ALGORITHMS FOR VOXEL-BASED ARCHITECTURAL SPACE ANALYSIS

Rhys Goldstein
Kean Walmsley
Nigel Morris
Alexander Tessier

Autodesk Research
661 University Ave
Toronto, ON, CANADA
{rhys.goldstein, kean.walmsley, nigel.morris, alex.tessier}@autodesk.com

ABSTRACT

The analysis of building geometry is an essential capability for generative design, multi-agent simulation, and other computational paradigms being explored in architectural research and practice. We propose a set of voxel-based algorithms that operate on a common data structure to support pathfinding, visibility calculations, direct sunlight calculations, and custom workflows that combine these basic spatial analyses. Our data structure is a binary 3D voxel model represented using a memory-efficient encoding that allows large buildings to be analyzed at fine resolutions. The algorithms are based on underutilized methods from the level set mathematics community that allow short travel paths and sightlines to be approximated without ray casting. The proposed approach has been made available in a visual programming package called VASA (Voxel-based Architectural Space Analysis). We present an example that demonstrates how voxel-based operations and analyses can be chained together to produce human-centric metrics.

Keywords: voxelization, pathfinding, visibility, daylight, generative design.

1 INTRODUCTION

Pathfinding, visibility, daylight, and other spatial analyses are increasingly used in architectural research and practice to compute metrics for generative design. Nagy et al. (2017) applied space analysis to evaluate walking distances to key amenities (adjacency), the likelihood of being distracted while working, views to the outside, and access to daylight. Illustrated in Figure 1, these metrics were optimized as part of a generative design workflow that was used to design a 250-person office space in Toronto. Spatial analyses can also be used to predict the experiences of building occupants over the course of a typical day. Danell, Ámundadóttir, and Rockcastle (2020) applied daylight simulation, which could be considered a sophisticated form of space analysis, to generate daily profiles of eye-level light exposure with the aim of quantifying the circadian performance of a building. Going a step further, space analysis plays a critical role in the development of multi-agent simulations capturing the interactions of people in buildings. These simulations require pathfinding analyses to generate walking routes (Pelechano, Allbeck, and Badler 2008), and sometimes incorporate domain-specific analyses such as acoustics (Schaumann et al. 2020).

Space analysis methods make use of a variety of spatial representations. Pathfinding, for example, may be performed using a grid, a waypoint graph, or a navigation mesh (Sturtevant 2019). Visibility calculations

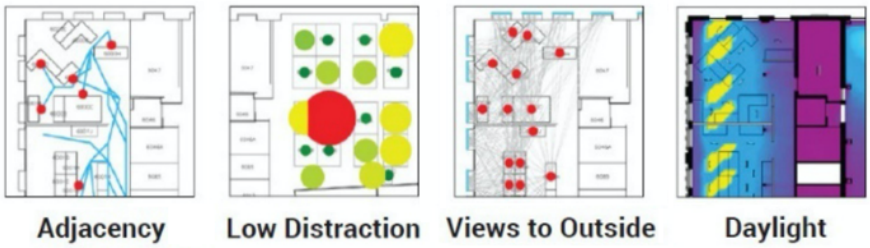


Figure 1: Four of the generative design metrics from Nagy et al. (2017).

are most commonly performed using either a set of lines or polygons in 2D, or a triangle mesh in 3D. Although there are advantages to employing different representations for different analyses, the use of a common representation of space promises to ease the task of preparing architectural models for analysis, integrating simulation code, and combining analysis results. Multiple analyses are often required to compute a generative design metric or provide the foundations of a simulation platform.

In this paper, we propose and report on a 3D grid-based approach for space analysis in which architectural models are first voxelized and then processed using a succession of voxel-based algorithms. The approach is based on a common representation of space: a 3D voxel model in which every voxel is either filled or unfilled, and where "plates" of 16×16 voxels are stored using a memory-efficient run-length encoding scheme that allows relatively large buildings to be analyzed at fine resolutions. We outline a collection of voxel-based algorithms including ones that (1) prepare architectural models for path analysis, (2) generate distance fields for pathfinding, and (3) compute visible regions for view analysis, direct sunlight, and artificial lighting or sensor coverage. The algorithms use binary voxel models as the primary format for both input and output data, allowing voxel-based operations and analyses to be chained together to create geometry processing pipelines and other custom workflows.

Our implementation of the proposed approach is publicly accessible via a tool called VASA, short for Voxel-based Architectural Space Analysis. VASA takes the form of a package to be used within the Dynamo visual programming environment, and can be installed via Dynamo's package management system. After reviewing related work and presenting the key algorithms, we provide an example of how VASA's voxel-based capabilities can be combined to produce human-centric metrics.

2 RELATED WORK

Algorithms involving geometry can often be classified as either vector-based or grid-based according to how they represent space. Our proposed voxel-based analysis approach draws heavily from recent developments in grid-based navigation and visibility.

2.1 Vector-based Space Analysis

Vector-based approaches represent geometry using sets of connected elements such as points, lines, curves, polygons, surfaces, and polyhedra. A canonical example of a vector-based representation is the 2D visibility graph (Lozano-Pérez and Wesley 1979), where (1) walls and other obstacles are represented by polygons, (2) the corners of these polygons become vertices in a graph, and (3) any clear sightlines between those vertices become edges in the graph. The graph can then be used to find shortest paths, and similar analytic techniques can be used to compute the polygonal region visible from a point (Ghosh 2007). Another vector-based approach involves representing a building using a 3D triangle mesh, then casting numerous rays through the model to (a) compute the region visible from a point, (b) identify the surfaces that will receive direct daylight given a particular sun direction, or (c) render 2D images of the 3D model.

Vector-based approaches have the potential to be highly accurate and efficient, since one can model regions of importance using a dense set of points and surfaces while approximating less important regions using a minimal number of elements. In practice, however, it can be challenging to ensure that only the most essential features are finely resolved. Another advantage of vector-based approaches is that architectural models are usually authored using a vector-based design tool.

The Topologic toolkit by Aish et al. (2018) is an example of a visual programming package dedicated to the vector-based approach for architectural space analysis. Whereas conventional vector-based design tools represent building geometry as collections of solid manifolds, Topologic is notable for its use of non-manifold vector-based representations.

2.2 Grid-based Space Analysis

Grid-based approaches represent geometry using a regular grid of points with associated states or properties. A strictly grid-based method visits the points using some form of graph, tree, or array traversal algorithm, and processes them by propagating information from each point to a predetermined set of neighbors. On a 2D grid, the neighbors are typically the four nearest points along the two primary axes, or the eight nearest points including the diagonal neighbors.

Grid-based analyses tend to be simple and robust. By representing architectural spaces and physical elements using discrete sets of 2D cells or 3D voxels, one avoids the tiny gaps, the thin overlapping regions, and other geometric anomalies that plague all but the most carefully implemented vector-based algorithms. Also, whereas vector-based methods may require sophisticated re-meshing techniques to optimize the resolution of the model, grid-based approaches allow the tradeoff between speed and accuracy to be conveniently adjusted by varying the grid spacing.

The SpaceAnalysis tool by Goldstein et al. (2020) is an example of a visual programming package providing 2D grid-based spatial algorithms. This paper demonstrates how similar capabilities can be achieved in 3D.

2.3 Grid-based Navigation

Grid-based pathfinding is a well-established and widely used alternative to vector-based navigation methods such as waypoint graphs, visibility graphs, or navigation meshes. The basic approach to grid-based pathfinding involves converting a grid into a graph by connecting neighboring points, disconnecting any points or edges that are cut off by walls or other obstacles, and using a classic shortest path search such as Dijkstra's algorithm (Dijkstra 1959) or A* (Hart, Nilsson, and Raphael 1968).

The well-known drawback to grid-based navigation is that there is usually a multitude of shortest grid paths between a given starting location and a given destination, and choosing one of these paths in an arbitrary manner tends to result in an indirect, unnatural-looking route. There are several known ways to solve this problem and generate highly direct paths despite the use of a grid. We describe the main strategies as sightline testing, path counting, and interpolation.

The sightline testing strategy involves performing a multitude of line-of-sight tests during the path search. This combination of grid-based and vector-based techniques is demonstrated by the Theta* "any-angle" pathfinding algorithm (Daniel et al. 2010), an adaptation of A* that tests a subset of possible straight-line shortcuts and usually eliminates unnecessary detours.

The path counting strategy involves counting the number of shortest grid paths that traverse each grid point, then selecting the points with the highest counts. The resulting central grid path is highly direct compared

with most shortest grid paths. Originally developed for the SpaceAnalysis package, Goldstein et al. (2022) report path counting to be 2-3 times faster than Theta* for maps that resemble real-world environments.

The interpolation strategy involves interpolating the travel distances associated with two nearby grid points to estimate the distance at a point in between, and then extrapolating from this intermediate point to compute a travel distance associated with a third grid point. The Field D* interpolation method is well known in the video game and robotics research communities (Ferguson and Stentz 2006), and Field A* deserves recognition as a simplification that isolates the core technique (Uras and Koenig 2015). Though there are differences in conventions, interpolation-based pathfinding was first developed in the level set community using implicit instead of explicit geometry. The approach was proposed by Sethian (1996) as a 4-neighbor algorithm known as the Fast Marching Method, and our analysis suggests that its 8-neighbor variant by Danielsson and Lin (2003) uses the same underlying formula as Field A*. Applications of the Fast Marching Method and various improvements are presented by Valero-Gomez et al. (2013).

2.4 Grid-based Visibility

Visibility analysis is almost always performed by applying analytic calculations or ray casting to a vector-based representation of space. Although strictly grid-based visibility algorithms are rarely used in practice, they are theoretically sound and offer the same benefits as other grid-based approaches: ease of implementation and a convenient means of adjusting the tradeoff between speed and accuracy. The limited use of grid-based visibility is likely due to a lack of awareness that such an approach exists and has practical advantages for certain applications.

Grid-based visibility was introduced by Tsai et al. (2004) as a 2D or 3D level set method, and shown by Kao and Tsai (2008) to converge on exact results. This level set method was adapted from implicit to explicit geometry by Goldstein et al. (2022), who refer to the overarching approach as linear grid-based visibility and show that the technique (a) can be applied using a variety of grid neighborhoods, (b) is mathematically equivalent to counting the fraction of unblocked shortest grid paths, and (c) can be demonstrated to converge on exact results using the central limit theorem instead of numerical analysis. The SpaceAnalysis package implements 8-neighbor visibility by counting, which produces the same results as linear grid-based visibility ignoring rounding errors.

Another grid-based visibility algorithm was proposed by Fisher-Gewirtzman, Shashkov, and Doytsher (2013) for analyzing urban environments in 3D. This bi-linear grid-based visibility approach differs from 3D implementations of linear grid-based visibility in that it uses bi-linear interpolation within rectangular pyramidal cones rather than linear interpolation within tetrahedral cones.

3 METHODS

The proposed approach to voxel-based architectural space analysis is developed as a collection of methods associated with (1) a common representation of space, (2) voxel model operations, (3) a path model representing walkable or traversable areas, and (4) pathfinding and visibility analyses.

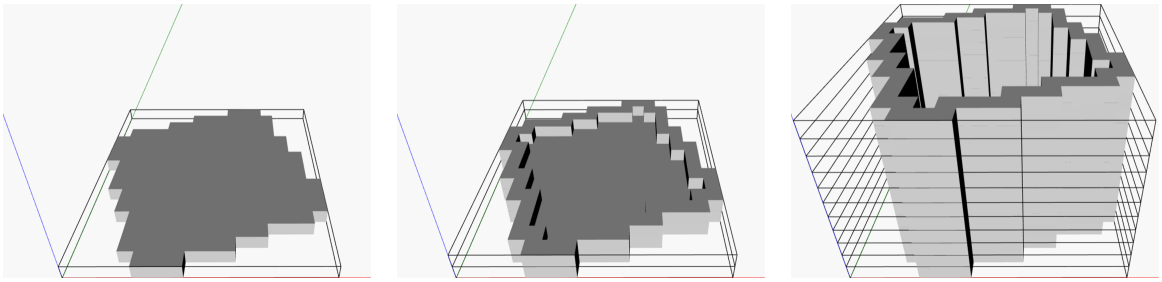
3.1 Voxel Model Representation

Establishing a common representation of space has the effect of uniting a collection of analyses and operations into a single approach, allowing a diverse set of capabilities to be rapidly combined while minimizing the need to convert spatial information from one format to another. To serve as a foundation for building simulation and generative design, our common representation is based on the following criteria:

- Support for **reasonably large models** containing a building and its surrounding neighborhood.
- Support for **reasonably fine resolutions** such that paths can be generated through narrow doorways.
- Support for **low memory data structures**, allowing dozens of intermediate voxel models to be instantiated as part of a geometry processing pipeline or generative design workflow.
- Support for **fast voxel-based algorithms**, avoiding brute force approaches such as checking a line of sight for every voxel.
- A design **optimized for buildings**, which feature a prevalence of walls, columns, vertical glazing, and other vertical elements.

The chosen representation is a 3D rectangular grid of voxels. The size of every voxel is specified by a parameter called the coarse voxel width, which we recommend be fixed at 1 meter (preferred) or 4 feet. Vertical and horizontal level of detail (LOD) parameters indicate the number of times the coarse voxel width is divided by 2. We recommend that the LOD be fixed at 2 in the vertical direction, meaning that the height of every voxel is 25 cm or 1 foot (1 meter or 4 feet, divided by 2 twice). In the horizontal direction, the LOD can be adjusted to control the tradeoff between speed and accuracy. In metric units, the horizontal LOD can be set to 2 for 25×25×25 cm voxels, which is often suitable for visibility at an outdoor scale; 3 for 12.5×12.5×25 cm voxels, which is normally sufficient for indoor pathfinding; and 4 for 6.25×6.25×25 cm voxels (the default recommendation), which tends to be sufficient for pathfinding even in the presence of detailed obstacles like furniture. Tall and narrow voxels are optimal for the analysis of buildings, where doorways can face any direction in the horizontal plane but rise straight up in the vertical dimension.

We store voxel data in the form of voxel plates, which are horizontal arrays of 16×16 voxels. Each row of 16 voxels in a plate is represented using a 16-bit unsigned integer, where each bit determines whether the associated voxel is filled (1) or unfilled (0). This encoding allows certain voxel model operations to be carried out by processing 16 voxels at once. For example, computing the union of two voxel models is performed by applying bitwise OR instructions to 16-voxel rows instead of individual voxels.



(a) The bottom voxel plate (b) The bottom two plates (c) All eleven voxel plates

Figure 2: A voxel stack in which the first two 16×16 voxel plates are unique.

A voxel model is partitioned into voxel stacks containing up to 64 voxel plates stacked one on top of the other. A typical model may be several stacks tall and many stacks long and wide. The stacks are run-length encoded in the vertical direction, meaning that only a single voxel plate is stored for every group of consecutive identical plates. In Figure 2, for example, only the first two voxel plates are retained in memory since the nine plates above are identical to the second. A separate list of indices records the vertical positions at which a voxel plate differs from the one below. This compression scheme tends to be highly efficient for large regions of empty voxels representing air, large regions of filled voxels representing earth, and regions inside buildings in which the only physical elements are walls, columns, and vertical glazing.

We find that reasonably large architectural models can be input as a triangle mesh and efficiently voxelized at a 6.25×6.25×25 cm resolution using the formula proposed by Schwarz and Seidel (2010), even without

GPU acceleration. We implemented their surface voxelization approach for single-threaded execution on a CPU. Although buildings are typically designed as a collection of solid manifolds, triangle mesh representations are usually also available since they are used to render 3D geometry on the screen.

3.2 Voxel Model Operations

An essential component of the proposed approach is the ability to perform generic operations on voxel models whenever needed. These include four standard solid operations: computing the **union** or **intersection** of a set of voxel models, **subtracting** one voxel model from another, and **inverting** a voxel model so that all filled voxels become unfilled and vice versa within a set of 3D rectangular extents. We also include three fill operations: flood filling a 3D region of unfilled voxels **from a point**, flood filling **all completely enclosed regions** in the model, and flood filling **all exposed regions** that are not completely enclosed.

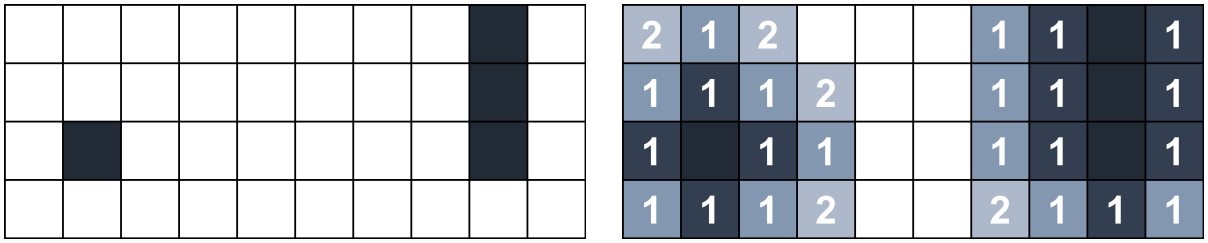
Of particular importance are ten surface operations, where "surface" in this context refers to the interfaces between regions of filled and unfilled voxels. Six of the ten surface operations **expand** or **reduce** all filled regions by a specified number of voxels in the positive direction, and a specified number in the negative direction, along the **X axis**, the **Y axis**, or the **Z axis**. There is also an operation that **expands surfaces by one voxel** according to two parameters: the horizontal adjacency and the vertical adjacency. If an unfilled voxel (a) has c_h adjacent filled neighbors in any of four horizontal directions, where c_h equals or exceeds the horizontal adjacency, and (b) has c_v adjacent filled neighbors in either vertical direction, where c_v equals or exceeds the vertical adjacency, then the voxel becomes filled. Similarly, there is an operation that **reduces surfaces by one voxel** according to a similar pair of parameters.

The final two surface operations, **expand horizontally** and **reduce horizontally**, expand or reduce surfaces in the horizontal direction according to a diameter offset parameter that loosely represents the width of a traveling agent in voxel spacings. The operations work by repeatedly expanding or reducing the surfaces by one voxel as described above, with a vertical adjacency of 0 and a sequence of horizontal adjacency values. The sequences for the first 12 offset values are shown in Table 1. The sequences for larger offsets continue the pattern of taking the last pair of sequences, inserting a 1 at the front to obtain the next pair of sequences, and inserting 1, 1, 2 to obtain the pair after that.

Table 1: Horizontal adjacency sequences for horizontal expansion and reduction operations.

Offset	Sequence	Offset	Sequence	Offset	Sequence
1		5	1, 2, 1	9	1, 1, 1, 2, 1
2	2	6	1, 1, 2	10	1, 1, 1, 1, 2
3	1	7	1, 1, 2, 1	11	1, 1, 2, 1, 1, 2, 1
4	1, 2	8	1, 1, 1, 2	12	1, 1, 2, 1, 1, 1, 2

The sequences were chosen to expand obstacles as much as possible without closing off passageways that are at least as wide as the diameter offset. This means that if the offset parameter matches the diameter of an agent, the agent should be able to move through the passageway. The 6-voxel wide passageway in Figure 3a illustrates how the process works. Suppose that the obstacles are expanded according to a diameter offset of 6. The corresponding sequence of horizontal adjacency values is 1, 1, 2, and Figure 3b shows in fading colors which voxels become filled during each of the three iterations. The passageway between the two obstacles shrinks but does not completely close, meaning that an agent with a diameter of approximately 6 voxels could get through. If the diameter offset had been 7, however, a fourth iteration with a horizontal adjacency of 1 would have caused the passageway to close. This algorithm provides a simple, memory-efficient alternative to more sophisticated level set expansion and reduction approaches, such as the Fast Marching Method. Obstacles grow or shrink in approximate correspondence with distance.



(a) A 6-voxel wide passageway between obstacles

(b) Obstacles expanded for a diameter offset of 6

Figure 3: An example of horizontal surface expansion preventing travel near obstacles.

3.3 Path Model Generation

Impassable objects in an architectural model include floors, ceilings, roofs, walls, columns, glazing, and furniture, for example, as well as outdoor elements such as the ground, tree trunks, dense vegetation, fences, and other barriers. Once all such elements are converted into a single input voxel model, we generate another voxel model called a path model representing all traversable or walkable areas.

Conceptually, the first step in generating a path model is to take the input voxel model, expand all surfaces by one voxel in the positive Z direction (up), then subtract away the initially filled voxels. This procedure yields a one-voxel-high layer of potentially traversable voxels above every solid surface. However, if all of these voxels were to be considered "traversable", then paths might be generated underneath tables, through tiny gaps, across narrow beams, and up steep slopes. Agile individuals might be able to negotiate these difficult routes, but our intent is to identify areas where people can comfortably move.

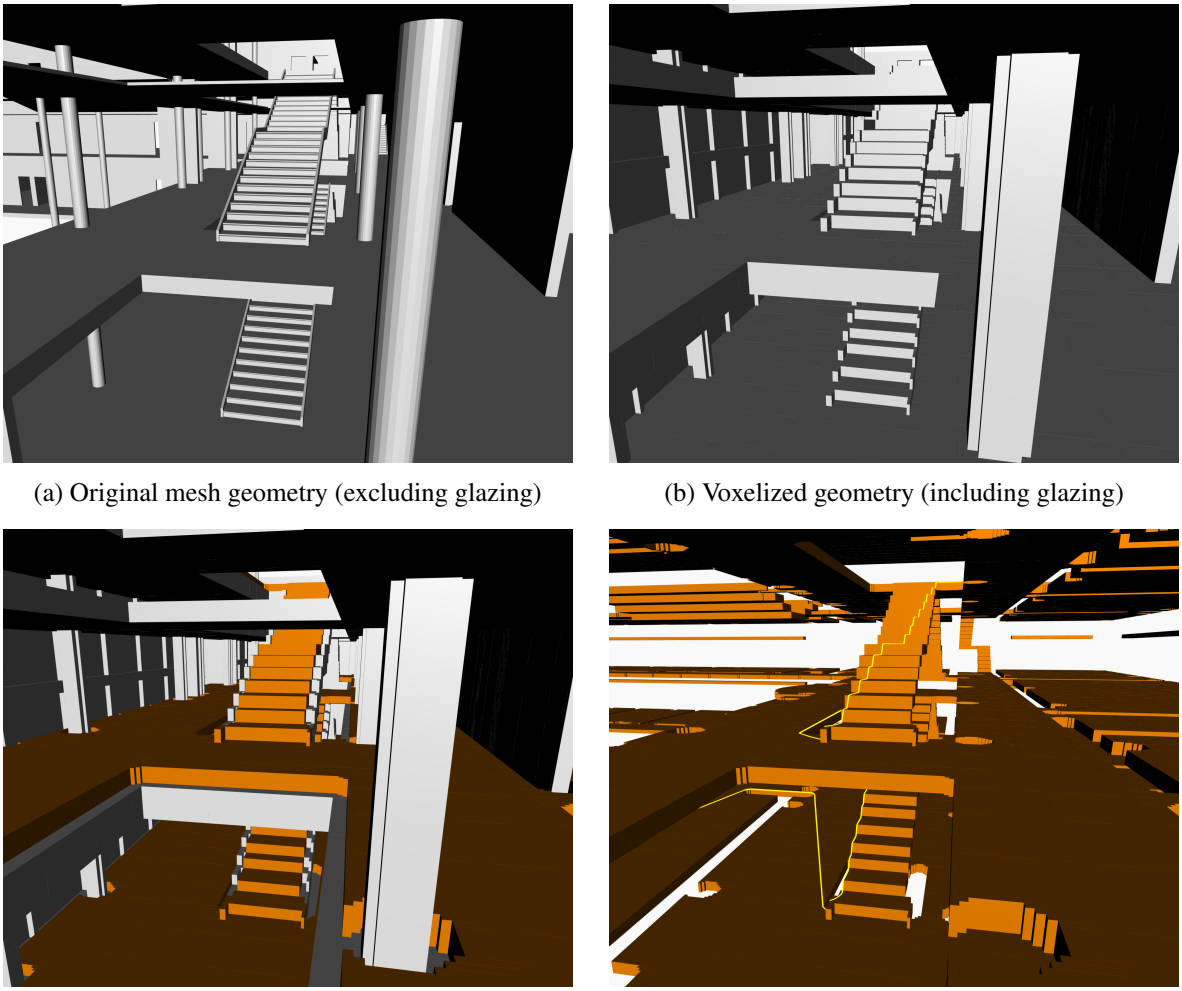
To prevent travel underneath tables or through small crawl spaces, we expand all obstacle surfaces by $n_v - 1$ voxels in the negative Z direction (down) where n_v is the agent height offset, the approximate height of a person in voxels. It is important to underestimate this parameter, or regular doorways may end up becoming closed off. We recommend $n_v = 4$, an agent height of 1 meter assuming a voxel height of 25 cm.

To prevent travel through tiny gaps, we expand obstacles horizontally as described in Section 3.2 based on an agent diameter offset parameter n_h. We recommend $n_h = 7$ for 6.25 cm wide voxels (a horizontal LOD of 4), and $n_h = 3$ for 12.5 cm wide voxels (a horizontal LOD of 3). These numbers correspond with an agent width of about 40 cm.

An important detail for 3D pathfinding is that solid obstacles be reduced by one voxel in the positive Z direction before they are expanded and subtracted away from the traversable voxels. Lowering obstacles by one voxel allows people to ascend and descend stairs, ramps, and gradual slopes, and negotiate small bumps or depressions in outdoor terrain. Our assumption is that people can move up or down one voxel at a time, but not two. This convention demands that voxels be slightly taller than the height of a step in a typical staircase, which is why we recommend fixing the voxel height at 25 cm or 1 foot.

To prevent travel across beams, close to drop-offs, or up and down excessively steep slopes, we generate "virtual obstacles" by expanding solid surfaces upward by two voxels and then inverting the result. These virtual obstacles are expanded in the same manner as physical obstacles, and subtracted from the set of potentially traversable voxels.

Figure 4 illustrates the results of this path model generation procedure. Our method can be contrasted with the human-centric accessibility graph by Schwartz (2021), which has a similar purpose but is constructed using ray casting instead of voxels, as well as the indoor drone navigation model of Li et al. (2018), which uses voxels but identifies accessible air space rather than walkable surfaces.



(a) Original mesh geometry (excluding glazing)

(b) Voxelized geometry (including glazing)

(c) With traversable voxels (orange)

(d) Traversable voxels only, plus a 3D path (yellow)

Figure 4: Generation of a path model starting from a triangle mesh and ending with the traversable voxels.

3.4 Voxel-based Analyses

The primary voxel-based analyses supported by our approach are 3D pathfinding, visibility from a point, and visibility from a direction. Our implementations of these analyses are based on the level set interpolation techniques reviewed in Section 2, which have previously seen little use in the architectural domain.

Once a path model is generated as outlined in Section 3.3, pathfinding can be performed by adapting one of the navigation strategies of Section 2.3 to operate on the traversable voxels. We avoid the sightline testing strategy, which we consider inefficient for computing long, meandering paths on a high-resolution grid. We regard path counting as a faster strategy for generating a single path or a set of paths. However, we chose the interpolation strategy because it can produce an accurate distance field: an estimate of the walking distance from a source voxel to every other traversable voxel that can be reached by an agent. By choosing a maximum threshold distance, one can represent a distance field as a voxel model and use it as an analysis result (Figure 5). Our interpolation-based pathfinding implementation is similar to that of Danielsson and Lin (2003), though modifications were necessary to (a) associate distances with the centers instead of the corners of voxels, and (b) allow agents to travel diagonally upward or downward.

We implemented the visibility from point analysis using the standard 6-neighbor 3D linear grid-based visibility method described in Section 2.4, and a nearly identical interpolation was used to implement visibility from a direction. Visibility from a point can be used to analyze a person's view from a specific location, or to estimate the coverage of an artificial light source, motion sensor, or security camera. Visibility from a direction can be used to approximate the area receiving direct sunlight at a particular moment in time, assuming a clear sky (Figure 5). Since a person can see but not walk through a window, we recommend that glazing be voxelized separately from other solid geometry. The glazing voxels can then be incorporated with a union operation to generate a path model for pathfinding, but excluded from any visibility analysis.

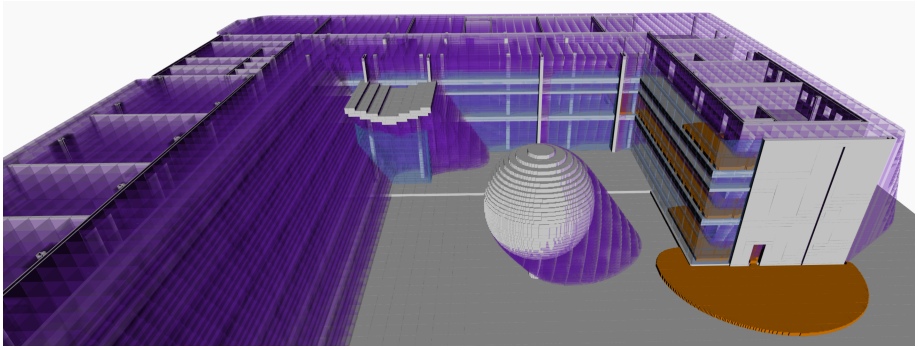


Figure 5: Shadows (purple) created by inverting direct sunlight results, and a distance field (orange).

4 EXAMPLE

Though unconventional, voxel-based algorithms can be used in combination to compute a wide range of human-centric metrics. The approach is particularly useful for early stage analyses and generative design processes where approximate results are usually good enough. Here we present an example of such a calculation using the VASA package that implements the proposed approach. VASA is similar to Topologic and SpaceAnalysis in that it provides a set of algorithms that operate on a common representation of space, though the representation itself differs. By using 3D voxel models as both the input and output of most analyses and operations, VASA allows metrics like the one in the example to be computed solely by linking visual programming nodes and avoiding traditional textual code.

The problem we tackle in the example is to approximate the fraction of some target that is visible from a point in the building. In this case the target is the tree shown using yellow voxels in Figure 6, but similar analyses could be performed for parks, waterfronts, heritage buildings, or other visually appealing features.

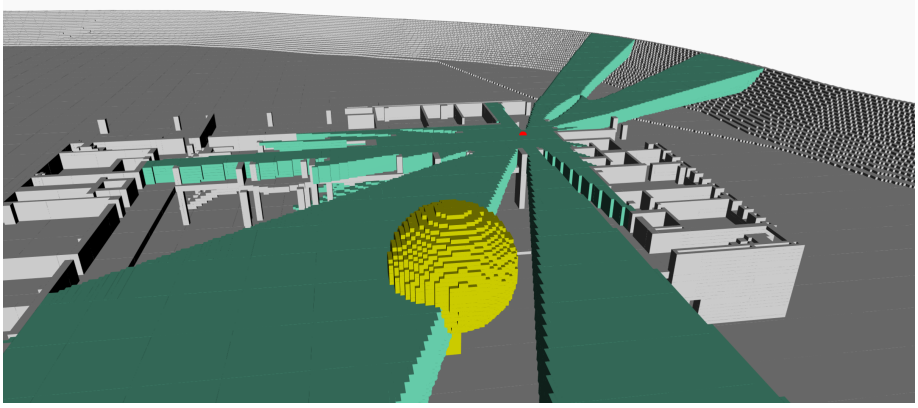
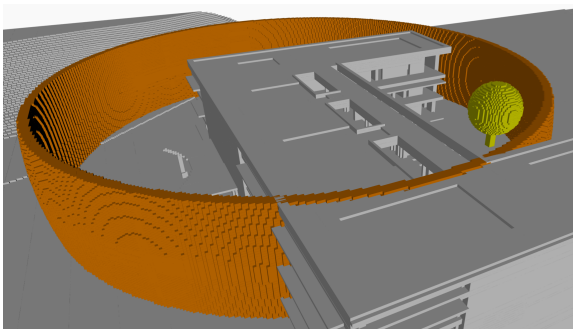


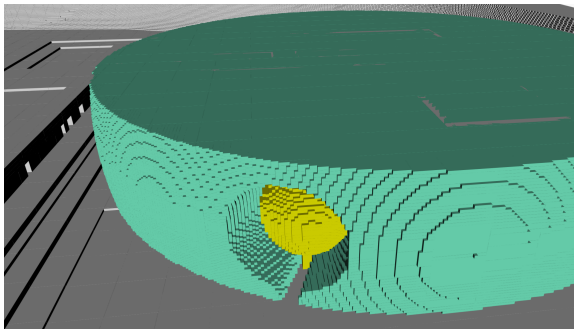
Figure 6: A problem of computing the fraction of a target that is visible from a point.

Different strategies could be used to estimate the visible fraction of the target. The solution we present in Figure 7 is not the simplest, but it should produce accurate metrics given a sufficiently fine grid resolution.

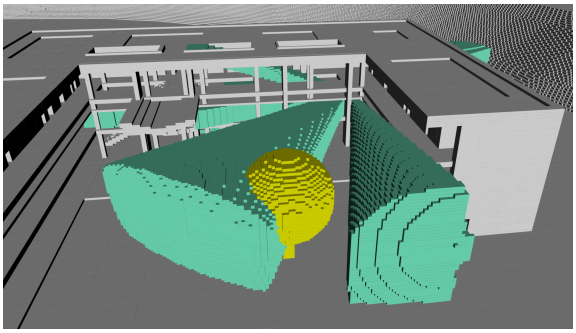
As shown in Figure 7a, we first create a spherical shell of voxels centered on the viewpoint and encompassing the target. It is created by (1) performing two unobstructed visibility from point operations with slightly different radii, which results in two solid spheres, then (2) subtracting the smaller solid sphere from the larger one. The resulting shell is used as an abstract "screen" on which to project "images" of the tree. The screen is several voxels thick to ensure that its relative thickness is similar in all directions.



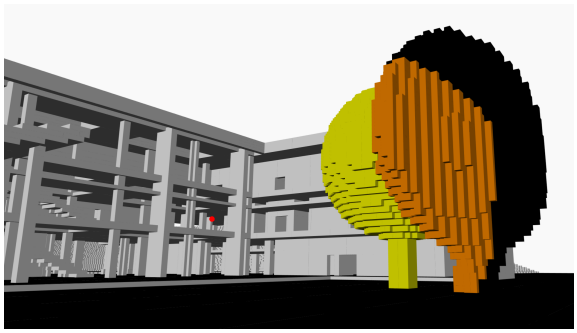
(a) Shell (orange) encompassing the target (yellow)



(b) View of the target, ignoring other obstacles



(c) View accounting for obstacles other than the target



(d) Projection of the visible fraction of the target

Figure 7: Steps in computing the visible fraction of a target.

Next we perform two additional visibility from point analyses. In the first analysis, shown in Figure 7b, the building voxels are excluded so that only the tree obstructs the view, In the second analysis, shown in Figure 7c, the tree voxels are excluded so that only the building obstructs the view. Taking the visibility results in Figure 7b and subtracting them from the shell in Figure 7a produces a projection of the tree, and intersecting this projection with the visibility results in Figure 7c yields the portion of the projected tree that is visible from the point in the building. These projections are shown in Figure 7d. The final metric is produced by simply counting the number of filled voxels in the "visible" portion of the projected tree, which is shown in orange in Figure 7d, and dividing by the total number of filled voxels in the projection of the tree. The result is that roughly 40% of tree is visible at the resolution shown.

The example is one of several included with the VASA package. It uses relatively coarse $25\times25\times25$ cm voxels to ensure interactive calculations on a variety of machines, but the voxel width can be decreased to 12.5, 6.25, or even 3.125 cm by adjusting an LOD slider. As the width decreases, the metric converges on around 60%, so it is important to understand that coarse-voxel results are highly approximate and intended only for prototyping new metrics or quickly ranking a diversity of design options. The example could be enhanced in a number of ways, such as evaluating viewpoints along a path instead of at a single location.

5 CONCLUSION

We propose a voxel-based approach to architectural space analysis in which efficiently encoded binary voxel models provide a common spatial representation supporting a variety of 3D grid-based algorithms. The approach was implemented and made available as a Dynamo package called VASA, which we use to demonstrate how voxel-based analyses and operations can be chained together to produce custom human-centric metrics for early stage design comparisons or generative design workflows. Voxel models and their associated algorithms may also serve as a foundation for multi-agent simulation platforms that incorporate time into the analysis of building performance.

We close with a few remarks about the name VASA, which was chosen as an abbreviation for Voxel-based Architectural Space Analysis but happens to also be the name of an infamous 17th century Swedish warship. The Vasa is known for being so heavily loaded with cannons and ornamentation that it sank on its maiden voyage before leaving the harbor, and has since become a metaphor for overly ambitious software projects that ultimately fail. Stroustrup (2018), the inventor of the C++ programming language in which the core of VASA was implemented, submits that the warship could have sailed given "a relatively modest increase of the Vasa's length and breadth", the lesson being that software must be built on a "solid foundation". We took this advice as a source of inspiration for our research, interpreting the common representation of space as a foundation that would have to support a comprehensive set of analyses.

REFERENCES

Aish, R., W. Jabi, S. Lannon, and N. M. Wardhana. 2018. "Topologic: Tools to Explore Architectural Topology". In *Proceedings of Advances in Architectural Geometry (AAG)*, pp. 316–341.

Danell, M., M. L. Ámundadóttir, and S. Rockcastle. 2020. "Evaluating Temporal and Spatial Light Exposure Profiles for Typical Building Occupants". In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (SimAUD)*.

Daniel, K., A. Nash, S. Koenig, and A. Felner. 2010. "Theta*: Any-Angle Path Planning on Grids". *Journal of Artificial Intelligence Research* vol. 39, pp. 553–579.

Danielsson, P.-E., and Q. Lin. 2003. "A Modified Fast Marching Method". In *Proceedings of the Scandinavian Conference on Image Analysis*, pp. 1154–1161.

Dijkstra, E. W. 1959. "A Note on Two Problems in Connexion with Graphs". *Numerische Mathematik* vol. 1, pp. 269–271.

Ferguson, D., and A. Stentz. 2006. "Using Interpolation to Improve Path Planning: The Field D* Algorithm". *Journal of Field Robotics* vol. 23 (2), pp. 79–101.

Fisher-Gewirtzman, D., A. Shashkov, and Y. Doytsher. 2013. "Voxel Based Volumetric Visibility Analysis of Urban Environments". *Survey Review* vol. 45 (333), pp. 451–461.

Ghosh, S. K. 2007. *Visibility Algorithms in the Plane*. New York, NY: Cambridge University Press.

Goldstein, R., S. Breslav, K. Walmsley, and A. Khan. 2020. "SpaceAnalysis: A Tool for Pathfinding, Visibility, and Acoustics Analyses in Generative Design Workflows". In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (SimAUD)*.

Goldstein, R., K. Walmsley, J. Bibliowicz, A. Tessier, S. Breslav, and A. Khan. 2022. "Path Counting for Grid-Based Navigation". *Journal of Artificial Intelligence Research* vol. 74, pp. 917–955.

Hart, P. E., N. J. Nilsson, and B. Raphael. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* vol. 4 (2), pp. 100–107.

Kao, C.-Y., and R. Tsai. 2008. "Properties of a Level Set Algorithm for the Visibility Problems". *Journal of Scientific Computing* vol. 35, pp. 170–191.

Li, F., S. Zlatanova, M. Koopman, X. Bai, and A. Diakité. 2018. "Universal Path Planning for an Indoor Drone". *Automation in Construction* vol. 95, pp. 275–283.

Lozano-Pérez, T., and M. A. Wesley. 1979. "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles". *Communications of the ACM* vol. 22 (10), pp. 560–570.

Nagy, D., D. Lau, J. Locke, J. Stoddart, L. Villaggi, R. Wang, D. Zhao, and D. Benjamin. 2017. "Project Discover: An Application of Generative Design for Architectural Space Planning". In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (SimAUD)*.

Pelechano, N., J. M. Allbeck, and N. I. Badler. 2008. *Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation)*. Morgan & Claypool Publishers.

Schaumann, D., S. Moon, M. Usman, R. Goldstein, S. Breslav, A. Khan, P. Faloutsos, and M. Kapadia. 2020. "JOIN: An Integrated Platform for Joint Simulation of Occupant-Building Interactions". *Architectural Science Review* vol. 63 (3-4), pp. 339–350.

Schwartz, M. 2021. "Human Centric Accessibility Graph for Environment Analysis". *Automation in Construction* vol. 127.

Schwarz, M., and H.-P. Seidel. 2010. "Fast Parallel Surface and Solid Voxelization on GPUs". *ACM Transactions on Graphics* vol. 29 (6).

Sethian, J. A. 1996. "A Fast Marching Level Set Method for Monotonically Advancing Fronts". *Proceedings of the National Academy of Sciences* vol. 93 (4), pp. 1591–1595.

Stroustrup, B. 2018. "Remember the Vasa!". Self-published paper accessed via www.stroustrup.com.

Sturtevant, N. R. 2019. "Choosing a Search Space Representation". In *Game AI Pro 360*, pp. 253–258. Boca Raton, FL: CRC Press.

Tsai, Y.-H. R., L.-T. Cheng, S. Osher, P. Burchard, and G. Sapiro. 2004. "Visibility and its Dynamics in a PDE Based Implicit Framework". *Journal of Computational Physics* vol. 199 (1), pp. 260–290.

Uras, T., and S. Koenig. 2015. "An Empirical Comparison of Any-Angle Path-Planning Algorithms". In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 206–210.

Valero-Gomez, A., J. V. Gomez, S. Garrido, and L. Moreno. 2013. "The Path to Efficiency: Fast Marching Method for Safer, More Efficient Mobile Robot Trajectories". *IEEE Robotics & Automation Magazine* vol. 20 (4), pp. 111–120.

AUTHOR BIOGRAPHIES

RHYS GOLDSTEIN is a Senior Principal Research Scientist at Autodesk Research who specializes in space analysis, visual programming, and multi-agent simulation for architectural design. His email address is rhys.goldstein@autodesk.com.

KEAN WALMSLEY is a Senior Manager and Software Architect at Autodesk Research who specializes in digital twins and generative design for the architecture, engineering, and construction (AEC) industry. His email address is kean.walmsley@autodesk.com.

NIGEL MORRIS is a Senior Manager and Senior Principal Research Scientist at Autodesk Research who specializes in generative design using high-performance computing, geometric algorithms, and optimization. His email address is nigel.morris@autodesk.com.

ALEXANDER TESSIER is the Director of Simulation, Optimization, and Systems at Autodesk Research. He specializes in interactive simulation, generative design, parallelism, hardware rendering, big data, computer vision, and digital twins. His email address is alex.tessier@autodesk.com.