge, England: Cambridge University Press, Aug. 1999. isbn: 978-0-521-49911-8.

[2] M. K. Thompson et al. "Design for Additive Manufacturing: Trends, opportunities, considerations, and constraints"•. In: *CIRP Annals* 65.2 (2016), pp. 737–760. doi: https://doi.org/10.1016/j.cirp.2016.05.004 10.1016/j.cirp.2016.05.004.

[3] J. Feng, J. Fu, X. Yao, and Y. He. "Triply periodic minimal surface (TPMS) porous structures: from multi-scale design, precise additive manufacturing to multidisciplinary applications"•. In: *International Journal of Extreme Manufacturing* 4, 022001 (Mar. 2022). doi: https://doi.org/10.1088/2631-7990/ac5be6 10.1088/2631-7990/ac5be6.

[4] J. Iamsamang and P. Naiyanetr. "Computational method and program for generating a porous scaffold based on implicit surfaces"•. In: *Computer Methods and Programs in*

Biomedicine 205, 106088 (June 2021). doi: https://doi.org/10.1016/j.cmpb.2021.106088 10.1016/j.cmpb.2021.106088.

[5] S. K. K. Raju and P. S. Onkar. "Lattice_Karak: Lattice structure generator for tissue engineering, lightweighting and heat exchanger applications" • . In: *Software Impacts* 14, 100425 (Dec. 2022). doi: https://doi.org/10.1016/j.simpa.2022.100425 10.1016/j.simpa.2022.100425.

[6] L. Nickels. "Software toolkits for architected materials, lightweighting, and more" • . In: *Metal Powder Report* 75.4 (July 2020), pp. 203–206. doi: https://doi.org/10.1016/j.mprp.2020.04.003 10.1016/j.mprp.2020.04.003.

[7] 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" • . In: *Procedia Technology* 16 (2014), pp. 1542–1547. doi: https://doi.org/10.1016/j.protcy.2014.10.176 10.1016/j.protcy.2014.10.176.

[8] O. Al-Ketan and R. K. Abu Al-Rub. "MSLattice: A free software for generating uniform and graded lattices based on triply periodic minimal surfaces" • . In: *Material Design & Processing Communications* 3.6 (Oct. 2020). doi: https://doi.org/10.1002/mdp2.205 10.1002/mdp2.205.

[9] M.-T. Hsieh and L. Valdevit. "Minisurf – A minimal surface generator for finite element modeling and additive manufacturing" • . In: *Software Impacts* 6, 100026 (Nov. 2020). doi: https://doi.org/10.1016/j.simpa.2020.100026 10.1016/j.simpa.2020.100026.

[10] M.-T. Hsieh and L. Valdevit. "Update (2.0) to MiniSurf—A minimal surface generator for finite element modeling and additive manufacturing" • . In: *Software Impacts* 6, 100035 (Nov. 2020). doi: https://doi.org/10.1016/j.simpa.2020.100035 10.1016/j.simpa.2020.100035.

[11] D. Morton. *TPMS-Modeler*. Version 2. Initial version 1 on 2021-01-04. Oct. 6, 2023. url: https://github.com/danielpmorton/TPMS-Modeler.

[12] A. Jones, M. Leary, S. Bateman, and M. Easton. "TPMS Designer: A tool for generating and analyzing triply periodic minimal surfaces" • . In: *Software Impacts* 10, 100167 (Nov. 2021). doi: https://doi.org/10.1016/j.simpa.2021.100167 10.1016/j.simpa.2021.100167.

[13] A. D. Jones. "Design and additive manufacturing of TPMS-like cellular structures" • . PhD thesis. RMIT University, 2022. url: https://researchrepository.rmit.edu.au/esploro/outputs/9922159113301341.

[14] A. Karakoç. "RegionTPMS — Region based triply periodic minimal surfaces (TPMS) for 3-D printed multiphase bone scaffolds with exact porosity values" • . In: *SoftwareX* 16, 100835 (Dec. 2021). doi: https://doi.org/10.1016/j.softx.2021.100835 10.1016/j.softx.2021.100835.

[15] I. Maskery, L. A. Parry, D. Padrão, R. J. M. Hague, and I. A. Ashcroft. “FLatt Pack: A research-focussed lattice design program”•. In: *Additive Manufacturing* 49, 102510 (Jan. 2022). doi: https://doi.org/10.1016/j.addma.2021.102510 10.1016/j.addma.2021.102510.

[16] M. B. Lodi, A. Makridis, N. M. Carboni, K. Kazeli, N. Curreli, T. Samaras, M. Angelakeris, G. Mazzarella, and A. Fanti. “Design and Characterization of Magnetic Scaffolds for Bone Tumor Hyperthermia”•. In: *IEEE Access* 10 (Feb. 14, 2022), pp. 19768–19779. doi: https://doi.org/10.1109/access.2022.3151470 10.1109/access.2022.3151470.

[17] F. Perez-Boerema, M. Barzegari, and L. Geris. “A flexible and easy-to-use open-source tool for designing functionally graded 3D porous structures”•. In: *Virtual and Physical Prototyping* 17.3 (Mar. 2022), pp. 682–699. doi: https://doi.org/10.1080/17452759.2022.2048956 10.1080/17452759.2022.2048956.

[18] K. Marchais, Y. Chemisky, R. d’Esparbès, M. R. Guevara, Y. Legerstee, and sudeep5511. *3MAH/microgen: v1.3.2*. Version 1.3.2. Initial release 1.0.0 on 2022-07-04. Jan. 14, 2025. doi: https://doi.org/10.5281/ZENODO.14643858 10.5281/ZENODO.14643858. url: https://github.com/3MAH/microgen.

[19] J. Pèrez-Barrera, A. Gómez-Ortega, M. I. Tenorio-Suárez, K. Corrales-Camacho, S. Piedra, and C. Fèlix-Martínez. “Version [2.0] - [MaSMaker: An open-source, portable software to create and integrate maze-like surfaces into arbitrary geometries]”•. In: *SoftwareX* 26, 101683 (May 2024). doi: https://doi.org/10.1016/j.softx.2024.101683 10.1016/j.softx.2024.101683.

[20] M. I. Tenorio-Suárez, A. Gómez-Ortega, H. Canales, S. Piedra, and J. Pérez-Barrera. “MaSMaker: An open-source, portable software to create and integrate maze-like surfaces into arbitrary geometries”•. In: *SoftwareX* 19, 101203 (July 2022). doi: https://doi.org/10.1016/j.softx.2022.101203 10.1016/j.softx.2022.101203.

[21] A. Forès-Garriga, H. García de la Torre, R. Lado-Roigé, G. Gómez-Gras, and M. A. Pérez. *Triply Periodic Minimal Surfaces Generator - TPMSgen*. Version 1.0.0. Jan. 17, 2023. url: https://github.com/albertforesg/TPMSgen.

[22] C. Lu, L. A. Lesmana, F. Chen, and M. Aziz. “MD-TPMS: Multi-dimensional gradient minimal surface generator”•. In: *Software Impacts* 17, 100527 (Sept. 2023). doi: https://doi.org/10.1016/j.simpa.2023.100527 10.1016/j.simpa.2023.100527.

[23] J. Leung, V. Verwilligen, D. Clarke, S. Adebileje, S. Latifi, H. Liu, B. McCane, B. Reynolds, and G. Yun. *TPMS Studio*. Version 0.8.4. Biomolecular Interaction Centre, Nov. 24, 2023. url: https://tpmsstudio.com/.

[24] D. Lin, C. Zhang, X. Chen, N. Wang, and L. Yang. “TPMS_Scaffold_Generator: A Scaffold-Structure Generator Based on Triply Periodic Minimal Surfaces”•. In: *Additive Manufacturing Frontiers* 3.2, 200123 (June 2024). doi: https://doi.org/10.1016/j.amf.2024.200123 10.1016/j.amf.2024.200123.

[25] H. Chris-Amadin and O. Ibhadode. “LattGen: A TPMS lattice generation tool” • . In: *Software Impacts* 21, 100665 (Sept. 2024). doi: https://doi.org/10.1016/j.simpa.2024.100665 10.1016/j.simpa.2024.100665.

[26] C. Bálint, G. Valasek, and L. Gergó. “Operations on Signed Distance Functions” • . In: *Acta Cybernetica* 24.1 (May 21, 2019), pp. 17–28. doi: https://doi.org/10.14232/actacyb.24.1.2019.3 10.14232/actacyb.24.1.2019.3.

[27] R. Malladi, J. A. Sethian, and B. C. Vemuri. “Shape modeling with front propagation: a level set approach” • . In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.2 (Feb. 1995), pp. 158–175. doi: https://doi.org/10.1109/34.368173 10.1109/34.368173.

[28] T. Chan and W. 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)* (San Diego, CA, United States). Washington, DC, United States: IEEE Computer Society, June 2005, pp. 1164–1170. isbn: 978-0-7695-2372-9. doi: https://doi.org/10.1109/cvpr.2005.212 10.1109/cvpr.2005.212.

[29] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. “Function representation in geometric modeling: concepts, implementation and applications” • . In: *The Visual Computer* 11.8 (Aug. 1995), pp. 429–446. doi: https://doi.org/10.1007/bf02464333 10.1007/bf02464333.

[30] I. Quilez. *distance functions*. url: https://iquilezles.org/articles/distfunctions/ (visited on 12/22/2023).

[31] J. A. Baerentzen and H. Aanaes. *Generating Signed Distance Fields From Triangle Meshes*. IMM Technical Report IMM-TR-2002-21. Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby: Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2002. url: http://www2.imm.dtu.dk/pubdb/pubs/1289-full.html.

[32] C. Bálint, G. Valasek, and L. Gergó. “Operations on Signed Distance Function Estimates” • . In: *Computer-Aided Design and Applications* 20.6 (Mar. 2023), pp. 1154–1174. doi: https://doi.org/10.14733/cadaps.2023.1154-1174 10.14733/cadaps.2023.1154-1174.

[33] I. Quilez. *interior SDFs*. 2020. url: https://iquilezles.org/articles/interiordistance/ (visited on 12/22/2023).

[34] L. Custodio, T. Etiene, S. Pesco, and C. Silva. “Practical considerations on Marching Cubes 33 topological correctness” • . In: *Computers & Graphics* 37.7 (Nov. 2013), pp. 840–850. doi: https://doi.org/10.1016/j.cag.2013.04.004 10.1016/j.cag.2013.04.004. url: http://www.sci.utah.edu/ etiene/pdf/mc33.pdf.

[35] L. Custodio, S. Pesco, and C. Silva. “An extended triangulation to the Marching Cubes 33 algorithm” • . In: *Journal of the Brazilian Computer Society* 25.6 (June 2019). doi:

https://doi.org/10.1186/s13173-019-0086-6 10.1186/s13173-019-0086-6.

[36] R. Grosso. “Construction of Topologically Correct and Manifold Isosurfaces” • . In: *Computer Graphics Forum* 35.5 (Aug. 2016), pp. 187–196. doi: https://doi.org/10.1111/cgf.12975 10.1111/cgf.12975.

[37] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. “Efficient Implementation of Marching Cubes’ Cases with Topological Guarantees” • . In: *Journal of Graphics Tools* 8.2 (Jan. 2003), pp. 1–15. doi: https://doi.org/10.1080/10867651.2003.10487582 10.1080/10867651.2003.10487582. url: https://www.ks.uiuc.edu/Research/vmd/projects/ece498/surf/lewiner.pdf.

[38] W. E. Lorensen and H. E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm” • . In: *SIGGRAPH ‘87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (Anaheim, CA, United States). Ed. by M. C. Stone. New York, NY, United States: Association for Computing Machinery, Aug. 1987, pp. 163–169. isbn: 978-0-89791-227-3. doi: https://doi.org/10.1145/37401.37422 10.1145/37401.37422.

[39] G. M. Nielson. “Dual Marching Cubes” • . In: *VIS ‘04: Proceedings of the conference on Visualization ‘04* (Austin, TX, United States). Washington, DC, United States: IEEE Computer Society, Oct. 2004, pp. 489–496. isbn: 978-0-7803-8788-1. doi: https://doi.org/10.1109/visual.2004.28 10.1109/visual.2004.28.

[40] J. Manson and S. Schaefer. “Isosurfaces Over Simplicial Partitions of Multiresolution Grids” • . In: *Computer Graphics Forum* 29.2 (May 2010), pp. 377–385. doi: https://doi.org/10.1111/j.1467-8659.2009.01607.x 10.1111/j.1467-8659.2009.01607.x. url: https://people.engr.tamu.edu/schaefer/research/iso_simplicial.pdf.

[41] N. Markuš. *A fast algorithm for generating triangle meshes from signed distance bounds*. Apr. 5, 2020. url: https://nenadmarkus.com/p/fast-algo-sdb-to-mesh/ (visited on 03/20/2025).

[42] F. Labelle and J. R. Shewchuk. “Isosurface stuffing: fast tetrahedral meshes with good dihedral angles” • . In: *ACM Transactions on Graphics* 26.3 (July 29, 2007), p. 57. doi: https://doi.org/10.1145/1276377.1276448 10.1145/1276377.1276448.

[43] D. Bhate. “Four Questions in Cellular Material Design” • . In: *Materials* 12.7, 1060 (Mar. 2019). doi: https://doi.org/10.3390/ma12071060 10.3390/ma12071060.

[44] C. Pan, Y. Han, and J. Lu. “Design and Optimization of Lattice Structures: A Review” • . In: *Applied Sciences* 10.18, 6374 (Sept. 2020). doi: https://doi.org/10.3390/app10186374 10.3390/app10186374.

[45] W. Tao and M. C. Leu. “Design of lattice structure for additive manufacturing” • . In: *2016 International Symposium on Flexible Automation (ISFA)* (Cleveland, OH, United

States). IEEE, Aug. 2016, pp. 325–332. isbn: 978-1-5090-3468-0. doi: https://doi.org/10.1109/isfa.2016.7790182 10.1109/isfa.2016.7790182.

[46] F. W. Zok, R. M. Latture, and M. R. Begley. "Periodic truss structures"•. In: *Journal of the Mechanics and Physics of Solids* 96 (Nov. 2016), pp. 184–203. doi: https://doi.org/10.1016/j.jmps.2016.07.007 10.1016/j.jmps.2016.07.007.

[47] *lattice*. In: *The American Heritage Dictionary of the English Language*. Ed. by Editors of the American Heritage Dictionaries. 5th ed. New York City, NY, United States: HarperCollins Publishers, Oct. 2018. isbn: 978-1-328-84169-8. url: https://ahdictionary.com/word/search.html?q=lattice (visited on 07/10/2023).

[48] D. M. Anderson, H. T. Davis, L. E. Scriven, and J. C. C. Nitsche. "Periodic Surfaces of Prescribed Mean Curvature"•. In: *Advances in Chemical Physics* 77 (Jan. 1, 1990), pp. 337–396. doi: https://doi.org/10.1002/9780470141267.ch6 10.1002/9780470141267.ch6.

[49] P. J. F. Gandy, S. Bardhan, A. L. Mackay, and J. Klinowski. "Nodal surface approximations to the P, G, D and I-WP triply periodic minimal surfaces"•. In: *Chemical Physics Letters* 336.3–4 (Mar. 2001), pp. 187–195. doi: https://doi.org/10.1016/s0009-2614(00)01418-4 10.1016/s0009-2614(00)01418-4.

[50] H. G. von Schnering and R. Nesper. "Nodal surfaces of Fourier series: Fundamental invariants of structured matter"•. In: *Zeitschrift für Physik B Condensed Matter* 83.3 (Oct. 1991), pp. 407–412. doi: https://doi.org/10.1007/bf01313411 10.1007/bf01313411.

[51] M. Wohlgemuth, N. Yufa, J. Hoffman, and E. L. Thomas. "Triply Periodic Bicontinuous Cubic Microdomain Morphologies by Symmetries"•. In: *Macromolecules* 34.17 (July 2001), pp. 6083–6089. doi: https://doi.org/10.1021/ma0019499 10.1021/ma0019499.

[52] J. W. Fisher, S. W. Miller, J. Bartolai, T. W. Simpson, and M. A. Yukish. "Catalog of triply periodic minimal surfaces, equation-based lattice structures, and their homogenized property data"•. In: *Data in Brief* 49, 109311 (Aug. 2023). doi: https://doi.org/10.1016/j.dib.2023.109311 10.1016/j.dib.2023.109311.

[53] J. A. Fernández-Fernández. *TriangleMeshDistance*. 2021. url: https://github.com/InteractiveComputerGraphics/TriangleMeshDistance (visited on 12/26/2023).

[54] SPDX Workgroup. *MIT License*. url: https://spdx.org/licenses/MIT.html (visited on 12/26/2023).

[55] T. Steer. *Baluster (Pembroke)*. CC0-1.0 license. url: https://www.thebasemesh.com/asset/baluster-(pembroke) (visited on 04/16/2025).

[56] T. Steer. *Acorn*. CC0-1.0 license. url: https://www.thebasemesh.com/asset/acorn (visited on 04/16/2025).

[57] D. Peter. *hyperfine*. Mar. 21, 2023. url: https://github.com/sharkdp/hyperfine.

[58] C. Audet, J. E. Dennis Jr., and S. Le Digabel. “Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm”•. In: *SIAM Journal on Optimization* 19.3 (Jan. 2008), pp. 1150–1170. doi: https://doi.org/10.1137/070707518 10.1137/070707518.

[59] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. 1st ed. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Dec. 13, 2017. isbn: 978-3-319-68912-8. doi: https://doi.org/10.1007/978-3-319-68913-5 10.1007/978-3-319-68913-5.

[60] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel. “Two decades of blackbox optimization applications”•. In: *EURO Journal on Computational Optimization* 9, 100011 (2021). doi: https://doi.org/10.1016/j.ejco.2021.100011 10.1016/j.ejco.2021.100011.

[61] C. Geuzaine and J.-F. Remacle. “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”•. In: *International Journal for Numerical Methods in Engineering* 79.11 (May 7, 2009), pp. 1309–1331. doi: https://doi.org/10.1002/nme.2579 10.1002/nme.2579.

[62] R. Anderson et al. “MFEM: A modular finite element methods library”•. In: *Computers & Mathematics with Applications* 81 (Jan. 1, 2021), pp. 42–74. doi: https://doi.org/10.1016/j.camwa.2020.06.009 10.1016/j.camwa.2020.06.009.

[63] C. Audet, S. Le Digabel, V. R. Montplaisir, and C. Tribes. “Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm”•. In: *ACM Transactions on Mathematical Software* 48.3 (Sept. 10, 2022), pp. 1–22. doi: https://doi.org/10.1145/3544489 10.1145/3544489.

[64] C. Audet, S. Le Digabel, and C. Tribes. “The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables”•. In: *SIAM Journal on Optimization* 29.2 (Jan. 2019), pp. 1164–1189. doi: https://doi.org/10.1137/18m1175872 10.1137/18m1175872.

[65] P. A. Patience. *Results of paper “Cesogen: Cellular solid generator”*•. Version 0.1-rc1. Zenodo, Apr. 23, 2025. doi: https://doi.org/10.5281/zenodo.15270544 10.5281/zenodo.15270544.

List of Figures

1 Tree representation of the operations Cesogen performs when invoked as ‘cesogen tpms.gyroid file mesh1.ply intersection tpms.p file mesh2.ply 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……31

2 Showcase of various TPMS-like endoskeletons generated by the command ‘cesogen -s ℓ Ω offset ξ ‘ with by default $\ell = 60^3$ and $\xi = 0.35$. ….32

3 Showcase of various TPMS-like shells generated by the command ‘cesogen -s ℓ Ω shell Ξ ‘ with $\Xi = 2\xi$ and by default $\ell = 60^3$ and $\xi = 0.35$. ……33

4 Neovius baluster, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh thebasemesh-balust-pembr and (b, c) filled meshes generated by the command ‘cesogen -s ℓ tpms.neovius scale κ_0 file baluster.obj intersection’, which took 1.3 and 4.2 to generate, respectively. … ……34

5 OCTO acorn, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh [56] and (b, c) filled meshes generated by the command ‘cesogen -s ℓ tpms.octo scale κ_0 file acorn.obj intersection’, which took 1.3 and 4.1 to generate, respectively. . ……35

6 Hierarchical cellular solid consisting of the intersection of (a) a truncated ball, (b) a P endoskeleton, (c) a finer D endoskeleton, and (d) an even finer gyroid endoskeleton. The figures at the left show the original models and those at the right the progressively inter-sected cellular solid. The command which generates the final re-sult is ‘`cesogen -b -4,-4,-4,4,4,4 -s 130^3 ball scale 5tpms.p scale 4 tpms.d scale 3.75 tpms.gyroid scale 1.25 intersection intersection intersection`’. ……36

7 Direct hierarchical L^2 point errors (●), L^2 cell errors (▲), max point distances (○) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.……37

8 Colored meshes corresponding to $\Omega_B \cap \Omega_P \cap \Omega_D$. The log-scale coloring corresponds to the distance from the points to the reference SDF ….38

9 Roundtrip L^2 point errors (□), L^2 cell errors (▲), max point distances(□) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes. ……39

10 Meshes before the first and 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 -b -4,-4,-4,4,4,4 -s 100^3 ball scale 4 tpms.gyroid scale 4 intersection’.....40

11 Mean time taken for Cesogen to generate gyroids via ‘cesogen $-s\ \ell$ tpms.gyroid’ with $\ell \in \{253,503,1003,2003,3003, ...,8003\}$. Sam-ples beyond 800^3 required more memory than that available on the test machine.....41

12 Block diagram of the optimization process. The generator consists of Cesogen and Gmsh [61], the FEM solver is MFEM [62], and the optimizer is NOMAD [63]. The optimizer starts by feeding a point $x \in X$ to the blackbox, where X is the domain of the objective function $f : X \to \bar{\mathbb{R}}$ and constraints $c : X \to \bar{\mathbb{R}}^m$, with $\bar{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$. From this point, and further informed by the configuration C, the blackbox produces the values $f(x)$ and $c(x)$ which are inspected by the optimizer to propose the following point, and the cycle repeats until a stopping criterion is met. . 42

13 Initial hourglass (a) and best solutions (b, c, d) to the optimization problem for each cellular solid kind. In the convergence plot (e), the filled shapes represent feasible solutions, the solid lines the best solution found so far, and infeasible solutions are omitted. 43

List of Tables

1 Various standalone, freely available cellular solid generators and their implementation languages, whether they offer batch-like CLI, GUI or library form, their features, year of publication of supporting article, or if none, initial release, and finally reference. Languages are one or more of C++ (C), Common Lisp (L), MATLAB (M), Python (P), Mathematica (W) and unknown (U). Having a CLI presupposes it is compatible with batch processing, i.e., requiring no user input. Generators with neither CLI, GUI or library form require editing the source code before running. Features may be one or more of extensive library of cellular solids (E), filling arbitrary solids (F), generating graded (G), heterogeneous (H), hierarchical (I) and multisymmetrical (S) cellular solids, applying mathematical transformations (T) and having advanced modeling features (M). We determined the features from a cursory inspection of the papers presented, the documentation and, when those were insufficient or incomplete and the source code was available, the source code. TPMS Studio in particular may have more features than those listed. 45

2 Expressions of TPMS-like surfaces described by Fisher et al. [52]. The right-hand sides of the corresponding implicit surface equations are 0. 46

3 Solutions to the optimization problem for each cellular solid kind, along with the time t taken for each. 47

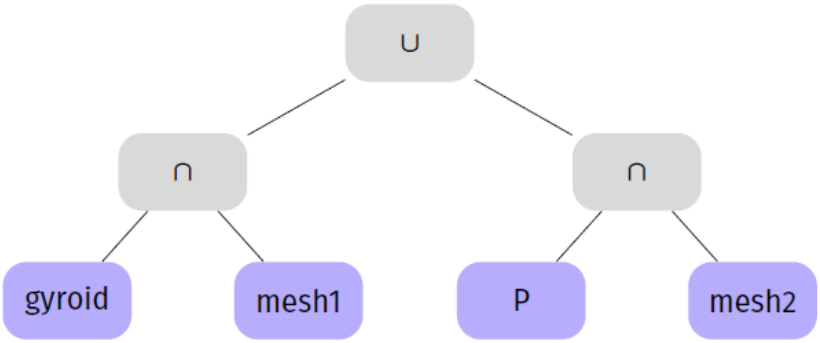
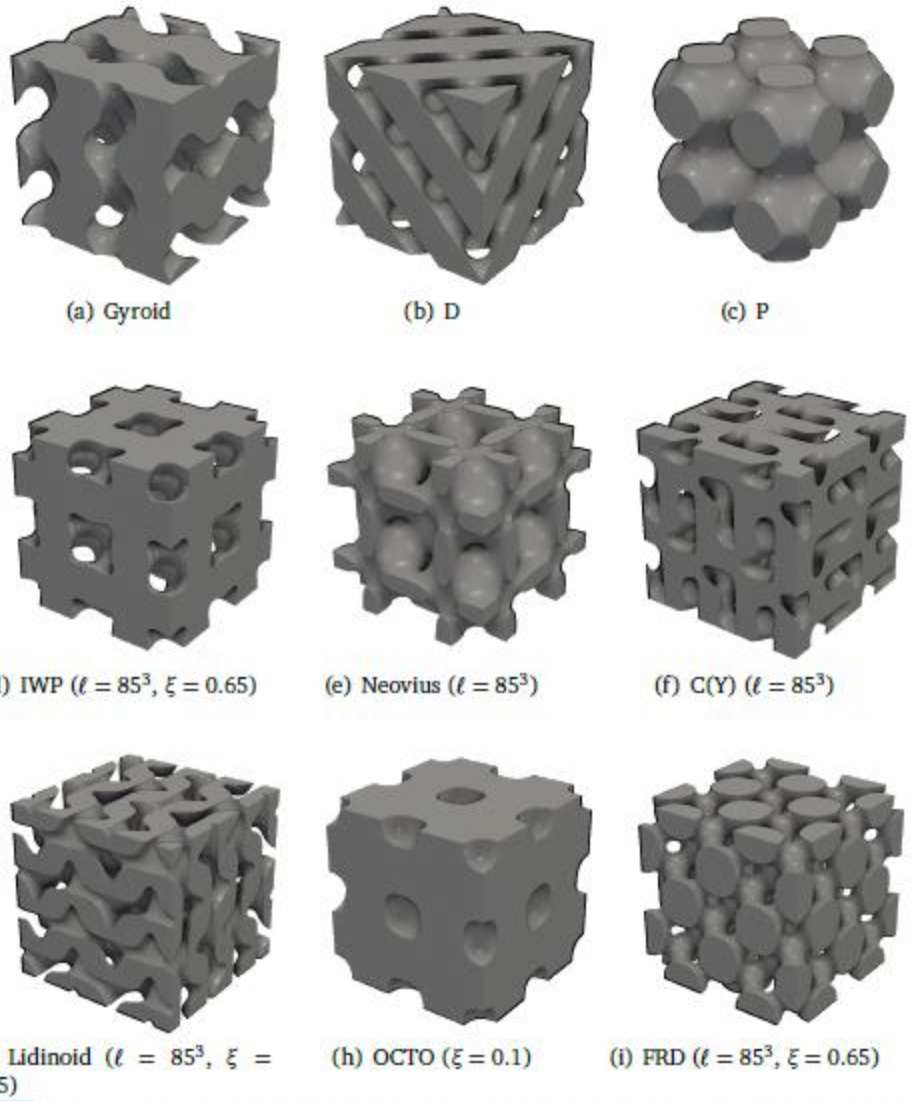


Figure 1: Tree representation of the operations Cesogen performs when invoked as '`cesogen tpms.gyroid file mesh1.ply intersection tpms.p file mesh2.ply intersection union`'. It evaluates the SDFs in a depth-first, post-order manner. In fact, the command-line arguments are a flattened version of the tree with the nodes visited in depth-first post-order.

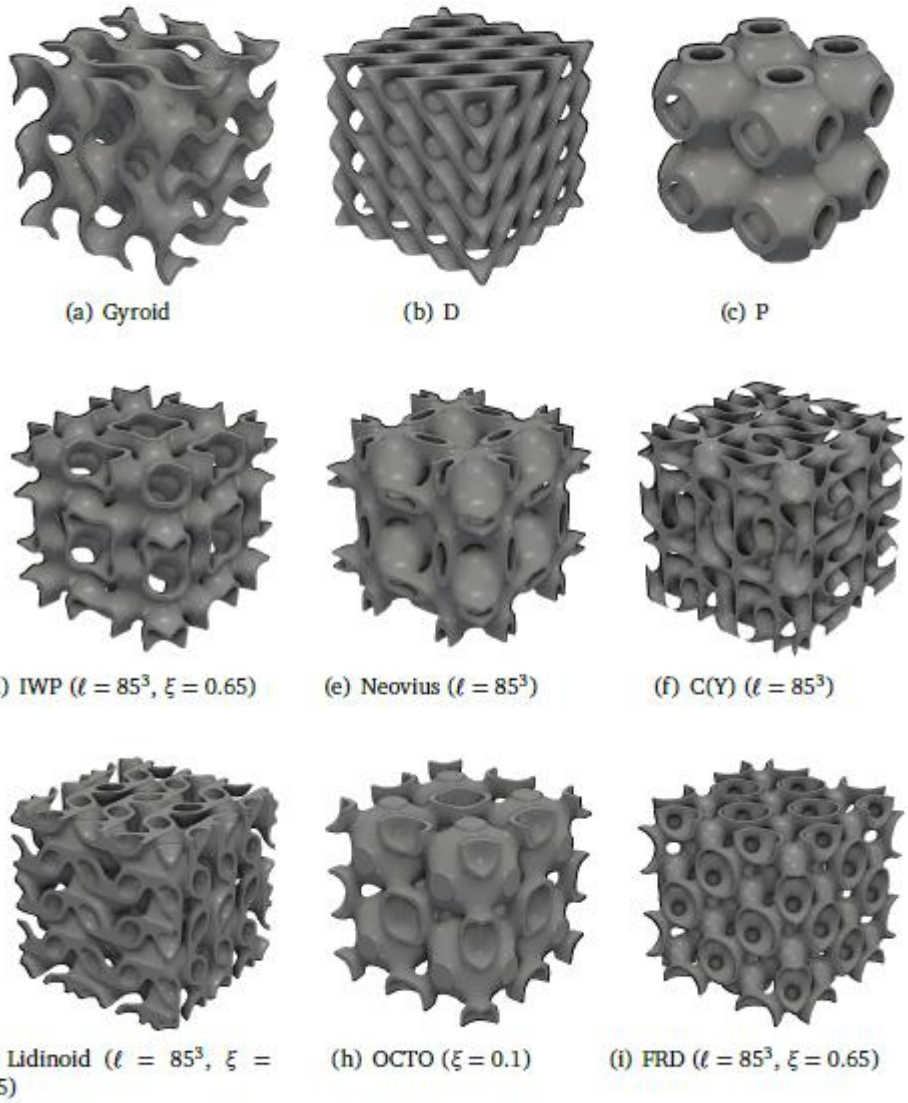


(a) Gyroid (b) D (c) P

(d) IWP ($\ell = 85^3$, $\xi = 0.65$) (e) Neovius ($\ell = 85^3$) (f) C(Y) ($\ell = 85^3$)

(g) Lidinoid ($\ell = 85^3$, $\xi = 0.65$) (h) OCTO ($\xi = 0.1$) (i) FRD ($\ell = 85^3$, $\xi = 0.65$)

Figure 2: Showcase of various TPMS-like endoskeletons generated by the command '`cesogen -s` ℓ Ω `offset` ξ' with by default $\ell = 60^3$ and $\xi = 0.35$.



(a) Gyroid (b) D (c) P

(d) IWP ($\ell = 85^3$, $\xi = 0.65$) (e) Neovius ($\ell = 85^3$) (f) C(Y) ($\ell = 85^3$)

(g) Lidinoid ($\ell = 85^3$, $\xi = 0.65$) (h) OCTO ($\xi = 0.1$) (i) FRD ($\ell = 85^3$, $\xi = 0.65$)

Figure 3: Showcase of various TPMS-like shells generated by the command '`cesogen -s` ℓ Ω `shell` Ξ' with $\Xi = 2\xi$ and by default $\ell = 60^3$ and $\xi = 0.35$.

(a) Original (b) Filled, $(\ell, \kappa_0) = (80^3, 0.065)$ (c) Filled, $(\ell, \kappa_0) = (120^3, 0.03)$

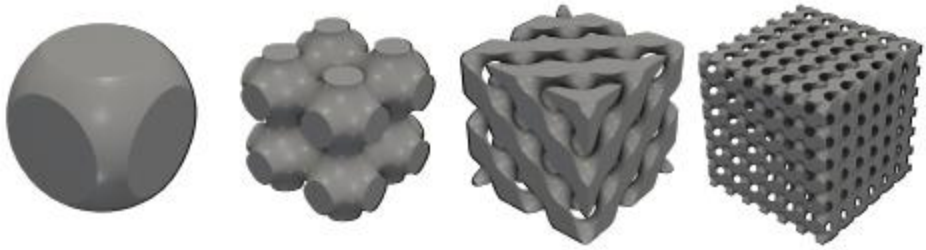
Figure 4: Neovius baluster, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh [55] and (b, c) filled meshes generated by the command '`cesogen -s` ℓ `tpms.neovius scale` κ_0 `file baluster.obj intersection`', which took 1.3 s and 4.2 s to generate, respectively.

(a) Original

(b) Filled, $(\ell, \kappa_0) = (80^3, 0.009)$

(c) Filled, $(\ell, \kappa_0) = (120^3, 0.003)$

Figure 5: OCTO acorn, (a) original triangle mesh converted from the quadrilateral mesh from TheBaseMesh [56] and (b, c) filled meshes generated by the command '`cesogen -s` ℓ `tpms.octo scale` κ_0 `file acorn.obj intersection`', which took 1.3 s and 4.1 s to generate, respectively.



(a) Truncated ball Ω_B (b) P endoskeleton Ω_P (c) D endoskeleton Ω_D (d) Gyroid endoskeleton Ω_G

(e) $\Omega_B \cap \Omega_P$ (f) $\Omega_B \cap \Omega_P \cap \Omega_D$ (g) $\Omega_B \cap \Omega_P \cap \Omega_D \cap \Omega_G$

Figure 6: Hierarchical cellular solid consisting of the intersection of (a) a truncated ball, (b) a P endoskeleton, (c) a finer D endoskeleton, and (d) an even finer gyroid endoskeleton. 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 -b -4,-4,-4,4,4,4 -s 130^3 ball scale 5 tpms.p scale 4 tpms.d scale 3.75 tpms.gyroid scale 1.25 intersection intersection intersection`'.

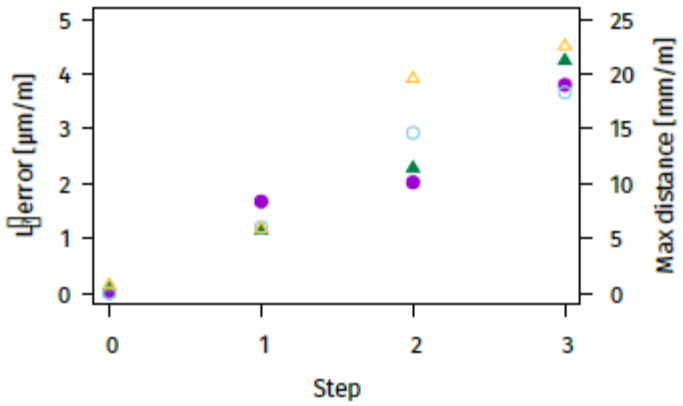
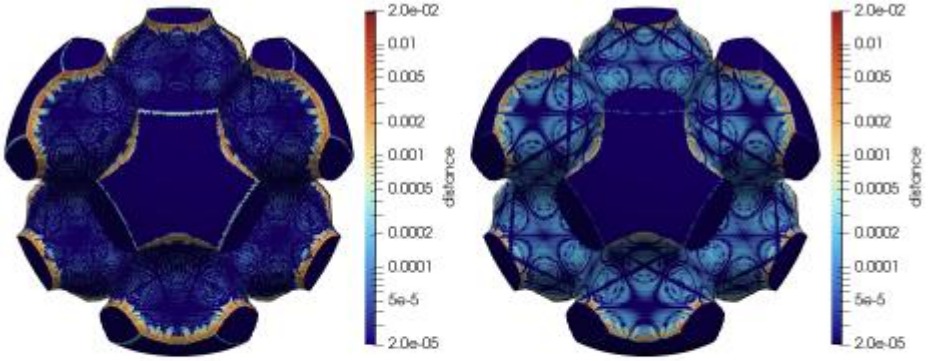


Figure 7: Direct hierarchical L^2 point errors (●), L^2 cell errors (▲), max point distances (○) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.



(a) Direct intersection (b) Sequential intersection

Figure 8: Colored meshes corresponding to $\Omega_B \cap \Omega_P \cap \Omega_D$. The log-scale coloring corresponds to the distance from the points to the reference SDF.

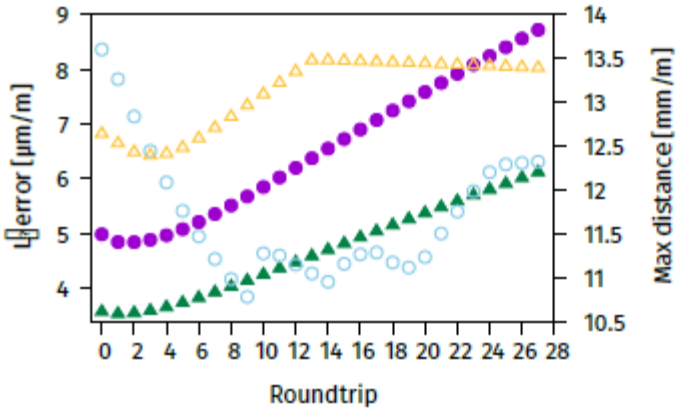
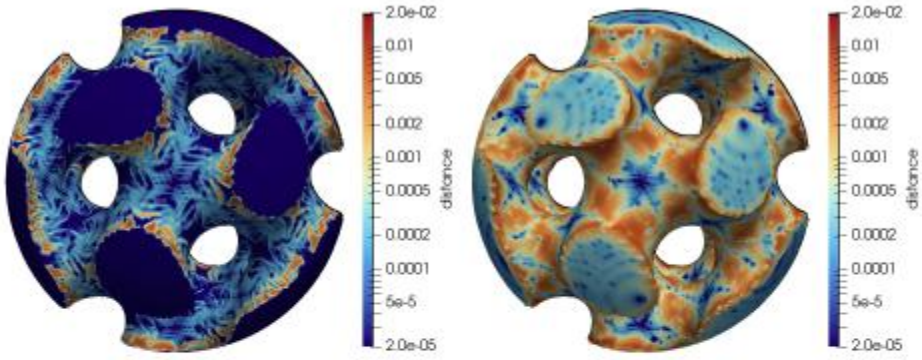


Figure 9: Roundtrip L^2 point errors (●), L^2 cell errors (▲), max point distances (○) and max cell distances (△). Points are represented by circles, cells by triangles. L^2 errors are represented by filled shapes, max distances by hollow shapes.



(a) Before first roundtrip (b) After last roundtrip

Figure 10: Meshes before the first and 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 -b -4,-4,-4,4,4,4 -s 100^3 ball scale 4 tpms.gyroid scale 4 intersection`'.

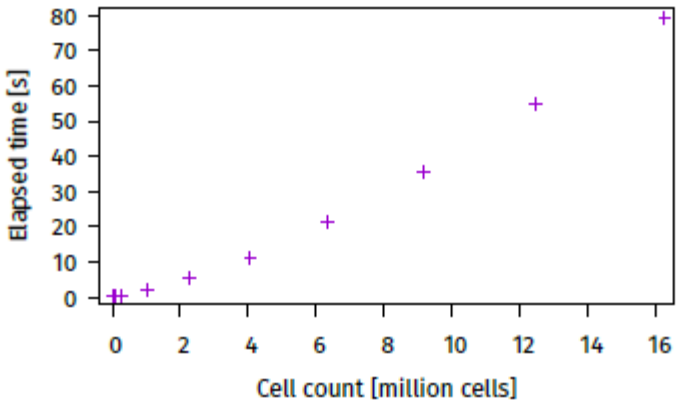


Figure 11: Mean time taken for Cesogen to generate gyroids via '`cesogen -s` ℓ `tpms.gyroid`' with $\ell \in \{25^3, 50^3, 100^3, 200^3, 300^3, \ldots, 800^3\}$. Samples beyond 800^3 required more memory than that available on the test machine.

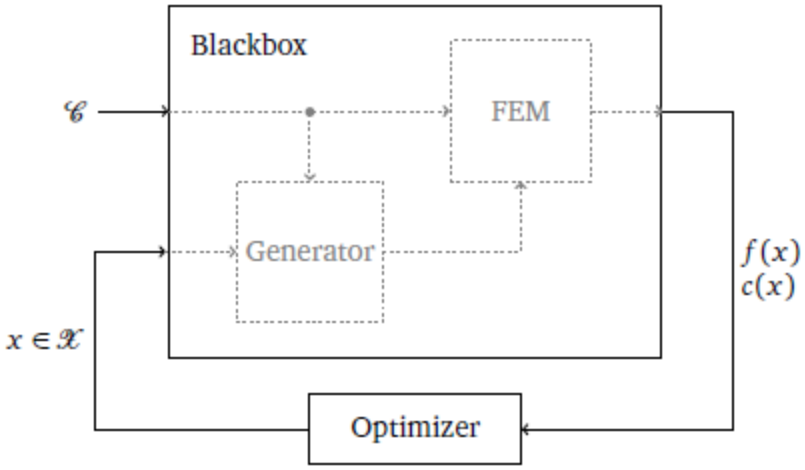
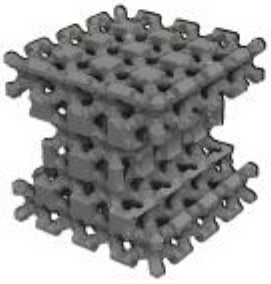


Figure 12: Block diagram of the optimization process. The generator consists of Cesogen and Gmsh [61], the FEM solver is MFEM [62], and the optimizer is NOMAD [63]. The optimizer starts by feeding a point $x \in \mathscr{X}$ to the blackbox, where $\mathscr{X}$ is the domain of the objective function $f : \mathscr{X} \to \overline{\mathbb{R}}$ and constraints $c : \mathscr{X} \to \overline{\mathbb{R}}^m$, with $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$. From this point, and further informed by the configuration $\mathscr{C}$, the blackbox produces the values $f(x)$ and $c(x)$ which are inspected by the optimizer to propose the following point, and the cycle repeats until a stopping criterion is met.

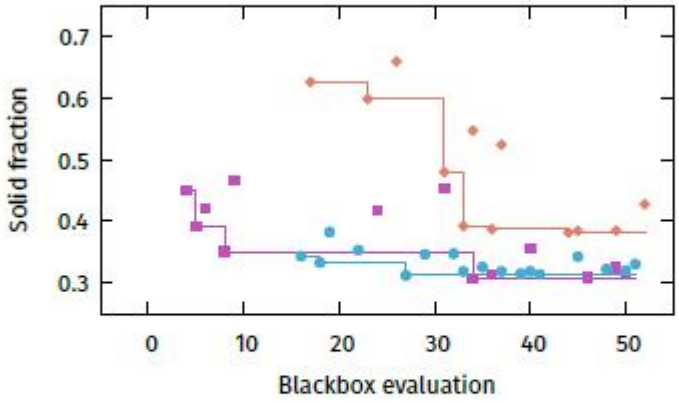
(a) Hourglass

(b) Gyroid ◆



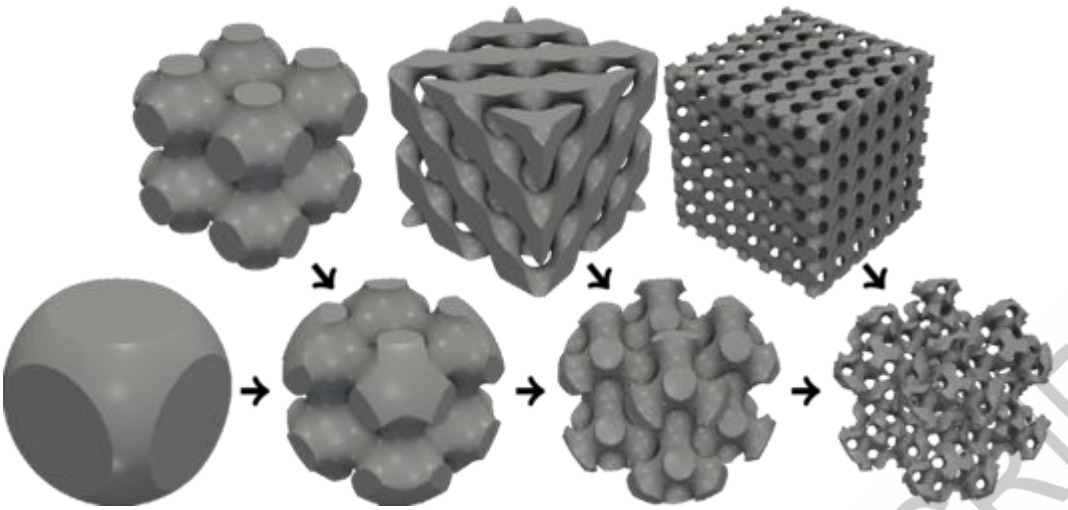
(c) IWP ■

(d) FRD ●



(e) Convergence plot

Figure 13: Initial hourglass (a) and best solutions (b, c, d) to the optimization problem for each cellular solid kind. In the convergence plot (e), the filled shapes represent feasible solutions, the solid lines the best solution found so far, and infeasible solutions are omitted.



Graphical abstract