A Graphics-Based Approach to Processing Voxel Models for Biopolymer 3D Printing

Algorithms From Digital Geometry

Cameron Nelson[1], Paul Nicholas[2], Mette Ramsgaard Thomsen[3]
[1,2,3]Royal Danish Academy
[1,2,3]{cnel | paul.nicholas | mette.thomsen}@kglakademi.dk

Voxel-based modeling is ideally suited to multi-material 3d printing, where many layers of volumetric fabrication data must be spatially registered. Beyond providing a generic format for volumetric data, voxels are also robust to topological changes that commonly hinder surface-based mesh approaches (Michalatos et al., 2016; Bernhard et al., 2018). Voxel algorithms for 3d printing are a growing area of research but remain limited in scope and application in comparison to 2d pixel algorithms from computer vision and graphics, despite that many of the latter have direct 3d analogues. The analogy with 2d pixels and suitability to GPU paradigms motivates the strong association between digital geometry and graphics. However, this burgeoning field carries untapped potential for architectural fabrication as well. We present a toolkit of raster algorithms with direct applications for 3d printing.

Keywords: *Volumetric Modeling, Multimaterial 3d Printing, Computer Graphics*

INTRODUCTION

As recent advances in GPU hardware and software have rapidly elevated the state of the art in massively multithreaded computation, parallelizations for common tasks from computational geometry have generated interest as efficient and robust alternatives to traditional, sequential algorithms. This paves the way for raster 3d geometry, where spatial data is represented by a discrete regular grid of voxels much as a 2d image may be composed of pixels; as well as general-purpose GPU (GPGPU) programming, possibly decoupled from any visual output. Our research employs volumetric models for multimaterial prints, locally tailoring a blend of custom biomaterials with varying structural performance and aesthetic properties. Voxel-based software solutions for general multimaterial printing have gradually gained popularity (Michalatos and Payne, 2016; Stamm et al., 2015). However, our custom material system carries unique challenges for toolpath planning, geometric constraint, and topology that warrant bespoke solutions (Rossi et al., 2023). To address these challenges of large-scale graded printing while leveraging the advantages of volumetric frameworks, we develop a toolkit of voxel algorithms drawing upon related concepts from 2d graphics to analyze and optimize geometry hand-in-hand with material and fabrication data.

This paper begins with an overview of 2d graphics operations with extensions to 3d,

followed by in-depth examples of several algorithms that leverage them as subroutines with applications to geometric problems in 3d printing. In particular, we describe heuristics for solid voxelization of point clouds, automatic analysis and amelioration of overhang and bridge conditions, and generation of closed, continuous print paths. We implement these algorithms using a combination of software, including Grasshopper via the C++ library OpenVDB, node-based graphics software Touchdesigner, Python, and custom GLSL shaders (Svilans et al., 2023). We make extensive use of parallelism and emphasize speed for realtime feedback wherever possible. In Discussion, we consider the results and performance of these algorithms and consider the context of physical biopolymer print prototypes. We expand upon existing voxel-based approaches to 3d printing, particularly in large-scale custom-material contexts, through novel applications of generalized graphics algorithms.

THEORY

Voxel representations are increasingly seen in multimaterial 3d printing due to the ease of extracting closed isocurves, their ability to specify material and other properties at arbitrary points in space, and their robustness to common operations from constructive geometry (Bernhard et al., 2018). As a preliminary, in this section we outline a selection of known algorithms for manipulating voxels in order to frame the contributions in the following section.

Conversion

Transfer of 3d curves, meshes, implicit distance fields, and point clouds to and from voxel representations is a key step for design tasks leveraging digital geometry. Bresenham and Digital Differential Analyzer algorithms are optimized methods for rasterizing lines and by extension triangles and other polygons (Håkansson, 2020). These enable realtime voxelization of polyline curves or mesh surfaces, which may in turn be solidified in realtime (Eisemann and Décoret, 2008; Schwarz and Seidel, 2010). However, many filling algorithms are slow or prone to error. Pointclouds can be naively voxelized by binning, i.e. rounding coordinates to the nearest grid point, but may contain numerous small errors or holes and often only capture the surface of an object. In Implementation we consider a more parallel and robust solid voxelization algorithm leveraging the generalized winding number. Figure 1 demonstrates this method on a triangle mesh.

Reverse conversions translate voxels into formats usable for fabrication or representation such as curves and meshes. Surface Nets, Dual Contouring, Flying Edges, and the Marching Cubes algorithm are popular choices, for which parallel implementations have been proposed (Schroeder et al., 2015). Curves can be extracted analogously, as 2d isocontours. Within reasonable constraints, these techniques guarantee desirable properties like watertightness. Signed distance fields can be approximated by interpolation techniques, or using a raster distance transform such as the Jump Flood algorithm. These methods are naturally suited to slicing voxel geometry for 3d printing, due to the ease of extracting closed, regularly spaced layer curves.

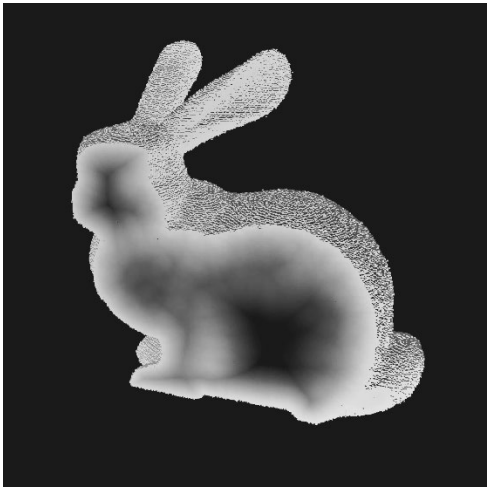

Figure 1
Cutaway of solid voxelized mesh, with grayscale indicating distance transform

Morphology

Mathematical morphology operations are a broad class of algorithms concerning the topology of an image. The atomic morphological operations are erosion and dilation: 3d morphology simply extends the original 4- and 8-connected definitions from 2d to the 6- and 26-connected cases in 3d. All are straightforward to implement in any shading language. Compositions of these operations form the (proper) opening and closing operations, as well as the auto-median. Applied successively, these can thicken and thin digital geometry, remove small noise, or detect edges. In 3d printing these are directly applicable for thickening walls of geometry, reinforcing corners and thin elements, or removing small islands. Skeletonization algorithms attempt to thin geometry while preserving its topology, offering quick analysis of complex models and shape recognition. Parallel implementations in 2d and 3d exist (Tsao and Fu, 1981).

Labeling & segmentation

Segmentation involves the detection and analysis of connected regions within a voxel image. A cornerstone of such methods is connected component labeling and analysis (CCL and CCA, respectively), of which solutions using modified union-find data structures have been implemented for GPU (Hennequin et al., 2018). Analogues from 2d graphics include blob detection and pose-tracking. Distance transforms record the distance of any point from a series of points or objects. As a method of calculating generalized labeled Voronoi diagrams, or in conjunction with gradient-based watershed algorithms, they can be used for segmentation as well. The Jump Flood algorithm is a highly parallel algorithm for computing such distance transforms, and we employ this algorithm as a subroutine in Implementation. For each voxel it approximates to high accuracy the coordinates of the nearest "seed" point with time complexity $O(uvw \log u)$ where $u \times v \times w$ is the resolution of the raster grid and $u = max(u, v, w)$. Jump Flood is input insensitive, that is, its running time is independent of the number of "seeds," making this algorithm an ideal choice for the distance function of a point cloud with potentially millions of points, or of irregularly shaped objects (Rong and Tan, 2006).

IMPLEMENTATION

Herein we describe four voxel-based techniques we have developed for application to large-scale, multimaterial 3d printing of biopolymers, using a combination of those methods outlined above. Specifically, we use the volumetric model to:

- solidify a point cloud scan for use in analysis and direct-on-surface printing
- analyze overhangs in a 3d printed model to determine an optimal orientation
- automatically generate supports
- generate a continuous toolpath using voxel thinning

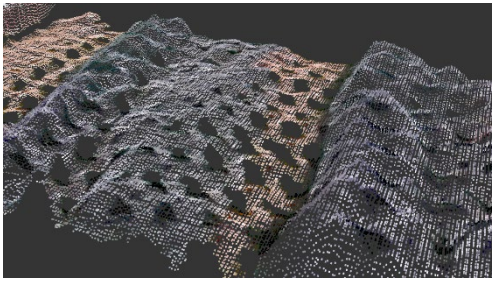

Figure 2
A pointcloud scan of a 3d printed biopolymer panel, showing holes and incomplete surface data

Solidifying pointclouds

Pointclouds are ubiquitous in 3d reality capture. In many cases the authors have used photogrammetry and LiDAR to digitize 3d printed panels with the goal of directly 3d printing onto these irregular surfaces to repair cracks, warps, and other material deformations (Thomsen et al., 2025). A challenge of working with point cloud data is that it is often irregularly distributed and

incomplete, as only visible portions of an object are captured. In robotic 3d printing, solid models of obstacles can aid in validating fabrication results or in collision detection tasks such as on-the-fly direct-on-surface printing. Pointcloud meshing algorithms are notoriously sensitive to noisy data and may fail in the presence of large holes or missing regions. When performing boolean and other volumetric operations on scanned objects, nonmanifold or open meshes are impermissible.

To address these challenges, we employ an algorithm for solid voxelization of pointclouds and meshes using the concept of the generalized winding number. A similar method is implemented in the DGTal library in C++ (Coeurjolly et al.). We adapt a GPU-friendly approximation, suitable for realtime and graphical workflows.

The winding number generalizes the notion of being "inside" an oriented surface. The true definition involves the integration in equation 1:

$$w = \frac{1}{4\pi}\int_S \frac{n \cdot (p - v)}{|p - v|^3} \qquad (1)$$

Here S is the surface, p is a point on that surface with normal n, and v is a query point which we would like to determine to be inside or outside the solid (Barill et al., 2018; Jacobson et al., 2013). In the discrete case this involves processing every triangle for every query point, which for large meshes with millions of triangles, or indeed unstructured point clouds with no triangles at all, is still unfeasible. However, given the inverse squared distance weighting of the contributions, we can form a good guess in a single-pass compute shader routine.

The winding number w is computed for every voxel. The results form a continuous distribution, with greater values indicating points that are more "inside" the undefined solid. The sensitivity of the estimate can be tuned by choice of r and k. Smaller values are faster and sufficient where the estimated normals are reliable. Thresholding these values, we obtain a solid voxel geometry.

Algorithm 1: SolidifyPointcloud(*P*,*N*)
Nearest = JumpFlood(*P*)
for each voxel *v* **do**
  $w \leftarrow 0$
  for *i* from 1 to *k* **do**
    $p \leftarrow$ evaluate(*Nearest*) at *v*
    $d \leftarrow$ distance $|p - v|$
    $n \leftarrow$ evaluate(*N*) at *p*
    $w \mathrel{+}= \frac{n \cdot (p - v_i)}{4\pi d^3}$
    $v_{i+1} = v_i + rd$ in random direction; $r < 1$
  end for
  return *w*
end for

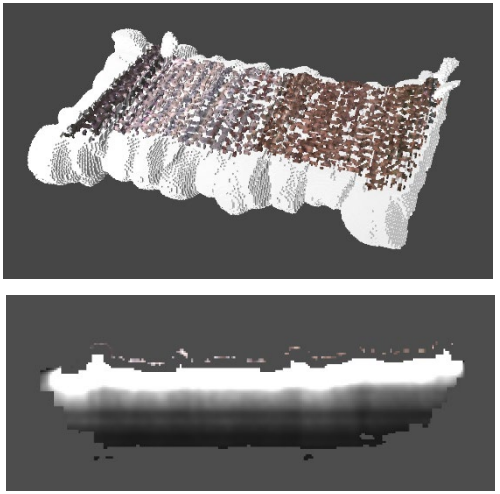

Figure 3 shows results of applying this method to an unstructured scan of a 3d printed biopolymer panel. We applied noise to the point positions and normals and removed up to 75% of the points, while maintaining consistent results.

Figure 3
Realtime solid voxelization of a noisy and incomplete pointcloud of a biopolymer panel. Voxel grayscale values decrease to black as samples are taken further from the scanned geometry.

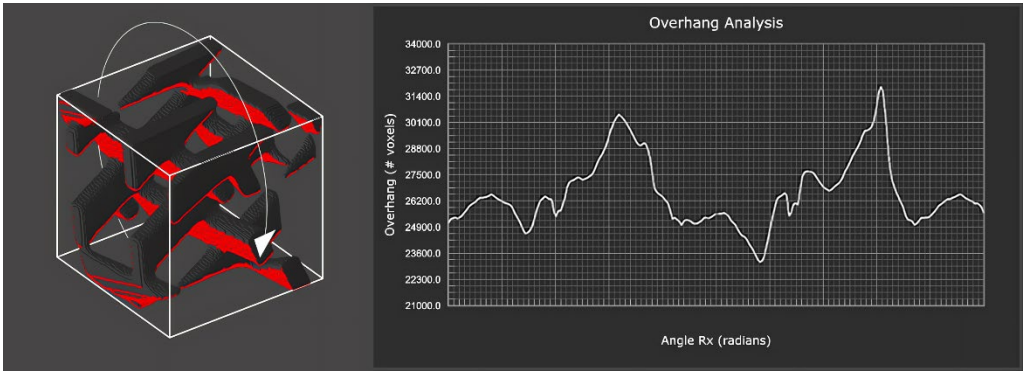

Figure 4
Overhang analysis of representative volume with respect to orientation

This method is also well-suited to Monte Carlo methods, as it can be adapted such that an initial rough estimate improves asymptotically the more samples taken. These tradeoffs between accuracy and speed make our approach adaptable to robotic fabrication scenarios.

Analyzing overhangs

The verticality of surfaces in 3d printed models is key to their stability. Biopolymer printing is even more sensitive to overhangs than commercial FDM given the low viscosity and slow curing times of the material (Nicholas et al., 2023). When choosing how to orient complex objects, it is desirable to minimize the overhangs within the print. To this end we use the voxel model and parallel computation both to quantify the presence of such overhangs as well as to locally adjust the geometry to reduce their slope.

We use the central difference method to compute per-voxel normals along the surface of our geometry. Taking the dot product of this normal with a given vector representing the downward direction and summing these values over the surface of the geometry using compute shader atomic operations or parallel prefix sum, we discretely approximating a surface integral. Figure 4 shows a visualization of these values as the red channel of a voxel texture, in this case a sample region of a custom 3d infill pattern. Note that such an approach is equally well-suited to mesh-based GPGPU solutions.

Automatic reinforcement

Once an optimal orientation is found, there may still exist overhangs or bridges in the print that could be unacceptable for fabrication. We develop an agent-based model which progressively adds supports. Unlike most print slicing software, our method deforms the geometry itself. This could be preferable when a) the geometry is too complex for cleaning removable supports; b) the orientation has already minimized overhangs, and only slight improvements are needed; or c) the print path must be continuous due to hardware restrictions, as is true in our case.

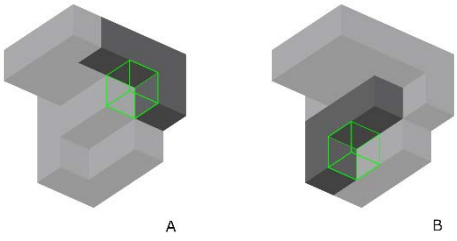

Figure 5
Support generation rules, up to mirror symmetry

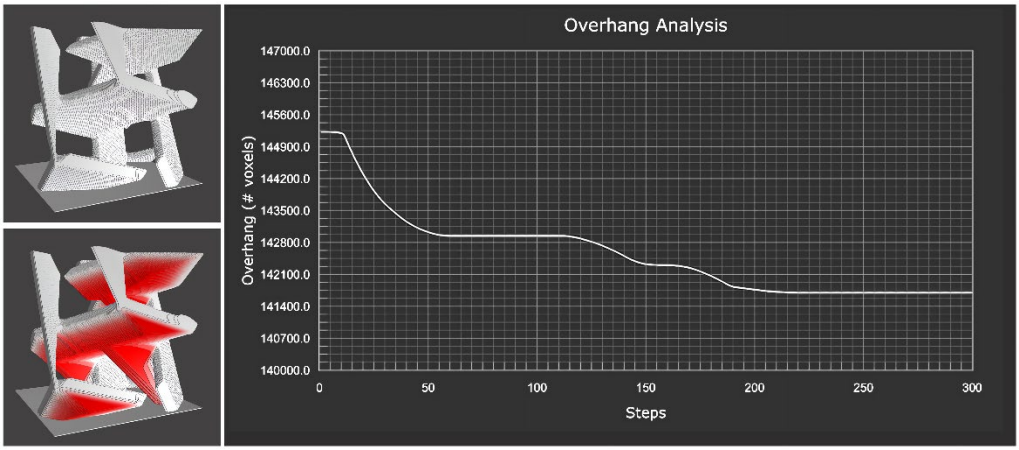

Figure 6
Before (top left) and after (bottom left) automatic support generation; increasingly red voxels indicate progressively later additions

The algorithm compares a lookup table (LUT) of predefined patterns against a voxel's 3d neighborhood, and the voxel is updated according to the matched case. In the example shown in Figure 6, additive rules adjust pitches to be at most 45 degrees. The resulting decrease in surface voxels with illegal normal is shown.

Continuous path generation

Our ABB IRB 1600 robot is fitted with a custom extruder fed by twin pumps for controlling the ratio of two blended materials, as shown in Figure 9. The material must be kept heated to flow continuously. Previous work has demonstrated the unique challenges of this setup, including the inability to precisely stop and start prints (Nicholas et al., 2023). Therefore, we require our geometry to be sliced into single continuous toolpaths.

Here we demonstrate a voxel-based solution employing a shrinkwrap procedure, which given a layer slice of solid geometry returns a single connected shape with treelike structure. Our algorithm follows the iterative paradigm of breadth-first search-based graphics processes like the grassfire algorithm. Several variants were implemented, all following the basic logic of Algorithm 2:

Algorithm 2: Shrinkwrap(V)
initialize empty voxels to 0
boundary voxels to 1

Step 1: PropagateDistance
for each empty voxel v **do**
  for each 4-connected neighbor v_{adj} **do**
    if $v_{adj} > 1$ **do**
      $v \leftarrow v_{adj} + 1$
    end if
  end for
end for

Step 2: FindRidges
for each empty voxel v **do**
  for each 8-connected neighbor v_{adj} **do**
    if $v_{adj} > v$ **do**
      $c = c+1$
    end if
  end for
  if $c < 3$ **do**
    mark $v \leftarrow$ ridge
  end if
end for

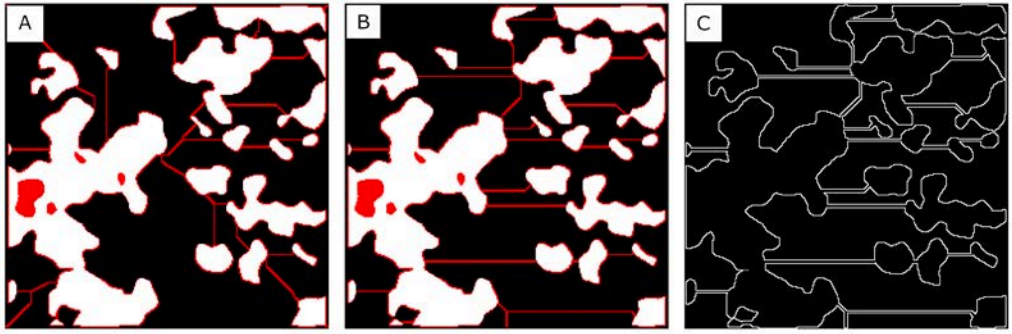

Figure 7
Continuous circuit of arbitrary layer curves; (A) shows result of rectangular initial boundary, (B) two parallel converging boundaries, (C) the resulting path

We initialize some bounding box for the geometry and compute the 4-connected distance transform initialized from the boundary, in iterative fashion. We then identify ridges in the transform, which include boundaries of solid geometry and wherever two propagating fronts have met. This shrinkwrap method ensures that the extracted ridges will form a closed loop around all geometry, with all solid regions connected. The choice of the initial boundary influences the resulting connectivity diagram. Figure 7A shows the result of a rectangular boundary shrinking from all four sides on both orthogonal axes, versus Figure 7B, which shows an initial boundary with only two opposite sides shrinking inward along the same axis. This choice could reflect an aesthetic need to conceal the added print paths, or it might concern the structural performance of the cross-section.

Notice that completely enclosed voids, colored red, are left solid. This could be seen as a useful byproduct of the algorithm, especially in zero-infill printing, as voids may be disconnected and hence unprintable. However, if it is desirable to preserve voids, one can simply apply the algorithm in negative by inverting the solid/void binary and then initializing the breadth-first search from a single seed point within each void-enclosing solid (identified, for instance, using CCL algorithms), instead of an outer bounding box. The voxels along the ridges are then deleted from, rather than added to, the original geometry.

DISCUSSION

These algorithms make heavy use of OpenGL Shading Language (GLSL) and in particular compute shaders. We implemented them in Touchdesigner, a node-based software for graphics programming, as well as in a standalone Python application using the ModernGL module. Performance metrics reflect a commercial grade NVIDIA RTX A2000 8GB GPU.

Solid voxelization is a nontrivial task, and existing methods such as the view carving tools found in the popular Open3D Python library may fail to robustly handle complex concavity or holes (Zhou et al., 2018). Even when a surface voxelization is available, as for mesh geometry, flood fill approaches are sequential, slow, and notoriously sensitive to holes and voxel connectivity, while raycasting parity tests like those of scanline approaches are sensitive to the orientation of the geometry (Håkansson, 2020). Our method builds on existing work but forgoes complex data structures in favor of approximation. In practice, the algorithm produces stable results even for very few samples, for instance for $k = 3$, $r = 0.1$ the primary compute routine consumed 1.189 msec, while the Jump Flood preprocessing step took only 0.029 msec on a $256 \times 256 \times 256$ grid. Higher r corresponds

to larger random walks from the query point, and whereas we expect the sign of the value to converge for long walks, its value may oscillate unpredictably.

The solidification algorithm relies on normal estimates; as these are often unavailable in realtime scenarios, we refer to (Liu et al., 2012) for a fast GPU-based estimation scheme. Realtime scanning for robotic fabrication is expected to become more prevalent as reality capture technology becomes more accessible (Akbari et al., 2025), therefore further developments of our approach could see interesting applications if the emphasis on realtime speeds is maintained.

Our method of overhang analysis is quite flexible; the basic approach could be employed for various tasks involving numerical integration over surfaces or solids, such as computing relative volumes of material or interfacial surface area in a multimaterial print in realtime while changes are made to the design. Analysis speeds easily maintain 60fps for a $256 \times 256 \times 256$ voxel volume, with the primary computation consuming 0.046 msec per frame. Although Figure 4 holds constant the downward direction while rotating the object, a suitable workflow would be to test uniformly distributed 3d spherical directions as the downward vector for some fixed orientation and record the minimum.

The overhang amelioration method described in this paper uses but one of many possible LUTs. It is sequential in nature and therefore less-than-realtime, although progressive results can be displayed for interactivity and the process may be terminated at any stage, for example when some threshold of added material volume or distance from the original geometry has been reached. We applied it to the geometry in Figure 6 with dimensions $256 \times 256 \times 256$ and measured 0.631 msec per step, and about 300 steps until convergence, although this could vary widely for different geometries. Nevertheless, as demonstrated by the monotonicity of the graph, reasonable results should be obtained within 1-2 seconds in non-interactive mode. Of interest in further work would be a means of parametrizing the target maximum incline.

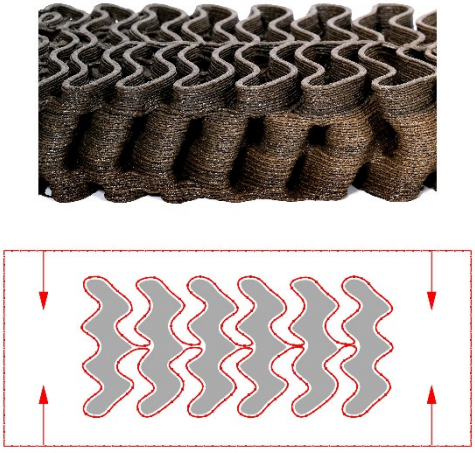

Figure 8
Section of a biopolymer 3d print showing a centrally symmetric continuous toolpath like those obtained from the shrinkwrap algorithm

The continuous path generation algorithm is, like the support generation technique, also iterative and thus not realtime, although progressive results can be viewed interactively. Moreover, in the form in which it has been presented it is primarily 2-dimensional, operating independently on each layer. This reflects the most common approach to slicing print geometry. However, added geometry may not be supported by the layer below in a 3d printing scenario, therefore it is of interest to adapt this algorithm to 3d in future work, as well as to further explore edge cases.

Figure 9
Custom dual pump biopolymer printing setup using an ABB IRB 1600 for control, showing an in-progress 1.3 x 1.6 m^2 panel print.

We prototyped our implementations of many of the basic morphological operations first with DeepSight, a plugin for Grasshopper wrapping the OpenVDB C++ library for sparse voxel grids (Svilans et al., 2023). Although many of the algorithms presented in this paper benefit from large speedups when ported to shaders, OpenVDB's ability to compress much larger voxel domains suggests next steps for this research. Indeed, memory limits remain a challenge for graphics processors, as resolutions beyond 1024^3 voxels can quickly exceed the capacities of most modern GPUs. Despite continuous advancements in hardware capabilities, solutions for streaming voxel data would be desirable. Further work in this area should consider adaptations of the algorithms for chunked voxel data or hierarchical data structures like (sparse) voxel octrees.

CONCLUSIONS

Our heuristics demonstrate the usefulness of a voxel-based description of geometry for solving design and fabrication problems from large-scale multimateral 3d printing. We establish a firm foundation in digital geometry with the intention of designing fast, transparent algorithms, while leveraging the growing accessibility of GPU programming and the extensive literature on 2d graphics algorithms alongside their 3d analogues. Although volumetric modeling with voxels has a long history, tailormade algorithms for digital fabrication tasks capitalizing on the regularity and universality of this data structure represent an under-explored area. This research contributes to that corpus in anticipation of its growing relevance to the fields of architecture and computational design.

REFERENCES

Akbari, S., Tamke, M., Ramsgaard Thomsen, M. (2025). 'A Workflow Integrated, Adaptive High-Res High Quality Robotic 3d Capture Environment' - Shiva. Presented at the *European Conference on Computing in Construction*, Porto, Portugal.

Barill, G., Dickson, N.G., Schmidt, R., Levin, D.I.W., Jacobson, A. (2018). (2018) 'Fast winding numbers for soups and clouds,' *ACM Transactions on Graphics*, 37(4), pp. 1–12. https://doi.org/10.1145/3197517.3201337

Bernhard, M., Hansmeyer, M., Dillenburger, B. (2018). 'Volumetric Modelling for 3D Printed Architecture'. Presented at the *Advances in Architectural Geometry*, pp. 392–415.

Coeurjolly, D., et al. (n.d.) DGtal - Digital Geometry Tools and Algorithms Library.

Eisemann, E., Décoret, X. (2008). 'Single-Pass GPU Solid Voxelization for Real-time Applications', in: *Proceedings of Graphics Interface*. Ontario, Canada, pp. 73–80.

Goldstein, R.; Walmsley, K.; Morris, N. & Tessier, A. (2023). 'Algorithms for Voxel-Based Architectural Space Analysis'. in María Julia Blas & Gonzalo Alvarez, ed., 'ANNSIM' , IEEE, , pp. 508-519.

Håkansson, T.(2020). 'A Comparison of Optimal Scanline Voxelization Algorithms' (Master of Science Thesis in Computer Science and Software Engineering). Linköping University.

Hennequin, A., Lacassagne, L., Cabaret, L., Meunier, Q.(2018). 'A New Direct Connected Component Labeling and Analysis Algorithms for GPUs'. Presented at the *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pp. 76–81. https://doi.org/10.1109/DASIP.2018.8596835

Jacobson, A., Kavan, L., Sorkine-Hornung, O. (2013). 'Robust Inside-Outside Segmentation Using Generalized Winding Numbers'. *ACM Trans. Graph*. 32, 1–12. https://doi.org/10.1145/2461912.2461916

Jones, M.W., Baerentzen, J.A., Sramek, M.(2006). '3D Distance Fields: A Survey of Techniques and Applications'. *IEEE Transactions on Visualization and Computer Graphics* 12,

581–599. https://doi.org/10.1109/TVCG.2006.56

Liu, M., Pomerleau, F., Colas, F., Siegwart, R. (2012). 'Normal Estimation for Pointcloud Using Gpu Based Sparse Tensor Voting', in: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Presented at the *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), IEEE*, Guangzhou, China, pp. 91–96. https://doi.org/10.1109/ROBIO.2012.6490949

Michalatos, P., Payne, A. (2016). 'Monolith: The Biomedical Paradigm and the Inner Complexity of Hierarchical Material Design'. Presented at the *eCAADe 2016: Complexity & Simplicity*, Oulu, Finland, pp. 445–454. https://doi.org/10.52842/conf.ecaade.2016.1.445

Nicholas, P., Lharchi, A., Tamke, M., Valipour, H.G., Eppinger, C., Sonne, K., Rossi, G., Thomsen, M.R. (2023). 'Biopolymer Composites in Circular Design: Malleable Materials for an Instable Architecture', in: *Habits of the Anthropocene: Proceedings of the 43rd Annual Conference of the Association for Computer Aided Design in Architecture*. Presented at the *ACADIA*, University of Colorado, Denver, pp. 166–173.

Reinhard, K., Rosenfeld, A. (2004). *Digital Geometry, The Morgan Kaufmann Series in Computer Graphics*. Morgan Kaufmann Publishers.

Rong, G., Tan, T.-S. (2006). 'Jump flooding in GPU with applications to Voronoi diagram and distance transform', in: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games - SI3D '06*. Presented at the *2006 symposium*, ACM Press, Redwood City, California, p. 109. https://doi.org/10.1145/1111411.1111431

Rossi, G., Rech, A., Harding, J., Nicholas, P., Daugaard, A.E., Tamke, M., Thomsen, M.R. (2023). 'Predicting the Recipe: Leveraging Parameterless Self-Organizing Maps (PLSOM) with Geometrical Polynomial Fitting (GPF) to predict biopolymer composites characterization with small datasets'. Presented at the *Future of Construction Symposium*, Munich, Thuringia, Germany.

Schroeder, W.J., Maynard, R., Geveci, B. (2015). Flying Edges: A High-Performance Scalable Isocontouring Algorithm. Presented at the *5th IEEE Symposium on Large Data Analysis and Visualization,* Chicago, IL, USA https://doi.org/10.13140/RG.2.1.3415.9609

Schwarz, M., Seidel, H.-P. (2010). 'Fast Parallel Surface and Solid Voxelization on GPUs'. *ACM Trans. Graph*. 29, 1–10. https://doi.org/10.1145/1882261.1866201

Stamm, C., Vilbrandt, T., Pasko, A., Petter Christensen, T., Adzhiev, V. (2015). MeshUp.

Svilans, T., Gatz, S., Tyse, G., Ramsgaard Thomsen, M., Ayres, P., Tamke, M. (2023). 'Deep Sight - A Toolkit for Design-Focused Analysis of Volumetric Datasets', in: Gengnagel, C., Baverel, O., Betti, G., Popescu, M., Thomsen, M.R., Wurm, J. (Eds.), *Towards Radical Regeneration. Springer International Publishing*, Cham, pp. 543–555. https://doi.org/10.1007/978-3-031-13249-0_43

Thomsen, M.R., Nicholas, P., Chiujdea, R.-S., Nielsen, S.D., Sonne, K., Eppinger, C. (2025). 'Additive Manufacturing for Repair: Continual Construction Through Bio-Based Materials'. *3D Printing and Additive Manufacturing* https://doi.org/10.1089/3dp.2023.0344

Tsao, Y.F., Fu, K.S. (1981). 'A Parallel Thinning Algorithm for 3-D Pictures'. *Computer Graphics and Image Processing* 17, 315–331. https://doi.org/10.1016/0146-664X(81)90011-3

Zhou, Q.-Y., Park, J., Koltun, V., 2018. Open3D: A Modern Library for 3D Data Processing. https://doi.org/10.48550/ARXIV.1801.09847