





Visual programming with invariant, pre- and post-conditions for approximation of a 3D model with assorted 1.5D and 2.5D lattice families

C. Builes-Roldan¹  and J. Lalinde-Pulido²  and C. Echeverri-Cartagena³  and O. Ruiz-Salguero^{†1} 

¹ Laboratory of CAD CAM CAE. U. EAFIT, Colombia

² High Performance Computing Facility APOLO. U. EAFIT, Colombia

³ Machine Tool Laboratory. U.EAFIT, Colombia

Abstract

End User Programming (EUP) endows a non-programmer user with software tools (normally in icon form in Visual Programming) to execute tasks in the domain of the user, by extending the capabilities of a given host software. This manuscript presents the results of (a) lattice family creation and (b) 3D region population with lattice families, programmed by a non-programmer Product Designer in a VP environment (i.e. Grasshopper). This non-programmer user is endowed with intuitive discussions on Pre- and Post-conditions and Invariants from Contract-based Programming (CBP). The lattice individuals are built from [EDGE + area] and [FACE + thickness] truss or frame structures. In spite of CBP being tailored for imperative programming and Grasshopper being a flow-based system, the formal structure of Pre- and Post-conditions and Invariants of CBP provides a successful frame for program correctness. Future work is required in non-affine deformations of the lattice individuals before docking them in the sought 3D region.

(see <https://www.acm.org/publications/class-2012>)

CCS Concepts

• **Computing methodologies** → *Volumetric models*; • **Software and its engineering** → *Visual languages*; *Correctness*; • **Theory of computation** → *Preconditioning*; *Computational geometry*; • **Social and professional topics** → *Computational thinking*;

1. Introduction

1.1. Research Target

This manuscript presents a development executed on Visual Programming tools (i.e. Grasshopper) which (a) defines lattice individuals based on the limbs of types 1.5D [EDGE + area] and 2.5D [FACE + thickness] due to their lower computing consumption, (b) executes the approximation of a BODY (i.e. 3D region) enclosed by a B-Rep M by an enumeration of the given lattice individuals, tuned by using the usual thresholds in Exhaustive Enumerations. This development is executed by a Designer who does not have previous training in programming but is equipped with intuitive knowledge of the Pre-condition, Invariant and post-condition formalisms for Contract Based Programming [Gri81, WGC16].

1.2. Context. Programing for Non-programmers

Visual Programming is built as a tool for practitioners and experts in topics other than from the programing aspect. This tool which would allow them to build / assemble applications based on icons of pre-defined pre-compiled functions. Visual Programing and in particular GrasshopperTM is a good starting tool for non-programmers to create scenario-driven, generative designs. However, Grasshopper does not offer: (1) *parametric* design, because its kernel (RhinocerosTM) does not have it, (2) iterative loops (`for`, `while`, `repeat`).

Section 2 reviews the state of art of Topologic and Geometric Modeling for lattices. Due to the special subject of this manuscript (Visual Programing for Lattice Modeling and Object approximation), Section 3 discusses the Methodology used and simultaneously presents the results of the modeling. Section 4 concludes the manuscript and indicates aspects for future work.

[†] corresponding author

2. Literature Review

2.1. Lattice Families and Properties

Ref. [PAHA18] evaluates 5 strategies (solid, intersected, graded, scaled, uniform) to translate Solid Isotropic Material Penalisation (SIMP) material densities for a cantilever load case into lattice parameters. Software used include Magics (Materialise Magics. Materialise N.V., Leuven, Belgium, 2014.), Autofab (Autofab Software, Marcam Engineering, 2011), MSC Nastran, Grasshopper - Rhino (no library specified), and others. No test is given to measure the discrepancy, resiliency, processing effort, and in-process support requirements.

Ref. [ABCP*17] addresses the tasks for generating the VoXel Representation by a lattice individual set that approximates the interior of a 3D body and joins the lattice set with a thick version of the skin. The lattices are approximated by much smaller VoXels. The number of voxels exponentially increases with the body size. The programming part was executed in MATLAB. No Finite Element computation is presented in this manuscript.

Ref. [GKR*19] presents a Programed Lattice Editor (PLE), based on a set of sequence parameters applied on one lattice type. Its strongest point is the application of affine and non-affine transformations of the basic rod-based individual to build a frame with torsion and gradients. PLE produces the set of Constructive Solid Geometry (CSG) instructions to build the full lattice domain. Therefore, no geometric modeling is actually executed in PLE. Neither are mechanics computations (FEA). PLE is embedded in Mithril, an environment developed in C/C++ for rapid prototyping. PLE is developed in CPython, C and C++.

IntraLattice [KTZ15] is a software running on Grasshopper, whose functionality is the generation of graded lattice sets filling up a solid region $\Omega \subset \mathbb{R}^3$, with and without $\partial\Omega$, the boundary or skin of Ω . *IntraLattice* contains modules: (a) cell generation, (b) affine transformations, (c) B-Rep generation (in mesh format) (d) post-processing for Additive Manufacturing.

2.2. Text vs. Visual Programming. Grasshopper

Ref. [JC11] compares 3 systems for CAD Visual Computing. This Ref. concludes that *Node-based* Visual programming (e.g. Houdini) presents these advantages: (a)- combines iteration and encapsulation, (b)- supports both forward- and reverse- order modeling methods, (c)- has implicit iterative process, and, (d)- allows to define more complex processes. On the other hand, *List-based* Visual Programming (e.g. Grasshopper - Rhino, Generative Components CG-Microstation) presents higher difficulty, specially originated in : (1)- encapsulation not available or available by reversing the modeling flow. (2)- non-available iterations.

Ref. [CV12] compares script vs. visual programming for Computer Aided Architecture (generative) Design. Scripts and Visual Programs are interpreted in the CAAD (as opposed to *compiled*). Visual Programming (Grasshopper) has a gentler learning curve, while Scripting (Visual Basic AutoCAD -VBA) is more convenient for sophisticated algorithms. Both tested groups, however, encountered problems with more ambitious projects.

Ref. [ATC15] discusses End-user Programming (EUP) in which

```

1
2 {Pre: <initial program variables status>}
3 WHILE <conditions for execution>
4 {Inv: <invariant status of variables during Loop>}
5 ...
6 ...
7 ENDWHILE
8 {Post: <final program variables status>}
9

```

Figure 1: Pre-Condition, Post-Condition and Invariants [Gri81].

non-programmer users create applications which expedite their work in particular domains (in this case, robotics). In Flow-based Programming (part of EUP): (a) box icons represent functions, (b) connectors represent data. (c) no flow control tokens (*while*, *for*, *repeat*) are available. (d) boxes are SIMO (Single Input Multi Output) functions, (f) semantics are located in the boxes and not in the connectors.

2.3. Pre-Condition, Post-Condition, Invariant. Contract-based Programming.

Refs. [WGC16,Gri81,FL11] address Contract-based Programming (CBP). CBP includes 1st-order logic predicates before, after or during a loop execution (Pre-, Post- and Invariant respectively, Fig. 1). Predicates *are not instructions*. They are statements about the memory status, which are *True* at the given lines (2,4,8 in Fig. 1). The Pre-condition (line 2) and Post-condition (line 8) describe the values of the *relevant* variables before and after the loop, respectively. The Invariant (line 4) depicts a typical un-finished state of variables when the loop is executing. One value of the invariant is that $Inv \wedge (\neg C) = Post$. Therefore, the instructions in the loop (lines 5,6) must work towards $(\neg C)$ while keeping *Inv* true. This last consideration dictates the instruction of the loop (lines 5,6).

Ref. [Seb19] addresses the absence of programming loops in Grasshopper and possible (cumbersome) repairs for this deficiency. In absence of loop instructions, the application of Pre-, Post-condition and Invariant would be apparently impossible. Yet, our project shows that the non-programmer Product Designer was able to enforce those logical predicates in Grasshopper and to obtain the correct voxel enumeration for the 3D region (i.e. solid).

2.4. Conclusions of the Literature Review

The state of the art for Lattice Design indicates: (1) Regarding computer environments, there are several reported. However, they require significant programming skills and professional code bases. (2) Regarding Geometric Modeling paradigm, lattice modeling postpones full Boundary Representation as much as possible, due to mechanics computation costs. Instead, Truss or Frame modeling is executed. (3) Regarding Programming paradigms, Visual Programming of Flow-based Programming appear as possibilities, in spite of requiring trained programmers to model the lattices.

In response to this status, we report here a development with this special set of circumstances: (a) the Product Designer with no programming experience, (b) Visual or Flow-Programming, (b) use of Pre-, Post-conditions and Invariants in spite of these being

planned for imperative script-based programming, (e) [EDGE + area] (1.5D) and [FACE + thickness] (2.5D) truss modeling formalism will be used for lattice modeling, postponing full B-Rep for the manufacturing stage.

3. Methodology and Results

3.1. Application of [EDGE + area] and [FACE + thickness] Limbs in Lattices

Fig. 2 displays how full 3D boundary Representations of a lattice are avoided in favor of 1.5D [EDGE + area] and 2.5D [FACE + thickness] limbs. Figs. 2(a) and 2(b) show lattice individuals whose medial axis are 1-dimensional and 2-dimensional, respectively ([MZCRS19]). Fig. 2(a) shows a 3D cross built by conic rods plus kinematic constraints. The EDGE is the medial axis $C(u)$ of the rod. The local rod cross section or area $\pi \cdot r^2(u)$ is dependent on the parameter u parameterizing the rod medial axis.

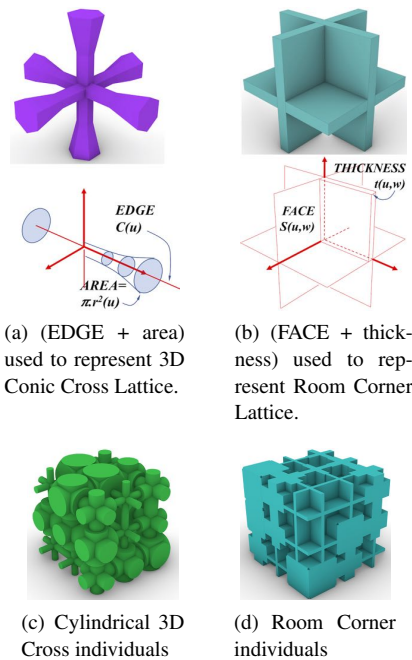
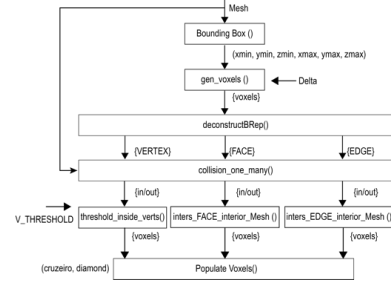
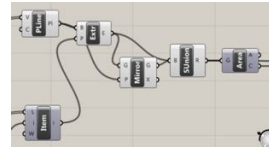


Figure 2: Lattice individuals by using [EDGE + area] and [FACE + thickness] limbs. Rectangular prismatic 3D region filled with non-uniform $N_x \times N_y \times N_z$ lattices.

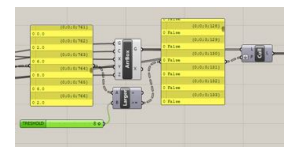
Fig. 2(b) shows lattices build with walls when the medial axis of the lattice individual is a 2-dimensional set. The Room Corner lattice individual is represented by 12 [FACE + thickness] elements plus kinematic constraints. Each element contains a FACE $S(u, w)$ and a thickness map $t(u, w)$ (Fig. 2(b)). Figs. 2(d) and 2(c) present prismatic domains with constant lattice topology but variable geometry.



(a) Workflow to determine Lattice occupancy in a 3D domain.



(b) Circuit for Reflection and FACE Boolean Union.



(c) Application of threshold to turn off lattices whose number of I VERTEXEs is below the threshold.

Figure 3: Example of Circuits for Lattice Construction and Enforcement of Threshold for Lattice Inclusion in 3D Region (3D Model).

3.2. 3D Region Lattice Occupancy

Fig. 3(a) displays the generic workflow for the construction of the Lattice enumeration that fills a given B 3D domain representing a solid, with boundary representation M (2-manifold mesh).

A given lattice is classified ([GHR05, XS93]), regarding its position in the body region M , as: **I**= inside M , **O**= outside M , **NIO**= neither inside or outside M .

Fig. 3 shows sample Grasshopper circuits for (a) building a lattice (in this case of the type [FACE + thickness], Fig. 3(b)) using operations such as reflections, boolean union, rotation, etc., (b) applying a Threshold value to decide whether the **NIO** lattices are graded as **I** (e.g. a lattice is considered **I** if more than Threshold=5 of its VERTEXEs are inside the solid region M , Fig. 3(c)) Results of this lattice enumeration process are displayed in Figs. 4 and 5.

4. Conclusions and Future Work

Visual Programming for a non-programmer Designer

The experiment of having a non-programmer Product Designer to program a lattice-based filling of a 3D region using a set of suitable lattice topologies with varying geometry had these circumstances: (a) no previous programming training, (b) independent learning (i.e. absence of programming tutors), (c) use of Visual Programming tools (i.e. Grasshopper), (d) informal seminars on Pre- and Post-conditions and Invariants ([Gri81]).

In spite of Pre- and Post-conditions and Invariants being devised for imperative languages, the Product Designer used Grasshopper,

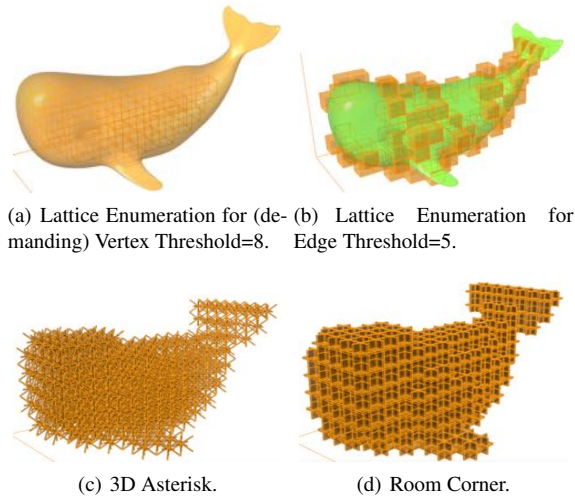


Figure 4: Whale data set. Lattice enumeration and assorted lattice individuals

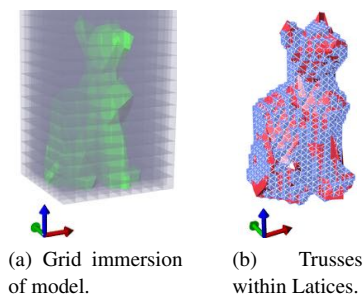


Figure 5: Cat data set. Results of visual Programming processing: immersion of model in grid and lattice approximation.

declarative or flow-based programming language, with high proficiency and technically correct results. It must be noticed however, that it was not intention of the present research to compete against professional lattice-based design tools.

Region Population with Lattice Individuals

This manuscript presents the geometric modeling of lattice-approximated 3D regions (i.e., objects) using lattices whose internal limbs are slender (i.e. thin rods and/or thin plates). For the purposes of computationally economic structural modeling, the rods are modeled as [EDGE + area] (a.k.a. 1.5D) elements and the plates are modeled as [FACE + thickness] (a.k.a. 2.5D) elements. These elements are normally supplemented with kinematic constraints at the computational mechanics stage.

Notice that a collection of lattices declared to approximate a 3D solid can be filled with lattice individuals of (a) uniform topology and diverse geometry (i.e. dimensions, thickness, area, etc.), or (b) diverse families.

Future work is required in the application of non-affine geometric transformations to the lattice individuals before their docking in the target 3D region. This extension would permit lattices whose

geometry contains straight-to-curve deformations, dictated by the functionality of the Additive Manufacturing.

References

- [ABCP*17] AREMU A., BRENNAN-CRADDOCK J., PANESAR A., ASHCROFT I., HAGUE R. J., WILDMAN R. D., TUCK C.: A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing. *Additive Manufacturing* 13 (2017), 1–13. 2
- [ATC15] ALEXANDROVA S., TATLOCK Z., CAKMAK M.: Roboflow: A flow-based visual programming language for mobile manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 5537–5544. doi:10.1109/ICRA.2015.7139973. 2
- [CV12] CELANI G., VAZ C. E. V.: CAD scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view. *International Journal of Architectural Computing* 10, 1 (2012), 121–137. doi:10.1260/1478-0771.10.1.121, ISSN 1478-0771. 2
- [FL11] FÄHNDRICH M., LOGOZZO F.: Static contract checking with abstract interpretation. In *Formal Verification of Object-Oriented Software* (Berlin, Heidelberg, 2011), Beckert B., Marché C., (Eds.), Springer Berlin Heidelberg, pp. 10–30. 2
- [GHR05] GARCIA M. J., HENAO M. A., RUIZ O. E.: Fixed grid finite element analysis for 3D structural problems. *International Journal of Computational Methods* 02, 04 (2005), 569–586. doi:10.1142/S0219876205000582. 3
- [GKR*19] GUPTA A., KURZEJA K., ROSSIGNAC J., ALLEN G., KUMAR P. S., MUSUVATHY S.: Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices. *Computer-Aided Design* 113 (2019), 35–47. 2
- [Gri81] GRIES D.: *The Science of Programming*. Springer-Verlag New York, 1981. ISBN 978-0-387-90641-6, eISBN 978-1-4612-5983-1. Chapter Developing Loops from Invariants and Bounds. 1, 2, 3
- [JC11] JANSSEN P., CHEN K.: Visual dataflow modelling: A comparison of three systems. In *Design Futures 2011 - Proceedings of the 14th International Conference on Computer Aided Architectural Design Futures* (Liege, Belgium, 01 2011), pp. 801–816. 2
- [KTZ15] KURTZ A., TANG Y., ZHAO F.: Intra lattice, 2015. Generative Lattice Design with Grasshopper. McGill’s Additive Design and Manufacturing Laboratory (ADML). URL: <http://intraLattice.com>. 2
- [MZCRS19] MONTOYA-ZAPATA D., CORTES C., RUIZ-SALGUERO O.: Fe-simulations with a simplified model for open-cell porous materials: A kelvin cell approach. *Journal of Computational Methods in Sciences and Engineering* (2019), 1–12. In Press. Published online: 27 May 2019. doi:10.3233/JCM-193669. 3
- [PAHA18] PANESAR A., ABDI M., HICKMAN D., ASHCROFT I.: Strategies for functionally graded lattice structures derived using topology optimisation for additive manufacturing. *Additive Manufacturing* 19 (2018), 81–94. doi:10.1016/j.addma.2017.11.008. 2
- [Seb19] SEBESTYEN A.: Loops in grasshopper. In *Bricks are Landing. Algorithmic Design of a Brick Pavilion*. T.U. Wien, 2019, pp. 15–24. ISBN 978-3-9504464-1-82. 2
- [WGC16] WANG B., GAO H., CHENG J.: Contract-based programming for future computing with ada 2012. In *2016 International Conference on Advanced Cloud and Big Data (CBD)* (2016), pp. 322–327. doi:10.1109/CBD.2016.062. 1, 2
- [XS93] XIE Y. M., STEVEN G.: A simple evolutionary procedure for structural optimization. *Computers and Structures* 49, 5 (1993), 885–896. 3

NON-OFFICIAL SUMMARY