Interpreting non-flat surfaces for walkability analysis

Mathew Schwartz[1] **and Subhajit Das**[2]
[1]New Jersey Institute of Technology, Newark, USA, cadop@umich.edu
[2]Georgia Tech, Atlanta, USA,das@gatech.edu

ABSTRACT

Through laser scanning, GIS data, new manufacturing methods, and complex designs, analysis of terrain in relation to human mobility is becoming ever more necessary. While standards for wheelchair ramps exist, they rarely show the entire picture, nor do they account for surface variation beyond a single axis. Although graph creation techniques in CAD exist for flat terrain, directional edge weights accounting for this variation are lacking. In this paper, a summary of research from both biomechanics and architecture in relation to surface walkability is presented, followed by a review of creation methods for a searchable graph representing an environment in CAD. A novel graph creation method that can respond to variations in surface height for walkability analysis is presented, where the edge weights of the graph are based on surface condition of parent-child height variations.

Author Keywords

human factors,walkability, computation, graph, analysis, architecture, space syntax, evaluation,urban design

ACM Classification Keywords

I.6 SIMULATION AND MODELING : [General]; J.5 ARTS AND HUMANITIES: [Architecture]; D.2.2 SOFTWARE ENGINEERING: Design Tools and Techniques—*Human Factors*; I.3.6 COMPUTER GRAPHICS: Methodology and Techniques—*Ergonomics*

1 INTRODUCTION

While acoustics, lighting, and thermal comforts are commonplace in building analysis, more individualized human factors are often left out. Two main challenges to integrating these individual factors are: the simulation and specific analysis from a human perspective, and the interpretation of a building or environment to run these metrics on. The latter issue is the focus of this paper, as the former is often the focus of biomechanics research through human subject studies and can be used as reference data.

To calculate walking distance and visibility in space, various academic approaches to the problem have been developed, largely revolving around line of sight and overall shortest path movement from different rooms. However, the manufacturing and traditional construction methods of the past kept the ground floor of a building flat. With newer technology and a look at the urban scale, this flat ground is not guaranteed. It is the goal of this paper to introduce an algorithmic approach to interpreting an unknown terrain or building environment in CAD as a searchable and directional graph whose nodes exist only in user-defined accessible terms. While the term walkability is used in various contexts, this paper focuses on the generation of the graph in which additional analysis can be generated.

1.1 Human Factors Based Evaluation

There is often a misunderstanding from the design perspective of the role of localized human factor analysis in buildings. While a building can be considered a structural frame or blank template that allow an occupant to freely move within, the layout, shape, and small details such as carpet type can have a profoundly larger impact than many realize. Furthermore, the lack of tools and analysis methods makes it nearly impossible for a designer to have the background knowledge and mental computation to make choices related to the human. Therefore, various building codes and standards have been developed to aid in the process, and bring some minimum standardization to the built environment.

In the case of accessibility, building codes such as the American Disabilities Act (ADA), while a good step towards securing a minimum standard, have also left many architects and designers assuming this standard is satisfactory. On the other hand, the ADA is often viewed as a hindrance to a building design, especially if the architect is unaware of the methodology behind a particular regulation. While at first this attitude toward the ADA may seem insensitive, this view is not entirely unwarranted, as the ADA is based on a prescriptive code. In fact, just three years after the ADA was enacted, researchers in architecture had developed computer-aided tools for interpreting the environment based on people, specifically pointing out the lack of quantitative methods for assessing a design in relation to the goals of the ADA [11]. Examples of standards lacking justification through research are too common for designers to blindly follow. In the case of stair width,

SimAUD 2019 April 07-09 Atlanta, Georgia

it had been defined as 44 inches for 2 files of people; however, no evidence had been used for arriving at this number. Likewise, assumptions have been made, such as an increment of 6 inches is too small to impact flow, that turn out to be incorrect [20].

In a comparison of perception of design elements between designers and medical staff, designers were more likely to focus on psychological elements such as views of nature or color, while the medical staff focused on physical health such as handrails and safety bars [10]. Part of this discrepancy may be due to the approaches and interactions each discipline has with the built environment, as designers may view the built environment on a macro scale, while the medical staff are inside, with localized interactions. The straightforward solution to this is through simulation and analysis; Design tools can alert and inform about these localized interactions, just as lighting and thermal analysis tools do now.

Notably, the increased use of machine learning in nearly every discipline has inevitably found its way into the design of the built environment. While the implementation details and results are out of scope for this paper, the approach taken and key observations are important. Specifically, in order to generate metrics to be optimized, [29] reviewed numerous papers and strategies of design in relation to the human, pointing out interior design metrics such as "a television should maintain a certain distance from the normal viewing area...The width of a pathway should depend on the habitant's body width..." [29]. While these may seem more or less obvious to a designer, the access to these metrics in CAD tools is still missing.

This paper focuses on the physical attributes to traversing a space through human mobility. In considering the walkability of a non-flat surface, the types of surfaces for human traversal can be characterized as:

- Flat (leveled floor)
- Ramp (slope along the progression axis)
- Cross-slope (slope perpendicular to the progression axis)
- Staggered (stairs)
- Uneven (variations in curvature, natural topography)
- Uneven staggered (faceted surfaces, bricks in walkway)

The items in this list, with even minor variations between them, have a profoundly different impact on people. Each of these items can be studied and referenced in the biomechanical literature. Furthermore the impact of these surface conditions are not always obvious. In the case of a staggered surface (most commonly presented as stairs), recent stair-climbing studies on the relationship between the rise and run of a step and the probability of a fall [19] have had an impact on architecture by leading to a 2015 change in building code for stairs [24]. Of importance for simulation and evaluation, the building code value of stair tread is not an absolute cut-off to fall probability and should still be evaluated on a case-by-case basis.

In non-flat ground condition studies through an instrumented treadmill, 62% increase in hip work was found, with an overall increase of 28% in net metabolic energy expenditure [27]. In the case of cross-slope walking, the asymmetrical movement required may lead to falls [6]. On an even more specific ground condition, small variations in brick height of walkways, often caused by weather over time, were found to cause a person to lower their center of mass for additional stability, while also increasing flexion in some joints, likely leading to a higher energy expenditure [5].

Walk-ability on various ground conditions at a large scale, such as within an urban environment, provide an improved metric to assess the space. In [28], the authors look at urban analysis in various ways, including comfort and mobility, with implications towards new zoning rules and regulations. Metrics such as mobility are directly impacted, as seen in the literature, by the ground condition, and likewise, so is comfort. In [16], various design metrics are outlined as having opportunity to be calculated with modern tools, including *Adjacency preference* and *Buzz*. Likewise, the accuracy of these metrics would be impacted by varying ground conditions, making it harder to reach a certain location, or causing changes in circulation speed at various locations.

An indoor walkability index (IWI) was defined as "a measure on which a path has good performance for pedestrians in the building" [13]. In particular, [13] describes three factors that make up the assessment: *Distance*, *Accessibility*, and *Pedestrian-friendly* for evaluating indoor walkability. However, surface quality, which through biomechanics research (stated above) has the largest impact on walkability, is not accounted for.

1.2 Graph Generation

The generation of a graph that can be used to evaluate metrics of a building has been approached in multiple ways within architecture research, largely revolving around the specific metrics in focus. Early work in this area revolved around the concept of an isovist,or visible points in space [1]. Following this intention, the graph constructed in the building was based on, and held information to, the visibility within the space [25]. Likewise, analysis of circulation throughout an entire building has been demonstrated by [12], referred to as the Universal Circulation Network (UCN). Rather than a dense graph of adjacent nodes, the UCN leverages the data structures within BIM to build a graph based on visible paths and shortest distances.

The visibility graph defined in [25] uses a combination of even grid projections and vertex intersections. The authors note the intention and possible use for considering the vertices as potentially occupiable spaces. While the idea of a permeability graph, in which a visibility graph is constructed at floor level, could be used for obstacle detection and accessibility, it is not demonstrated or explicit. A problem in using the visibility graph in [25] for accessibility is the distribution of an even grid in plan at eye-level. In particular, the need to connect staircases to separate graphs on each floor illustrates the single dimensionality of the approach. Given a ramp or slope ground condition, the visibility graph plane would intersect, not completing the series of connections described.

Implementing 3D Isovists, [21] projects rays spherically at is what is referred to as an *observation point*, although the authors do not detail how these points are decided. The use of these spherical projections is to essentially map visible depths throughout a space, similar to Lidar on an autonomous robot, and classify the space based on feature extractions. The isovist in this case, while analogous to vision through the possible lines of sight dictated by the rays, is not employing a method of human factor analysis, but rather a tool for parsing and understanding the environment. In the case of binocular vision, [9] analyzes the human view from nodes placed through possible sitting locations. The nodes in the graph are populated along a curve using user input parameters.

A common technique in generating a building graph is the distribution of a rectangular and equally spaced graph across a floorplan. This may be done either through a generation of points, or a method employing ray tracing. This method, similar to [25], comes with important drawbacks. First, for cases in which the floor plan is shaped as an L or U, a large number of unused points are generated and need to be dealt with through wasted computation. Second, the planar aspect limits the ability to interpret uneven surfaces. Finally, edge connections may be generated in inaccessible locations within the environment. A method for reducing the node/edge connections is culling, as demonstrated in [17], where the intersection between an object and the line connected from two nodes invalidates the edge. Alternative strategies have followed the implementation of the UCN, with navigational boundaries dictated by the convex points of objects projected onto the plane [7]. In [8], a voxel based approach in which the entire environment is discretized, with each voxel containing additional properties for use in a large variety of evaluations.

Graph search for navigation

Graph search for spatial navigation is a well-established technique in path finding problems in the domains of automation and planning, robotics and game development. An early work in this area by Botea et al. showed a hierarchical path finding method to find optimal paths on grid based maps. Following a clustered map approach, they showed the hierarchy could be extended to more than two levels [2]. In the context of Architectural planning and human behavior simulation, the work of Chu et al. shows a way to sense the vicinity of the physical obstacles within a visible space to simulate the influence of social behavior on evacuation. They discretized the continuous 2D space into square cells forming a 2D grid, which is further connected via edges linking visible navigation points [3]. Turner et al. showed their technique can improve human behavioral response using artificial evolution of existing navigation rules Their technique proves that human's guidance mechanism does not depend on the spatial properties acting on their direct perception [26].

Path-finding in sloped-terrains

A smoothest path through a sloped surface or terrain is a topic of research within gaming and robotics which overlaps with the goals presented here. Roles et al. showed a novel technique to compute the smoothest path through a sloped terrain [22]. Based on Dijkstras algorithm, their technique optimized distance and slope to retrieve the least rigorous path between two queried points A and B on sloped terrain. These pointts were chosen from a set of points V which are retrieved from the input mesh geometry. They hypothesized that a minimally sloped path would be most desirable over terrain or sloped land surface for various reasons, i.e., in hospital, etc. Their algorithm is based on a node, edge graph system, where a starting point A appends all the adjacent vertices or nodes nearest to A by a threshold distance. Then they compute a set m where vertices of least weights to each explored nodes in the graph are added. This is repeated till the ending vertex or point B is reached. Finally, the smoothest path is retrieved starting from the ending point B to starting point A. The weights of each node are computed using traditional shortest path method, i.e., the sum of between two nodes and the distance already traveled.

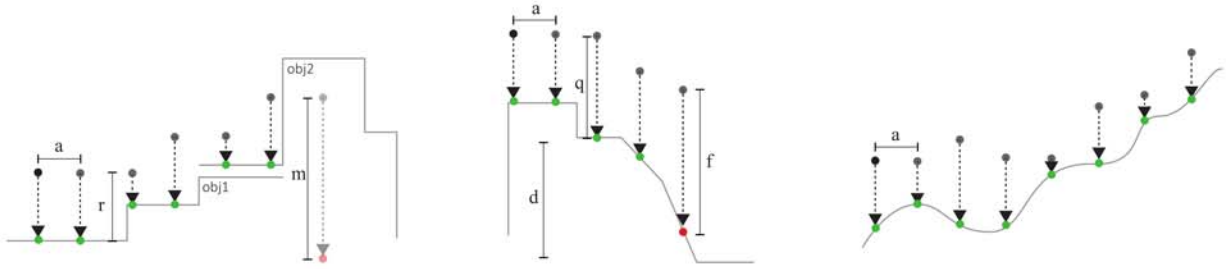
Liu et al. studied and implemented slope constraint for terrain surfaces. Further, their method involved intelligent surface simplifications in searching for shortest paths using the input slope constraint efficiently. They described the technique of surface simplification to reduce the complexity of finding the shortest path [14]. One intuitive benefit of surface simplification is gain in speed to compute the shortest path (i.e., less number of nodes, thus fewer computations). However, one key challenge is the slope of the simplified surface might not satisfy the slope constraints of the original surface. Even if the slope constraints are satisfied, it might be longer than the shortest smooth path discovered on the original surface. To address the second challenge, the authors introduced a distance requirement in addition to the slope requirement while searching for the shortest path. Our technique however differs by deploying a novel technique of graph creation on an input 3D geometry representing any architectural or urban space.

Agents on terrain

While human behavior analysis in terms of architectural spaces often benefits from character path detection for evacuation, shortest paths for nurses to patient beds, etc., these techniques help in simulating critical scenarios like natural disasters. Beyond the large scope of agent based simulation, the control methods for an agent on a surface are relevant to this work.

One common method is to use physics-based environment controls so that a character (e.g., occupant or agent) remains on the ground due to physics-based constraints or the character automatically knows how to orient oneself to stabilize its posture naturally given the dynamically changing neighboring terrain and environmental condition. Further, navigational paths and accessibility within the environment are then decided on-the-fly based on the characters interaction, position, and orientation (e.g., [15]). As this works on a single and relative location, it is not a suitable technique for complete building analysis.

In general, the following assumption can be made based on prior literature: a dense graph containing nodes only in human accessible locations can be used for a large span of evaluative metrics. Albeit through an IWI, a buzz metric, or visibility maps, resolution of the graph with directional edges is advantageous. The drawback to the dense graph approach is



(a) Ray misses a surface (b) Node position outside limit (c) Graph continues by relative heights

Figure 1: Illustration of three common occurrences in graph creation of non-flat surfaces. Green circles represent a valid node and red circle as invalid based on the parameters given as variables.

computational time in generating and searching through. In some cases, the computational time is reduced through the selective node/edge pair generation. In the case of search times, various methods in computer science have addressed the issue through a variety of datastructure reduction techniques, suggesting a dense graph could take the form of a superset composed of multiple optimized subsets for a particular metric. Importantly, the sparsity of the graph cannot be so that variations in ground conditions that would impact mobility are missed.

2 METHODOLOGY

In this section we define an accessibility graph, where all nodes are possible locations of an occupant in an environment, given user-specified parameters (e.g., a 12:1 slope). The graph itself is directed, and nodes consist of a parent-child relationship. The algorithm for generating the graph, and the method for implementing the system in the Grasshopper environment of Rhino3D are shown.

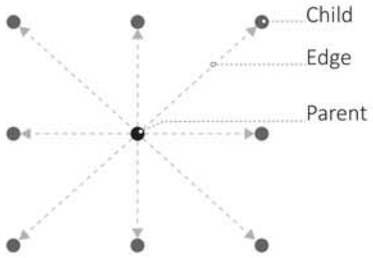


Figure 2: The parent-child relationship in the (xy) plane.

2.1 Graph Creation

Given a known possible occupant location in the environment, possibly the sidewalk of an urban space or lobby in a building, the other possible locations of an occupant can be found by using ray casting and a queue, such as in [23]. There are two main components to the directed graph $G = \{N, E\}$ with N nodes and E edges. In concrete terms, the set N represents the accessible locations of an environment for a person given a specific starting location, while E is the set of costs between these locations. As the graph is directed, a cost value $e_{i,j}$, where $e \in E$, corresponds to an ordered pair (n_i, n_j) where $n \in N$. The ordered pair of nodes (n_i, n_j) is referred to in this paper as the parent-child relation (Fig. 2), where n_i is the parent and n_j is the child, with the corresponding cost $e_{i,j}$ applied from the parent to the child. The cost can be calculated both at the time the node pair is created or after the construction of N is complete, as described later in this section.

The algorithm to build the graph using recursion can be seen in Algorithm 1 (can be implemented with a loop as well). The given start location initializes the first parent node p. The parent set P is derived from the first node n_i in the parent-child relation (n_i, n_j) contained in $P = \{n_i | \forall (n_i, n_j) \in N\}$. During each iteration of the algorithm, the given parent node is checked for valid children. If a child is valid, a parent-child relation is created and an edge cost is assigned to that ordered pair. The cost value is defined in `setEdgeCost(p,c)`, where a typical definition of the edge cost for the ordered pair may be Euclidean distance. However, weighted parameters for various surface conditions can be used as well (see Sec. Edges). Finally, the children that are not in the parent set are then added to the queue Q and passed to the `buildGraph(G,Q)` function until all possible parents and children have been evaluated, and $Q = \emptyset$.

Algorithm 1: Build directed graph of nodes and edges

$N, E \leftarrow \emptyset, \emptyset$
$G \leftarrow \{N, E\}$
$Q[0] \leftarrow$start location
Function `buildGraph(`G, Q`)`**:**
  $p \leftarrow Q.pop()$
  $C \leftarrow$`getNodes(`p`)`
  for $c \in C$ **do**
    $e_{p,c} \leftarrow$ `setEdgeCost(`p, c`)`
    $N \ni (p, c)$
    $E \ni e_{p,c}$
    if $c \notin P$ **then**
      $Q \ni c$
  return `buildGraph(`G,Q`)`

Nodes

While N defines the edges used in search algorithms, N itself is useful as well. In particular, the set N can be tessellated to

generate a valid walkable surface, and used in ways similar to that detailed in Section Introduction. Furthermore, this set can be used in data structures to define additional parameters and qualities of specific locations throughout the environment; albeit acoustic, lighting, and view-ability, reach, or fall probability (e.g., [23]) in which a building graph for circulation (e.g., [12]) alone does not provide the resolution for.

An important contribution of this paper is the extension of the node creation protocol from [23] to include height variations when generating the graph. Similarly, the number of nodes in the set N increases as each valid node location is used to check for additional valid nodes in immediate proximity (i.e., finding the child nodes). While the initial configuration of the parent-child relation is the same in the (xy) plane (Fig. 2), modifications to the inclusion of a child node were made to extend the graph creation to non-flat surfaces. To illustrate how the graph creation accounts for non-flat surfaces when generating nodes, Figure 1 shows the node evaluation in the (xz) plane where the ray cast direction is $-\hat{z}$, and as the graph is in relation to physical space, $-\hat{z}$ corresponds to the direction of gravity. The surfaces illustrated provide examples of a step, faceted surface, and natural topography (e.g., a hill or mountain). Each sub-figure uses the same parameters but illustrates various situations in which a node can be valid or invalid. The variables used in this figure are also used in the equations and algorithms. In Figure 1a a is the parent-child offset in the (xy) plane, r is the set height increment in $\hat{z}$ from parent to child, $m = \infty$, such that a ray that does not hit a surface. In Figure 1b d is the allowable height variation from a parent in $-\hat{z}$ such that $q \leq d$ and $f > d$, resulting in an invalid node (red). When a possible node is invalid, it is not added to the possible parents queue Q, and in the simple example within the figure, the graph completes. To further illustrate the relative positioning of parent-child nodes, the vertical increase shown in Figure 1c demonstrates how the height offset r is relative to the previous node (when viewing left to right). Likewise, for each valid (green) node, the node to the left would be the parent.

The explicit definition of the valid nodes is given in Eq. 1.

$$N = \{n_i | n_i \in (n_i, n_j) | (n_i, n_j) \in E\} \tag{1}$$

After defining a starting location, Algorithm 1 tests the start as the initial parent node by calling the function `getNodes(p)` to check for possible children. This function is shown Algorithm 2.

The parent node p passed to `getNodes(p)` is checked for possible children by initializing locations in the (xy) plane in eight directions of a bounding square with a length of $2a$ (visualized in Fig. 2). The z component of the parent node is then added with the height threshold r. The possible child is then passed to a function `getChild(c)` that checks a ray for intersection with the nearest surface or geometry using the start location c and direction $-\hat{z}$. If the intersection is outside the defined criteria for a valid node (i.e., intersection distance is $> d$), or there is no intersection found, the function returns

Algorithm 2: Check for valid child nodes

Function `getNodes(p)`**:**
`// Check parent p for valid children c`

for $i \leftarrow -1$ **to** 1 **do**
  for $j \leftarrow -1$ **to** 1 **do**
    c $\leftarrow (p.x + (i \times a), p.y + (j \times a), p.z + r)$
    $c \leftarrow$`getChild(c)`
    if $c \neq$ *false* **then**
      $c \in C$
return C

false. If the intersection matches the criteria for a valid node, the intersection location is returned.

After checking for valid child nodes,`getNodes(p)` returns the child set C, where $c|c_i \in C$. Given at least one valid child in `buildGraph(G,Q)` of Algorithm 1, such that $C \neq \emptyset$,the parent is added to the graph with a directional edge to its child node(s). The child is then added to the queue if it is not already a parent in the graph.

As established in [23], the use of ray tracing to build the graph has a specific advantage when it comes to unknown geometry. Illustrated in 1a, the ray intersection allows the graph to be indifferent to the construction of object geometry. Given two objects $obj1$ and $obj2$, the geometry of the two can overlap. Albeit from incorrect modeling or through issues with automated surfacing. More common may be the alignment of two different polygons or surfaces at the edge. While mesh planarization and algorithms used in various fields for understanding slopes in Cartesian space exist, the recognition of where one object ends and another begins can be ambiguous and only important in relation to the accessibility of the space. Simple additions in any CAD program, such as objects contained on different layers, can provide additional refinement to the building graph creation.

Edges

The final graph is composed of ordered pairs of nodes that correspond to an edge with a cost value. A parent-child relation for nodes n_i and n_j is valid given in Eq. 2. We define the vector $\vec{V}_i$ as parent and $\vec{V}_j$ child x, y, z positions, where $\vec{V}_{i_{xy}} = \langle n_{i_x}, n_{i_y}, 0 \rangle$ and $\vec{V}_{j_{xy}} = \langle n_{j_x}, n_{j_y}, 0 \rangle$.

$$\begin{aligned} E = \{&(n_i, n_j) = \gamma | (\delta = a \vee \delta = \sqrt{2a}) \wedge \\ &((0 < \beta \leq r) \vee (0 > \beta \leq d) | \beta = (n_{i_z} - n_{j_z}))\} \end{aligned} \tag{2}$$

Where δ is $\|\vec{V}_{i_{xy}} - \vec{V}_{j_{xy}}\|$, γ is the calculated edge cost, a is the spacing factor between nodes in (xy), r is the threshold value for $\hat{z}$, and d is the threshold for $-\hat{z}$, corresponding to Fig. 1.

The cost for moving from a parent to child node is defined by the edge value. While this value is ambiguously defined in this paper through `setEdgeCost(p,c)` in Algorithm 1 for each child c at a time, it can be modified at any point after

the graph is generated. In the simplest implementation, the cost e of a parent-child pair is the euclidean distance. This can often be a safe metric to use within the scope of the built environment, especially when the set of nodes has already accounted for accessibility. However, as the node relationships are stored and locations are known, the cost function can include the slope from parent to child, a similar technique used in [22, 14], making upward and downward transitions weighted differently. As multiple directions are considered in the set of children for a given parent node, cross-slope information can also be used in the cost function. Furthermore, using the relation between all children to a given parent, scoring methods for a node can be applied, such as chemical diffusion rates for agent modeling shown in [18]. It is this fine-grained graph that affords integrating the human based metrics described in 1.1. At the urban scale, the inclusion of varying ground conditions to walkability analysis that can better predict comfort and fatigue can also be realized.

2.2 Data and Implementation

The algorithms described above were implemented in the Grasshopper environment of Rhino 3D software. For user control, the offsets and various parameters were implemented with the UI elements, while the majority of code was written in python using the *GhPython* component and interfacing with the Rhino Python API. The red nodes and green line for visualization rely on the drawing elements of Grasshopper in Rhino.

The graph is stored in a Python dictionary consisting of parent-child relationships. The average lookup time complexity of $\mathcal{O}(1)$ for the dictionary provides an efficient method for interacting with the dense and high-fidelity graph. The algorithm was initially and conceptually recursive, however default limits in python made implementation with a *while* loop more robust.

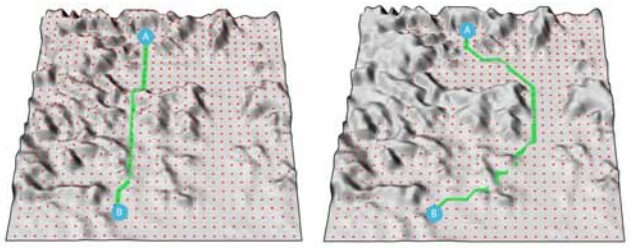
While there have been projects for expanding the Rhino ironpython scope [30, 4], they have specialized scripts and installations not included in the standard python library. Due to the simplicity, a Socket based communication was implemented to communicate with a machine-local python instance with the SciPy library. Using the $scipy.sparse.csgraph.shortest\_path$ method, any search algorithm type available can be used and applied to the graph. In the examples within this paper, Dijkstras search algorithm was used.

3 RESULTS AND DISCUSSION

3.1 Uneven

As a demonstration of the graph's ability to include only accessible spaces, we first look at Fig 3a, in which an entire topology is included in the graph. This case may be prevalent when using GIS data and/or Lidar scans of a topology for a site-specific study. Important to note, this is not from a planar grid above, but rather from the start point A. In the search, A to B is clearly defined as the shortest path outlined in green. The reason the resulting graph is complete across the surface is that there is no threshold that limits the height variation between a parent and child, meaning all nodes are connected. However, in the case that this topology represents a mountain or urban environment, it is unlikely a user would be able to follow this path as the height variation between a parent-child node relation is excessive for a reasonable person to be able to access.

Beyond accessibility standards or guidelines, the continually changing terrain makes the shortest path by distance an unlikely representation of a path a user would follow. If all locations of the topology are considered accessible, the notion of the shortest path can be modified to account for energy requirements of the terrain. For example, a cost function of the slope can be associated to an edge, whereby the shortest path would be not based on distance alone. However, if certain limits are placed on the ability to traverse a terrain, or within some guideline, parameters of the graph creation can be set to include only those spaces.

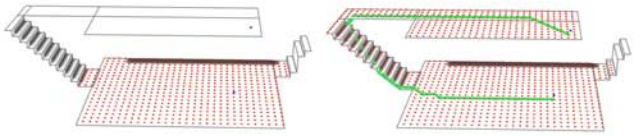


(a) Graph without threshold (b) Graph with threshold

Figure 3: A topology with nodes generated from A, parsing the entire surface. The green line is the search algorithm finding the shortest path from A to B. In 3a, the threshold for parent-child height variation is larger than the terrain while in 3b the lower threshold changes the graph

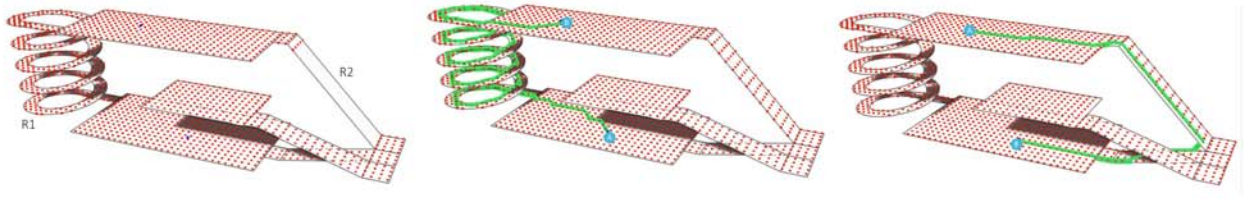
Fig 3b demonstrates the user-centric graph in which a height variation limiter can be used. The resulting graph is the accessibility graph for that user parameter, and the same points A and B are used for the search algorithm, which now follows a more level and easy terrain. Combined with the options to modify the cost described previously, this graph search can be used for city or urban planning to find the least costly, or most efficient, method of terrain removal for human comfort and accessibility.

3.2 Staggered



(a) Stairs excluded from graph. (b) Stairs included in graph

Figure 4: Nodes of the graph visualized in red, with the calculated path shown in green. (4a) A limiting height variable preventing the graph from including stairs. (4b) A limiting height variable that allows the inclusion of short steps on the left, but not the taller steps on the right.



(a) Threshold preventing surface inclusion (b) Search starting on bottom to top (c) Search starting on top to bottom

Figure 5: Nodes of the graph visualized in red, with the calculated path shown in green. The two main ramps are labeled as $R1$ and $R2$ (5a) Equal thresholds prevent the graph from including nodes on *R2*. (5b) Nodes are on all surfaces, while the shortest distance from A to B uses *R1* rather than *R2* due to the directional graph. (5c) The shortest path is taken from the top surface to the bottom surface, utilizing *R2*.

In the case of stairs, an important value is in the elimination of steps from a building graph, just as much as it is important to be able to include them. In particular, a building occupant in a wheelchair would not be able to traverse steps. In this case Fig 4a demonstrates a second floor surface that is not included in the building graph.

Conversely, Fig 4b demonstrates a modification in the height variable such that the graph automatically can traverse the steps and include the second floor. This traversal is relative, as described in the methodology, and can be seen by the still too large right sided steps. While these steps are in the world coordinate system, lower than the steps and second floor plane, they are not included based on the height offset.

3.3 Ramp and Cross-slope

As a final complex demonstration, Figure 5 shows three applicable cases to the height variation and directional graph. In the first example, the graph is built with a height offset setting equal when moving up and down. Within this threshold, all but the ramp *R2* is covered in nodes. As the ramp *R1* is set at a smaller incline, the top surface can be included in the graph.

Next is a case in which nodes cover all surfaces. However, node locations alone do not provide enough information about the environment. In this case, the tolerance for an edge to be created with the child node being higher than the parent is smaller than the tolerance for an edge to be created with the child node being lower than the parent. This distinction can be seen by the shortest path algorithm applied to the nodes A and B, where A is on the bottom plane. The shortest path uses the *R1* ramp, while physical distance between nodes is shorter with the *R2* ramp. This relationship can be further understood by using the same graph but inverting the location of A and B. When starting from the top surface the shortest path uses the *R2* ramp.

While both up and downhill walking create additional load on the human body, there are many instances in which one direction can still accommodate for a comfortable and safe path. For example, going downhill may pose additional risks for falling or balance compared to moving uphill. In this example, multiple ramps at various slopes are used for simplicity and clarity of the overall system. However, the linear ramps (*R2*) could easily be replaced by stairs, such as in Fig. 4, and then demonstrate that it is not accessible by an occupant with a wheelchair (e.g., in Fig. 5a).

3.4 Discussion

In this paper, we present a method for building a directional graph that can interpret non-flat ground conditions. The literature and approach are focused on the physical act of mobility without regard for social or economic factors. However, the resulting graph contains nodes that, when associated with locations of interest, additional metrics can be incorporated. Through thresholds in the algorithm, variations in the ground condition can prevent the graph from including certain spaces, allowing the designer to see which parts are too extreme or inaccessible for a given occupant. Additionally, this method allows for the graph creation to account for staircases without explicit reference, staying true to the ability to interpret unknown geometries in CAD. The algorithm presented and the method for creation provides a platform for layering complex and human-based analysis methods at both the building and urban scale with high-fidelity and dense node creation.

REFERENCES

1. Benedikt, M. L. To take hold of space: isovists and isovist fields. *Environment and Planning B: Planning and design 6*, 1 (1979), 47–65.
2. Botea, A., Mller, M., and Schaeffer, J. Near optimal hierarchical path-finding. *Journal of Game Development 1* (2004), 7–28.
3. Chu, M. L., Parigi, P., Law, K., and Latombe, J.-C. Safegress: a flexible platform to study the effect of human and social behaviors on egress performance. In *Proceedings of the Symposium on Simulation for Architecture & Urban Design*, Society for Computer Simulation International (2014), 4.
4. Digital-Structures. Regular python from ironpython in rhino/grasshopper. `https://github.com/Digital-Structures/ghpythonremote`, 2017. [Online; accessed 26-Nov-2018].
5. Dixon, P. C., Jacobs, J. V., Dennerlein, J. T., and Schiffman, J. M. Late-cueing of gait tasks on an uneven brick surface impacts coordination and center of mass control in older adults. *Gait & posture 65* (2018), 143–148.

6. Dixon, P. C., and Pearsall, D. J. Gait dynamics on a cross-slope walking surface. *Journal of Applied Biomechanics 26*, 1 (2010), 17–25.
7. Doherty, B., Rumery, D., Barnes, B., and Zhou, B. A spatial query & analysis tool for architects. In *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, Society for Computer Simulation International (2012), 4.
8. Goldstein, R., Breslav, S., and Khan, A. Towards voxel-based algorithms for building performance simulation. In *Proceedings of the IBPSA-Canada eSim Conference* (2014).
9. Hudson, R., and Westlake, M. Simulating human visual experience in stadiums. In *Proceedings of the Symposium on Simulation for Architecture & Urban Design*, Society for Computer Simulation International (2015), 164–171.
10. Kim, D., Lee, J. H., and Ha, M. Exploring perceptions of designers and medical staff in south korea about design elements for the elder-friendly hospital. *Journal of Interior Design 39*, 4 (2014), 15–32.
11. Lantrip, D. B. Environmental constraint of human movement: A new computer-aided approach. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 37, SAGE Publications Sage CA: Los Angeles, CA (1993), 1043–1043.
12. Lee, J.-k., Eastman, C. M., Lee, J., Kannala, M., and Jeong, Y.-s. Computing walking distances within buildings using the universal circulation network. *Environment and Planning B: Planning and Design 37*, 4 (2010), 628–645.
13. Lee, J.-K., Shin, J., and Lee, Y. Circulation analysis of design alternatives for elderly housing unit allocation using building information modelling-enabled indoor walkability index. *Indoor and Built Environment* (2018), 1420326X18763892.
14. Liu, L., and Wong, R. C.-W. Finding shortest path on land surface. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, ACM (New York, NY, USA, 2011), 433–444.
15. Liu, L., Yin, K., van de Panne, M., and Guo, B. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 154.
16. Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., and Benjamin, D. Project discover: An application of generative design for architectural space planning. In *SimAUD 2017 Conference proceedings: Symposium on Simulation for Architecture and Urban Design* (2017).
17. Nagy, D., Villaggi, L., Stoddart, J., and Benjamin, D. The buzz metric: A graph-based method for quantifying productive congestion in generative space planning for architecture. *Technology— Architecture+ Design 1*, 2 (2017), 186–195.
18. Narahara, T. *Self-organizing Computation A Framework for Generative Approaches to Architectural Design*. Harvard University, 2010.
19. Novak, A. C., Reid, S. M., Costigan, P. A., and Brouwer, B. Stair negotiation alters stability in older adults. *Lower Extremity Review 2*, 10 (2010), 47–51.
20. Pauls, J. L., Fruin, J. J., and Zupan, J. M. Minimum stair width for evacuation, overtaking movement and counterflowtechnical bases and suggestions for the past, present and future. In *Pedestrian and evacuation dynamics 2005*. Springer, 2007, 57–69.
21. Peng, W., Zhang, F., and Nagakura, T. Machines perception of space: Employing 3d isovist methods and a convolutional neural network in architectural space classification.
22. Roles, J. A., and ElAarag, H. A smoothest path algorithm and its visualization tool. In *2013 Proceedings of IEEE Southeastcon* (April 2013), 1–6.
23. Schwartz, M., Azeez, A., and Patel, K. Human task and disability based automated evaluation of space and design in cad. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, SIMAUD '18, Society for Computer Simulation International (San Diego, CA, USA, 2018), 18:1–18:8.
24. Senn, M. *TORONTO REHAB RESEARCHERS HELP CHANGE BUILDING CODE FOR STAIRS*, 2015 (accessed November 25, 2018). `https://www.uhn.ca/corporate/News/Pages/toronto_rehab_researchers_help_change_building_code.aspx`.
25. Turner, A., Doxa, M., O'sullivan, D., and Penn, A. From isovists to visibility graphs: a methodology for the analysis of architectural space. *Environment and Planning B: Planning and design 28*, 1 (2001), 103–121.
26. Turner, A., and Penn, A. Evolving direct perception models of human behavior in building systems. In *Pedestrian and Evacuation Dynamics 2005*. Springer, 2007, 411–422.
27. Voloshina, A. S., Kuo, A. D., Daley, M. A., and Ferris, D. P. Biomechanics and energetics of walking on uneven terrain. *Journal of Experimental Biology* (2013), jeb–081711.
28. Wilson, L., Danforth, J., Harvey, D., and LiCalzi, N. Quantifying the urban experience: Establishing criteria for performance based zoning. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (simAUD 2018)*, simAUD (2018).
29. Yu, L. F., Yeung, S. K., Tang, C. K., Terzopoulos, D., Chan, T. F., and Osher, S. J. Make it home: automatic optimization of furniture arrangement.
30. Zurich, B. R. G. E. Compas. `https://compas-dev.github.io/main/tutorial/xfuncs.html/`. [Online; accessed 26-Nov-2018].