IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 23, 2020, accepted November 17, 2020, date of publication November 30, 2020,
date of current version December 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3041276

Quantifiable Isovist and Graph-Based Measures for Automatic Evaluation of Different Area Types in Virtual Terrain Generation

ANDREW PECH, CHIOU PENG LAM, AND MARTIN MASEK, (Member, IEEE)
School of Science, Edith Cowan University, Perth, WA 6027, Australia

Corresponding author: Martin Masek (m.masek@ecu.edu.au)

This work was supported by an Australian Government Australian Postgraduate Award Scholarship.

ABSTRACT This article describes a set of proposed measures for characterizing areas within a virtual terrain in terms of their attributes and their relationships with other areas for incorporating game designers' intent in gameplay requirement-based terrain generation. Examples of such gameplay elements include vantage point, strongholds, chokepoints and hidden areas. Our measures are constructed on characteristics of an isovist, that is, the volume of visible space at a local area and the connectivity of areas within the terrain. The calculation of these measures is detailed, in particular we introduce two new ways to accurately and efficiently calculate the 3D isovist volume. Unlike previous research that has mainly focused on aesthetic-based terrain generation, the proposed measures address a gap in gameplay requirement-based terrain generation – the need for a flexible mechanism to automatically parameterise specified areas and their associated relationships, capturing semantic knowledge relating to high level user intent associated with specific gameplay elements within the virtual terrain. We demonstrate applications of using the measures in an evolutionary process to automatically generate terrains that include specific gameplay elements as defined by a game designer. This is significant as this shows that the measures can characterize different gameplay elements and allow gameplay elements consistent with the designers' intents to be generated and positioned in a virtual terrain without the need to specify low-level details at a model or logic level, hence leading to higher productivity and lower cost.

INDEX TERMS Gameplay elements, terrain generation, level design.

I. INTRODUCTION

Terrain, the physical shape of a piece of ground, has a major effect on how that ground can be used and interacted with. In computer games set in a 3D outdoor environment, terrain is a key means of implementing gameplay design elements that shape player experience. Some examples of gameplay design elements, identified by Hullett and Whitehead [1] for first person shooter levels, include sniper locations, arenas and choke points. In an outdoor 3D game, these patterns can be achieved through appropriate shaping of the game terrain.

The process of game terrain creation typically involves level designers working with environmental artists to achieve terrains with the required aesthetics and gameplay considerations. Existing work in automating this process has mainly focused on generation of aesthetic terrains rather than incorporating gameplay elements, as these are hard to quantify. The development of measures that could be used to indicate how suitable an area of terrain is for a gameplay style is a significant goal. Such measures would give a person creating the terrain from a design brief an objective assessment of how their terrain meets the brief. In addition, the measures could be employed in approaches that seek to automatically generate a terrain incorporating specific gameplay elements.

In this article, we extend our work from [2] where a set of measures was introduced to capture the characteristics of gameplay elements of a terrain. These measures focus on two aspects of a terrain, namely: the properties of a local area within the terrain, and the relationships between local areas within the entire terrain. To characterize local areas, we look to the field of space syntax [3] and the properties of an isovist [4]. To examine relationships between the set of areas on a terrain we utilize measures from graph theory. In [2], we showed that these measures can be used to categorize

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

gameplay elements associated with an existing set of game terrains. The contribution of this article is to provide details of the algorithms to compute the measures, provide results of using the measures to drive an evolutionary process that generates terrain based on gameplay requirements, as well as provide an expanded review of existing terrain generation techniques. In particular, we introduce and evaluate two novel ways of calculating the volume of a 3D isovist that is more efficient than existing techniques.

The remainder of this article is organized as follows: Section II serves as an introduction into the field of procedural content generation, focusing on terrain. Also, as our work introduces new efficient algorithms of computing the isovist volume, a review of existing methods for such computation is presented. Section III details the measures we use for characterizing gameplay elements in a terrain and algorithms for their calculation, including two new isovist volume estimation methods we have introduced. Section IV has three sections that describe our evaluations: A) using the proposed measures to classify area types, B) comparing the introduced isovist volume estimation methods to existing isovist volume estimation methods, and C) using the proposed measures in an evolutionary approach towards generating terrains that automatically incorporate user-specified gameplay elements.

II. BACKGROUND AND RELATED WORK

Procedural Content Generation (PCG) has many uses in game development from generation of game resources (i.e. textures and meshes) to the generation of entire game levels, with generated layouts and objects being automatically placed around the scene. Originally, simple PCG techniques were used to generate random dungeon layouts for a game called ''Rogue'' [5] and a genre of games thereafter referred to as ''Rogue-likes'' (''NetHack'', 1987; ''Moria'', 1994; ''Diablo'', 1996). Since then, there have been many contributions to PCG due to its potential in the automation of game development. PCG techniques have evolved to be capable of generating level layouts that meet specific gameplay requirements such as generating desired player paths or designing levels based around a story or series of specified events. These are however mostly constrained to the 2D space and not utilized in terrain generation. Examples of these include the work of Ashlock *et al.* [6] that used a Genetic Algorithm (GA) to evolve maze-like level layouts according to factors such as path length and branching factor. Other techniques produce grammars that are capable of generating game levels that meet the criteria of a given story or action graph [7]–[9]. Levels generated using these approaches contain a series of areas that are set out in the logical order for game or story events to unfold.

Advancements in technology have allowed the creation of games that feature open worlds, with expansive outdoor environments and detailed interior settings. These games have become increasingly popular and therefore PCG techniques have been created that aid in the generation of these virtual worlds. For interior environments, there are techniques that can generate entire house or building layouts [10]. These interior levels can then be furnished with other techniques that place objects such as tables, chairs, and even dinner plates and cutlery [11], [12]. For exterior environments there are techniques that can generate roads through an existing terrain following an optimal path based on a set of cost functions including terrain slope and the presence of bodies of water [13]. Other techniques can generate entire road networks, villages, and even cities [14], [15]. The largest and one of the most important components of an exterior environment is the terrain and therefore many techniques that generate terrain have been developed.

A. AESTHETIC-BASED TERRAIN GENERATION

Most terrain generation techniques focus solely on the terrain's visual appearance without considering how the terrain might be used for gameplay. Physics-based methods include Cordonnier *et al.* [16], which simulates tectonic uplift and stream powered erosion to generate realistic terrains. This approach requires a grey-scale uplift map as input, which represents the speed at which the terrain is raised by tectonics. and creates realistic-looking terrains that require post-editing for use in video games. Cordonnier *et al.* [17] also introduced another physics-based terrain generation method that represents the terrain using a discrete layered model. The layers in this model collectively represent the state of the scene at each frame and are updated at each time step by geomorphological and ecological events such as rain, gravity, temperature, wind, fire, and lightning that can be adjusted interactively by the user during the terrain generation process.

Peytavie *et al.*'s [18] simulation-based approach offers high level terrain authoring tools involving physical simulations to automatically stabilise layers of sand and rock to keep the terrain looking natural and detailed. For example, the user is able to add sand to the terrain and have the physical simulation use the sand to automatically fill crevices, or the user may cut out a section of cliff face and the physical simulations erode the terrain and place rock piles where the terrain has crumbled.

A collection of example-based methods that take a basic sketch of a terrain as input and output a detailed version of the terrain have been used in [19]–[24]. Most of these approaches allow the user to sketch a terrain as a series of lines that represent ridges, roads, or rivers, and use various algorithms to generate the terrain around them. For example, Rusnell *et al.* [19] uses Dijkstra's pathfinding algorithm to get the distance between points on the terrain and the sketched ridges, and uses these distance values to blend the ridges into the terrain. Other example-based techniques have use real-time input from the designer, rather than a pre-drawn sketch. For example, Gain *et al.* [25] allows users to draw feature silhouettes in real-time and the terrain will update with these drawn-in features.

Raffe *et al.* [26] proposed another sample-based technique that pieces a terrain together using existing terrain tiles via

interactive evolutionary computation (IEC). This technique represents terrains as a 2D grid of terrain tiles, which are small square chunks of pre-generated terrain. The IEC used in this technique initialises a population of generated terrains as a random selection of tiles. The IEC is similar to a GA except rather than use an automatic fitness evaluation method and selection scheme, the user is required to interact with the process at each generation to select the most desirable terrains to be used in mutation and crossover. The user is also able to specify tiles that they do not want to be used in future generations, giving them finer control over the process. This technique was developed with gameplay in mind but is not an automated process, requiring the user to intervene regularly during the generation process. Also, the grid-like nature of this technique is apparent in the generated terrains making them look more uniform and less natural.

A similar patch-based approach was developed by Antoniuk and Rokita [27], where each patch was assigned a base-height, dispersion values, and a terrain type. The terrain type determined which algorithm was used to generate the height values of the patch, which were offset by the patch's base height. Once the terrain for all the patches has been generated, the boundaries between patches are blended to form smooth transitions. Although this technique does not require frequent user input, it does require detailed information to be provide. This includes a low resolution height map that stores the base heights of each patch, a text file containing multiple dispersion values for each patch, and a type map that stores the terrain type of each patch.

Most recently, deep learning approaches such as Generative Adversarial Networks (GANs) [28] have been explored for terrain generation [29]–[33]. For example, Guérin *et al.* [29] proposed an approach that combine the use of GANs and a sample-based terrain generation method. Their method uses four types of terrain synthesisers to allow the user to interactively edit a terrain. These four synthesisers are used to 1) transform a sketch to a terrain, 2) transform a levelset to a terrain, 3) erase a section of terrain, and 4) apply erosion to the terrain for added realism. Each synthesiser is a Generative Adversarial Network (GAN) [28], which has been trained on existing terrain exemplars to produce realistic edits to the generated terrain. This method suffers from the same drawback as other sample-based methods in that it requires existing terrain data to train the GANs as well as not considering any form of gameplay.

The primary downside to these sketch-based or interactive approaches is the amount of input required from the user who must design where every river, mountain ridge and road must go as well as specifying additional parameters such as the roughness and elevation of the terrain for each of the sketched features. Also, as the techniques' focus is on producing terrain based on physics and aesthetics, the user must explicitly design their environments to allow for any desired gameplay elements.

B. GAMEPLAY REQUIREMENT-BASED TERRAIN GENERATION

Several fully automated techniques that generate terrain containing features to facilitate generic gameplay requirements, such as balance and accessibility, have also been developed. One such technique is proposed by Togelius *et al.* [34] where a GA is used to generate Real Time Strategy (RTS) game maps, including terrain, resource placement, and player base placement. The chromosome in this approach uses lists of integers to represent 2D location coordinates for bases, resources, and hills, which also include an additional height dimension. The fitness function evaluated the game maps based on five objectives, namely, distance between player bases and resources, distance between player bases and other player bases, distance between resources and other resources, elevation of player bases, and map symmetry. These objectives could be selected to be used in a multi-objective evaluation producing a Pareto front of solutions from which a designer could choose from. This is a useful technique except that it does not produce any aesthetic details and testing has so far been limited to RTS maps.

Another technique for generating terrain maps for RTS games is proposed by Olsen [35] who developed an efficient erosion algorithm to form aesthetic terrain with a high probability of containing large accessible areas suitable for RTS gameplay. This technique used Voronoi diagrams to represent mountain and hill boundaries, and ''pink'' noise for terrain detail. An optimised hybrid algorithm was then applied to simulate thermal and hydraulic erosion to add realism. Owing to the use of Voronoi cells to represent mountains and hills, simple image processing techniques could be applied to alter their elevation and widen paths between them to increase accessibility. This technique is not only efficient but also produces aesthetically pleasing terrain while taking accessibility into account. Although, with the right parameters, this technique is likely to produce terrain with a suitable amount of accessible area, there are many other game related aspects that are not considered.

Frade *et al.* [36] introduced a method of evolving terrain programs via interactive evolutionary computation. A terrain program is a program that can generate multiple terrains that all contain the same morphological characteristics. This work initially required user intervention to determine good terrain programs during the evolution process, but in later works user intervention was replaced with a fitness function. The fitness function proposed in Frade *et al.* [37] evaluates a generated terrain based on the amount of terrain that is accessible. The evaluation is performed by creating a binary accessibility map which represents areas of terrain as either accessible or inaccessible based on a slope inclination threshold. After the initial accessibility map has been generated, a component labelling algorithm is run to determine the number of accessible areas in the map, and the number of cells that contribute to each accessible area. Only the largest accessible area remains in the map while the smaller areas are removed. The overall fitness of the terrain is determined by the size

of the largest accessible area in comparison to the desired accessible area size. In a subsequent article, Frade *et al.* [37] introduced another method of evaluating generated terrains based on obstacle edge length, which promoted the inclusion of inaccessible terrain for the player to navigate around.

Andereck [38] introduced another PCG technique which generated terrain that conformed to an accessible player path. This technique allows users to specify path control points which are then connected by Hermite splines to form a smooth player path. Once the path is generated, a terrain fitting algorithm is run which transforms a flat plane into believable terrain while ensuring the inclusion of the desired path. Although this technique was not designed with video games in mind, fitting terrain to desired player paths would be a valuable tool in PCG for video games.

Smelik *et al.* [39] introduced a sketch-based approach for procedurally generating terrains for military training games. This technique allows users to provide a rough sketch of the terrain, including locations to place buildings, roads, and vegetation. It then converts the sketch to an editable layered terrain map that contains detailed terrain data, and finally generates the terrain as a combination of a height map and 3D models. Although designed for creating game levels, this approach still requires a large amount of user input with desired game design elements having to be manually expressed by the user in the sketch.

Another technique proposed by Smelik *et al.* [40] used two types of constraints to control the generation process of virtual worlds: semantic constraints and feature constraints. Feature constraint is linked to a single feature in the terrain (e.g. forest) and a semantic constraint is composed of several feature constraints, Semantic constraints could be specified by the designer and included line-of-sight, chokepoint, route, and concealment constraints. Feature constraints govern the placement and state of virtual world features such as trees and terrain. Semantic constraints are enforced by an interaction with feature constraints where the feature constraints are required to alter their current state to conform to the requirements of the semantic constraints. For example, a line-of-sight constraint may be placed between two locations which are separated by terrain that contains two feature constraints, hilly terrain and a forest. The line-of-sight constraint interacts with the hills and forest features, forcing them to alter their state to conform to the semantic constraint. The hills feature conforms to the semantic constraint by lowering its hills appropriately to make line-of-sight possible while the forest feature removes trees from the line-of-sight, meeting the constraint requirements. This technique is a step towards enforcing desired gameplay requirements during a virtual world generation. However, it is not a terrain generation technique and contains no procedural generation of virtual world features.

C. ANALYSIS OF THE STATE OF THE ART

In summary, there are currently many terrain generation techniques that are focused on aesthetics and very few are focused on incorporating gameplay requirements. This forms a noticeable gap in current knowledge and there is still no method that can generate terrains that meet a wide variety of gameplay requirements. Limited work has been done in this area by Olsen [35], Togelius *et al.* [34], and Frade *et al.* [37] but their sole focus is either on accessibility or the RTS genre. Andereck's [38] work focused on generating terrain to conform to a desired path, which may be useful as a terrain generation component, but is still far from being a complete terrain level generator that incorporates gameplay elements. Smelik's approach includes game design in a terrain and uses semantic constraints that can interact with the terrain to ensure it meets the desired design of a user [40]. Their results showed that using semantic constraints is a viable approach towards incorporating gameplay elements in terrain generation, but their approach is not scalable. Their approach allows game design constraints to be enforced, but the user is still required to manually design and create each of the virtual environments and implement the desired constraints. With accessibility and the RTS genre being the main focuses in this field there is room for more scalable approaches that can generate terrains with a wider array of gameplay elements for other game genres.

D. QUANTIFYING GAMEPLAY ELEMENTS – LOOKING TO THE FIELD OF ARCHITECTURE

As mentioned earlier, a number of gameplay elements have been described by Hullett *et al.* [1], where they are referred to as design patterns. While various characteristics of these gameplay elements have been detailed, no attempts were made in [1] at quantifying them. However, some existing work in the field of architecture have examined using isovist measures to characterize areas of physical space.

An isovist is defined as the visible (non-occluded) volume of space surrounding a given point within an environment. The properties of an isovist have been used to inform the design and analysis of landscapes and urban environments. Various measures can be calculated for an isovist, such as its size and the maximum distance that can be seen from the isovist's origin and these can be mapped to characteristics of the area. For example, Benedikt [4] suggests that the privacy of an area could be determined by the volume and skewness of its isovist.

Example applications of isovists include Conroy-Dalton and Dalton [41], where isovists were generated from 2D floor plans of buildings or urban spaces, allowing the user to see various measures associated with the generated isovists. Van Bilson and Poelman [42] also proposed using isovist fields in 3D virtual environments, presenting measures from a field of isovists as 2D images for further analysis. However, neither of these approaches have used space analysis for automatic classification of area types.

Volume becomes an important measure when we work with 3D isovists. Little existing work has involved using 3D isovists so techniques for calculating the volume of a 3D isovist are limited [43]–[47]. The isovist volume technique

in [43] depends on having a voxel-based world model, where isovist volume calculation simply becomes a voxel summation exercise. This is similar to the work in [44], where isovist volume, referred to as the 'spatial openness index', is calculated by dividing space using a discrete grid and summing volumes. In [45] the isovist, referred to as the 'volume of sight' is decomposed into 3D segments whose volume is calculated and summed. Rodriguez *et al.* [47] introduced the Rayburst Sampling algorithm, in the context of calculating volume within medical images, but directly transferable to isovist volume calculation, where a set of radials are cast from a central point within a structure in a volumetric image towards the surface of that structure. These radials were used to define a triangular surface mesh, and to form a set of pyramidal volumes that, for a star-convex shaped structure, could be summed to determine the volume.

The main limitations with current techniques are the complexity of these techniques, in terms of either complexity of the volume calculation or complexity of transforming the environment into a form for the calculation (as in voxel-based method). An attempt at a fast isovist calculation method was made by Dalton *et al.* [46], whose method involves summing the radial lengths of an isovist. This method does not result in the actual isovist volume, rather in a value that has a strong correlation to it.

III. ISOVIST AND GRAPH-BASED TERRAIN MEASURES

As discussed in the Section II C, existing work has mainly focused on generation of aesthetic terrains rather than those incorporating gameplay elements that capture high-level designers' intent as these elements are hard to quantify. The development of measures for characterization and automatic evaluation in terms of suitability of an area of terrain for a gameplay style is an important step towards automatic gameplay requirement-based terrain generation. To characterize local areas, we employ properties of an isovist [4] and to examine relationships between the set of areas on a terrain measures from graph theory are used. Details of the algorithms to compute these measures are detailed in the following sections.

The input for calculating the proposed set of terrain measures are: (1) a terrain heightmap and (2) a layout graph. The heightmap is a two-dimensional array of values representing the surface of a terrain as a series of uniformly spaced height values. An example is shown in Fig. 1. (a), where the height value of each array cell is represented using grayscale shading. This is a common representation used for terrain in computer games. The layout graph is a set of nodes and edges that encodes the expected areas and paths that would be traversed by a player. This layout graph can be designed manually by a game designer or automatically through an algorithm [48]. Each node in the layout graph represents an area of terrain that can contain a gameplay element, with attributes that specify the coordinates of the area on the terrain surface. The edges in the graph represent the physical connectivity of the areas (i.e. traversable paths) in the terrain. An example layout graph

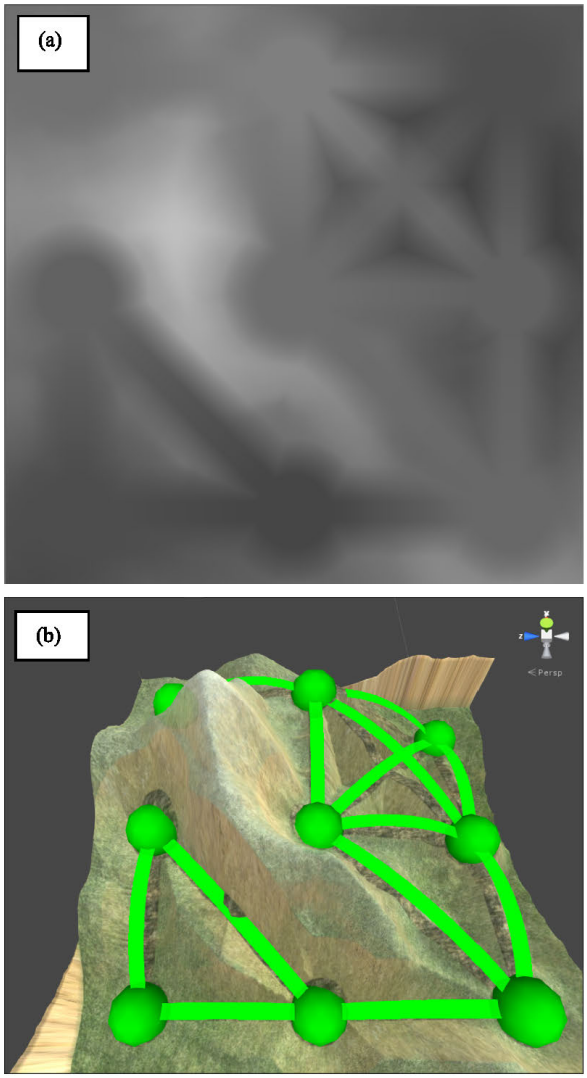

FIGURE 1. Examples of the two inputs required to characterize a terrain's areas. (a): height map (b): the layout graph (superimposed on the rendered terrain).

is shown in Fig. 1(b), overlaid on the 3D rendering of the terrain, with green spheres representing nodes of graph, and green curves showing the edges between nodes.

Table 1 lists the complete set of measures utilised in this approach for characterizing an area in the terrain. Each measure, apart from the *Area Radius*, belong to one of two categories; graph measures (G1-G4), and isovist measures (I1-I11). The following sub-sections discuss the calculation of these measures.

A. THE 'AREA RADIUS' MEASURE

We define the *Area Radius* measure for a node in the layout graph as the radius of a circular area centered on the location of the node on the terrain where the area is at least 90% traversable. This represents a measure of the extent of traversable terrain from the center of the specific area. To calculate the *Area Radius*, we use the terrain heightmap

TABLE 1. List of measures used to characterize an area of Terrain.

Measure	Description
Area Radius	The radius of a circular area of traversable terrain surrounding the areas center.
Degree Centrality (G1)	Number of edges connected to a node.
Eigenvector Centrality (G2)	The influence of the node in the graph based on its degree centrality and the degree centrality of its neighbors, and their neighbors, etc.
Betweenness Centrality (G3)	A measure corresponding to the number of shortest paths a node belongs to.
Closeness Centrality (G4)	A measure of how close a node is to every other node based on the shortest paths between them.
Average Radial Length (I1)	Average distance from a given location to its isovist's perimeter.
Dispersion (I2)	The difference between the values of the mean and the standard deviation of the isovist's radial lengths.
Drift 3D (I3)	The distance between the isovist's origin and the isovist's center of gravity.
Drift 2D (I4)	Same as Drift 3D but ignores the Y (Up) axis.
Maximum Radial Length (I5)	Maximum distance from a given location to its isovist's perimeter.
Minimum Radial Length (I6)	Minimum distance from a given location to its isovist's perimeter.
Skewness (I7)	Skewness of the distance from a given location to its isovist's perimeter.
Sphericality (I8)	How well the isovist volume approximates a sphere.
Standard Deviation (I9)	Standard deviation of distance from a given location to isovist perimeter.
Variance (I10)	Variation in distance from a given location to its isovist's perimeter.
Volume (I11)	Area that can be seen (is not occluded) from a given location.

to calculate an *Accessibility Map* of the terrain and apply a 32 steps binary search process on the *Accessibility Map* to iteratively refine an initial value of the radius, aiming for 90% traversability.

The *Accessibility Map* is a 2D array of binary values indicating the traversable (accessible) and non-traversable sections of the associated terrain which is determined using terrain gradient and a gradient threshold. It is generated from the input terrain height map by calculating the gradient of the terrain at uniformly spaced locations and using a gradient threshold to determine if the terrain is too steep to be traversed at each point. Terrain gradients are first calculated using a Sobel filter, which returns gradient values, in the range of [0, 1], which linearly maps to an angle in the range of [0, 90] degrees. For example, a gradient threshold of 0.5 means any terrain with a gradient of 45 degrees or greater is too steep and thus non-traversable. The Sobel operator is employed in our approach, as it is one of the simplest and most commonly used first order derivative operators for computing gradient in the literature. All first order derivative operators work similarly and suffer from sensitivity to noise. However, the Sobel operator with its larger convolution kernel (3×3 instead of 2×2 in Robert) makes it less sensitive to noise but is slower to compute. The Prewitt operator, uses a slightly different kernel to the Sobel operator but works in a similar way.

B. GRAPH BASED MEASURES

This section briefly details the methods of calculating the four graph-connectivity measures that are listed in Table 1 (G1-G4). Graph measures are calculated for each area in the *Layout Graph*. These include *Degree Centrality*, *Eigenvector Centrality*, *Betweenness Centrality*, and *Closeness Centrality.* These measures can be used to determine various characteristics of an area, such as the likelihood of the area being visited. The discussion is brief as we have used standard means of calculation for these established graph measures, which can be found in various texts, such as [49].

Degree Centrality is calculated by counting the number of edges that are connected to the node. *Eigenvector Centrality* is a measure of how much influence a node has in a graph. Calculation of this measure works off the concept that a node's influence is determined by the influence of its neighboring nodes. Calculating this measure is an iterative process, where each iteration further refines the influence of each node based on the influence of neighboring nodes. This means performing more iterations will result in a more accurate centrality measure. We chose 16 iterations in the calculation for *Eigenvector Centrality*.

In order to calculate *Betweenness Centrality*, the shortest path between each node in the graph to every other node in the graph is required. The *Betweenness Centrality* for any given node is then calculated as the number of these paths that travel through the given node (excluding the paths that begin or terminate at the given node). For undirected graphs, such as a layout graph, this value must be halved as paths get counted twice (e.g. a node that exists on the path between node A and node F will also exist on the path between node F and node A). Once *Betweenness Centrality* has been calculated it is normalized to be in the range of [0, 1].

In order to calculate *Closeness Centrality* of a given node, a set consisting of the shortest path between the given node and every other node in the graph is required. The *Closeness Centrality* is then calculated as the sum of the lengths of the paths in this set. This value is then normalized by dividing the number of paths in the set by the sum of path lengths. The length of a path is defined as the sum of its edges' lengths, where each edge length is the distance of traversable terrain separating two areas.

C. ISOVIST-BASED MEASURES

This section details the calculation of the 11 isovist measures that are listed in Table 1 (I1-I11).

1) GENERATING AN ISOVIST

An isovist associated with an area is first generated before calculating the measures associated with the area. As in [4], we represent an isovist as a set of radial vectors. Our representation consists of a number of randomly distributed 3D vectors cast out from the origin (a point in 3D space) of the isovist, as shown in Fig. 2., where green lines represent the radials of the isovist. The length of each radial vector

is set to a maximum view distance unless terminated by intersection with the geometry of the environment. Fig. 2. demonstrates this, where some radials reach the maximum view distance, while others intersected with occluding obstacles and terminate early.

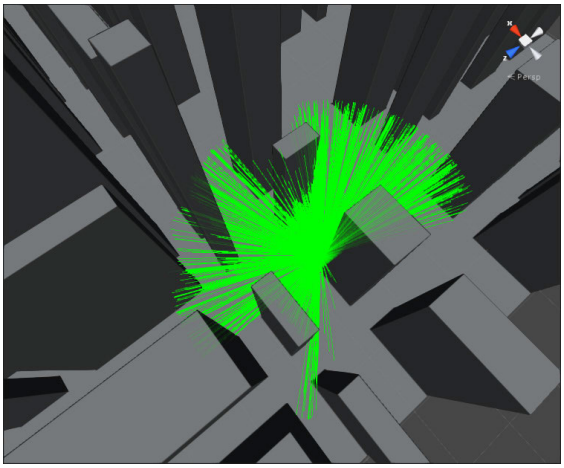

FIGURE 2. An example of a three-dimensional isovist represented as a collection of radial vectors (green lines). While some vectors reach a maximum view distance, others hit occluding obstacles and terminate.

In our approach, outlined using pseudocode in Table 2, four inputs are required in the *GenerateIsovist* function for generating an isovist; the *origin* of the isovist, the maximum view distance, *maxViewDistance*, the *resolution* (the number of radials), and the *environment* (e.g. terrain). The origin is a point in 3D space that the radial vectors are cast out from and is set to the center of the specified area retrieved from the corresponding node in the *Layout Graph*. The height of the origin is set to the eye-height of a theoretical game character that is standing at the center of the area. The maximum view distance is the maximum allowed length of a radial based on visibility considerations. The resolution is the number of radials to cast, where greater resolutions result in more accurate isovist measures, but also an increased computation time. The environment (i.e. the terrain) is used to perform intersection tests with radials to determine any occlusion.

TABLE 2. Pseudocode for Generating an Isovist.

```
FUNCTION GenerateIsovist(origin, maxViewDistance,
resolution, environment)
        isovist = NewIsovist()
        isovist.SetOriginTo(origin)
        isovist.SetRadialCount(resolution)
        FOR EACH radial IN isovist DO
                direction = GetRandomDirection()
                radial.SetDirectionTo(direction)
                distance = RayCast(origin, direction,
environment)
                IF distance < maxViewDistance THEN
                        radial.SetLengthTo(distance)
                ELSE

        radial.SetLengthTo(maxViewDistance)
                END IF
        END FOR
        RETURN isovist
END FUNCTION
```

The process of generating an isovist in the *GenerateIsovist* function involves creating a new *isovist* variable, which is a composite datatype consisting of:

- A 3D point to hold the isovist origin using (x,y,z) coordinates.
- A container holding a set of 3D vectors, each vector representing one of the radials using (x,y,z) components.

First, the isovist origin is set to the value of the *origin* variable from the input to the function. This is followed by a loop that sets the components of each radial vector, where *resolution* number of vectors are created.

For each radial, it is first initialized by setting its x, y, and z components to random values from a pseudo-random number generator, with the vector then normalized to give it a length of 1. This determines the direction of the radial. Next, the length of the radial is determined by casting a ray from the isovist origin in the direction of the radial vector, to determine where the ray intersects with artefacts in the environment. If the distance between the origin and the first ray intersection point is less than the maximum view distance, then the radials length is set to the distance to the intersection point, otherwise it is set to the maximum view distance. This is done by multiplying the unit length initial direction radial by the desired length.

Once all of the radials have been initialized with direction and magnitude, the *isovist* variable is returned from the function.

2) CALCULATING ISOVIST MEASURES

The collection of isovist measures used in this research has been selected from those used by [3], [35], and [36] along with four new measures. These new measures include *Volume* and *Sphericality*, which replace their 2D counterparts, area and circularity, from the above works, and the *Drift 2D* and *Drift 3D* measures, which are variations of the drift measure from [41].

The *Minimum Radial Length* and *Maximum Radial Length* measures are the smallest and largest, radial lengths of the radials of an isovist, respectively. The *Average Radial Length* is the sum of radial lengths divided by the number of radials.

Variance is the standard statistical variance calculated using the radial lengths of an isovist. It is calculated as the sum of squared differences between each radial length and the *Average Radial Length*, then divided by the number of radials minus one. Equation (1) shows this calculation, where N equals the number of isovist radials, r_n is a single isovist radial, and *arl* equals the *Average Radial Length*. The *Standard Deviation* can then be calculated as the square root

of the *Variance*.

$$Variance = \frac{\sum_{n=1}^{N} (|r_n| - arl)^2}{N - 1} \quad (1)$$

Skewness is calculated, in the same manner as *Variance*, using (2).

$$Skewness = \frac{\sum_{n=1}^{N} (|r_n| - arl)^3}{(N - 1) * sd^3} \quad (2)$$

Dispersion is calculated as the *Average Radial Length* minus the *Standard Deviation*. *Sphericality* is calculated as the volume of a sphere, where the radius of the sphere is equal to the *Average Radial Length*, divided by the actual volume of the isovist. Equation (3) shows this calculation, where *arl* is the *Average Radial Length*, and v is the *Volume* of the isovist, which is calculated using (5).

$$Sphericality = \frac{\frac{4\pi}{3}(arl)^3}{v} \quad (3)$$

Drift 3D and *Drift 2D* are calculated as the magnitude of the average of the radials of an isovist. The difference between *Drift 2D* and *Drift 3D* is that *Drift 2D* only uses the x and z components of the isovist radials, while *Drift 3D* includes the y component (the up axis). Equation (4) demonstrates this.

$$DriftXD = \left| \frac{\sum_{n=1}^{N} r_n}{N} \right| \quad (4)$$

D. NOVEL ISOVIST VOLUME ESTIMATION METHODS

As discussed previously, there is limited existing work associated with the calculation for the volume of a 3D isovist. In this section, we introduce two methods that can be used to approximate the volume of 3D isovist.

1) METHOD 1: ISOVIST METHOD

Our method estimates volume of a 3D isovist using only the set of N isovist radial vectors, shown in (5). The approximation can be interpreted as follows: each radial defines a spherical volume where the surface of that sphere touches the isovist surface at the end of that radial. The estimated volume of the isovist is obtained by taking the average of the spherical volumes.

$$Volume = \frac{4\pi}{3} \frac{\sum_{n=1}^{N} |r_n|^3}{N} \quad (5)$$

This technique represents the volume of an isovist, or any volume that has the property of being a star-convex set [50] with respect to the origin of the radials. The volume approximation does not hold for non-convex volumes that are not star-convex, as in such volumes some of the cast out radials will be occluded from interrogating the complete surface of the shape.

The algorithm is efficient, as it relies on knowing only the radials, which would in any case need to be obtained to compute other measures of the isovist, such as those discussed in [4] and [41], in order to characterize the space it represents. The algorithm is also simple, as it does not depend on any relationship between the radials as is needed by techniques that use triangulation, such as in [47].

2) METHOD 2: MONTE CARLO-BASED METHOD

Based on the Monte Carlo principle [51], Monte Carlo integration is an established method of calculating volume [52] of objects, that based on our search of the literature has not been applied to isovist volume computation. Monte Carlo integration relies on the unknown volume being enclosed by a known volume and being able to test whether a point is within the unknown volume. A set of random points is taken from within the known volume and the number of these points that also lie inside the unknown volume is determined. The unknown volume is then approximated as the known volume, multiplied by the ratio of points in the unknown volume to the total number of points. As the number of random points approaches infinity, the volume approximation approaches the true volume. In the calculation of the volume of a 3D isovist, the known volume equates to the volume of a sphere with a radius equal to the maximum view distance. A random point is considered being inside the 3D isovist if its distance to the origin of the isovist is less than the maximum view distance, and if there are no occluding obstacles between the point and the origin. The calculation of 3D isovist volume then entails the added computational burden of testing a multitude of random points against the environment.

IV. EXPERIMENTS AND RESULTS

The measures proposed in this article were first evaluated using an existing set of terrain-based levels with manually identified gameplay elements. Section A contains the details of this evaluation, based on our discussion from [48]. Section B describes evaluations for the novel 3D isovist volume estimation methods. Lastly, Section C contains evaluations for demonstrating the application of the set of proposed measures for incorporating gameplay elements into a user-specified terrain, including a measure of the time taken to produce the complete set of measures. The performance times obtained for the experiments are from running on a computer with an Intel Core i7-4700HQ CPU with 32 GBRAM.

A. EXPERIMENTS AND RESULTS OF USING PROPOSED MEASURES TO CLASSIFY AREA TYPES

To determine the suitability of the proposed measures in being able to identify gameplay elements, we used a set of terrains containing areas that mapped to specific gameplay elements, as defined in [1]. The terrains consisted of publicly available height maps for the video game "Savage: The Battle for Newerth" [53]. For each terrain, we used the player traversable regions to manually construct the layout graph and the gameplay element descriptions from [1] to identify areas on the terrain corresponding to hidden areas, open areas, vantage points, choke points, and strongholds. The number of instances of each gameplay element is shown in Table 3.

TABLE 3. Gameplay element Instances in the Testing dataset.

Gameplay Element Type	Number of Instances	Description (reproduced from [1]).
Hidden Area	11	Small, well-occluded area, usually off the main route.
Open Area	15	A large, open area usually used for large-scale combat in FPS style games.
Vantage Point	19	An elevated location which looks over a portion of the game level.
Choke Point	15	A small area which must be traversed to reach a portion of the game level.
Stronghold	10	A well-covered area with limited access points.
Total	**70**	

For each gameplay element instance, the area radius and isovist-based measures were calculated from a point in the centre of the area, 50 units up from the terrain height at that point (defined as eye-height for a character). The graph-based measures were calculated from the layout graph of the corresponding terrain. This produced a dataset, where each data point consisted of the set of measures along with the gameplay element type.

The measures were first evaluated using an information theory-based approach, calculating the information gain of each measure with respect to gameplay element type. The information gain of each attribute is shown in Table 4, with the attributes ranked in descending order of information gain.

TABLE 4. The measures, ranked by information gain with respect to game element type.

Attribute	Information Gain
Area Radius	1.144
Degree Centrality	0.918
Isovist Sphericality	0.750
Eigenvector Centrality	0.692
Isovist Dispersion	0.692
Betweeness Centrality	0.626
Isovist Skewness	0.455
Isovist Volume	0.407

Based on information gain, the most influential attribute is the area radius, followed by the graph-based measure of degree centrality. Of the isovist-based measures, *Sphericality* had the largest information gain with respect to gameplay element type. These showed that the amount of accessible terrain, shape of the visible volume and the location of the area with respect to the rest of the terrain can characterize the gameplay elements, which is supported by their descriptions from Table 3.

Both the minimum and maximum isovist radial lengths had an information gain of zero. This is due to the particular nature of the terrain as every area has an equal value for both of these attributes, which means they cannot differentiate between different areas. All of the areas are based on outdoor terrain environments with no occlusion from above, and guaranteed occlusion below by the ground. As such, we found that in each area type, the minimum radial length was the distance from the isovist centre to the ground and the maximum distance resulting from radials facing directly up, with these distances being capped to the maximum view distance. Although the minimum and maximum radial lengths were not essential in the context of our chosen terrain environments, it is important to include them in our proposed measures as they can play a more definitive role in different environments, such as indoor game levels and outdoor games with features such as caves and tree canopies.

To further test the ability of the proposed measure to differentiate gameplay elements, the gameplay element dataset was used to construct classifiers whose performance was measured. Four different classifiers were evaluated, J48 decision tree, naïve Bayes, multi-layer perceptron (MLP), and a probabilistic neural network (PNN). The classifiers used were from the WEKA [54] software package. Each classifier was evaluated using 10-fold cross validation and parameters related to the classifiers and training process were left at their default values. Table 5 shows the classification accuracy and Cohen's Kappa [55] for each of the classifiers. Cohen's Kappa provides a measure of how closely the classifier matches the ground truth class for nominal data, in this case how closely the game element type predicted by the classifier based on the proposed measures matches the actual game element type as manually labeled when constructing the dataset. It is argued in [56] that a Kappa between 8.1 and 1 corresponds to an 'almost perfect' strength of agreement, and this is achieved by three of the four classifiers, with only the J48 classifier scoring below.

TABLE 5. Classification accuracy and Cohen's Kappa for the gameplay element classifiers.

Classifier	Accuracy (%)	Cohen's Kappa
MLP	87.14%	0.8373
PNN	85.71%	0.8183
J48	84.29%	0.8041
Naïve Bayes	85.71%	0.8194

Overall, the classifiers performed similarly with the number of correctly classified instances in the lowest performing classifier (J48) being 59 and in the highest performing classifier (MLP) being 61, out of a total of 70 instances. To provide a feel for the individual gameplay area classifications, and their miss-classifications, the confusion matrices of the worst (J48) and best (MLP) performing classifiers are shown in Table 6 and Table 7. For each confusion matrix, the label of the actual class an area belongs to is in the far-right

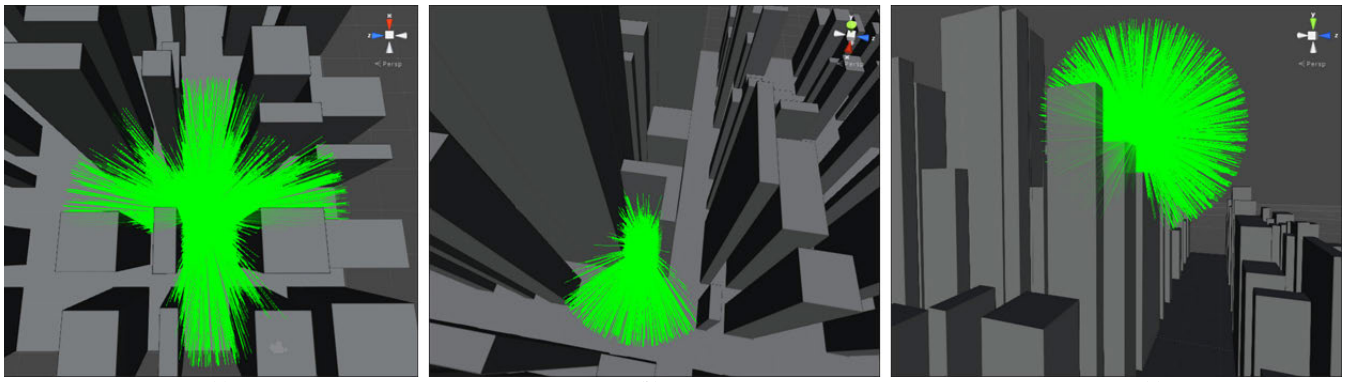

FIGURE 3. Three scenarios, where an isovist has been generated in a complex environment. (a) an isovist generated in the middle of a street in an urban environment. (b) an isovist generated in a hidden alcove. (c) an isovist generated at a vantage point on top of a building.

TABLE 6. Confusion matrix for the J48 classifier.

Hidden Area	Open Area	Vantage Point	Choke Point	Strong hold	<- Classified As.
10	0	1	0	0	Hidden Area
0	12	0	1	2	Open Area
1	0	18	0	0	Vantage Point
1	0	0	12	2	Choke Point
0	3	0	0	7	Strong hold

TABLE 7. Confusion matrix for the MLP classifier.

Hidden Area	Open Area	Vantage Point	Choke Point	Strong hold	<- Classified As.
10	0	1	0	0	Hidden Area
0	12	0	1	2	Open Area
0	0	18	1	0	Vantage Point
1	0	0	13	1	Choke Point
0	1	0	1	8	Strong hold

column, with each other column listing the number of area instances classified as a particular area. For example, for the J48 classifier confusion matrix in Table 6, the first row of numbers correspond to the 11 hidden areas present in the dataset. Out of the 11, 10 were correctly classified as hidden areas and one was miss-classified as a vantage point,

By examining the confusion matrices for both J48 and MLP, we can see most areas correctly classified, with the most misclassification between the Strong hold and the Open Area classes. For J48, three of the 10 strong holds were miss-classified as open areas and two of the 15 open areas were miss-classified as strong holds. The MLP classifier performed better in terms of differentiating these two areas, with only one of the strong holds being miss-classified as an open area, whilst still two open areas miss-classified as strong holds.

B. EVALUATIONS OF VOLUME ESTIMATION METHODS

Three methods for estimating volume of an isovist are examined. These methods are: Rodriguez's Rayburst Sampling algorithm [47], Isovist method and Monte Carlo-based method. Three outdoor scenarios were used, where the terrain is populated by a variety of boxes to occlude visibility. Fig. 3. displays these three complex scenarios and their associated isovists. The scenarios are labelled (a) Urban Street, (b) Hidden Area, and (c) Vantage Point. The isovist in the Urban Street scenario was placed in the middle of a street intersection, while the isovist in the Hidden Area scenario was placed in an alley cul-de-sac. The isovist in the Vantage Point scenario was placed on top of a building.

To measure the accuracy of the isovist volume estimation methods, a measure the ground truth was obtained. Here, we have relied on the property of Monte Carlo integration, where as the number of points tends to infinity, the estimated volume converges to the actual volume. An estimate of the ground truth was obtained using the Monte Carlo-based method where the resolution was increased to a large number (64000000 points).

In terms of the experimental setup, each experiment involved three methods to estimate the volume of an isovist at one of four resolutions, 256, 1026, 4098 and 16386. Each volume estimation was repeated 2048 times due to the non-determinism in the algorithms.

Evaluation of the results from these experiments is divided into two parts. First, we compare the distribution of estimated volume across the 2048 runs associated with each of the three methods, then we examine the computation times of each method.

Results from the urban scenarios are visualized as box and whisker plots in Fig. 4. Each box and whisker represent the distribution of estimated volume from one particular algorithm at a particular resolution for the 2048 runs.

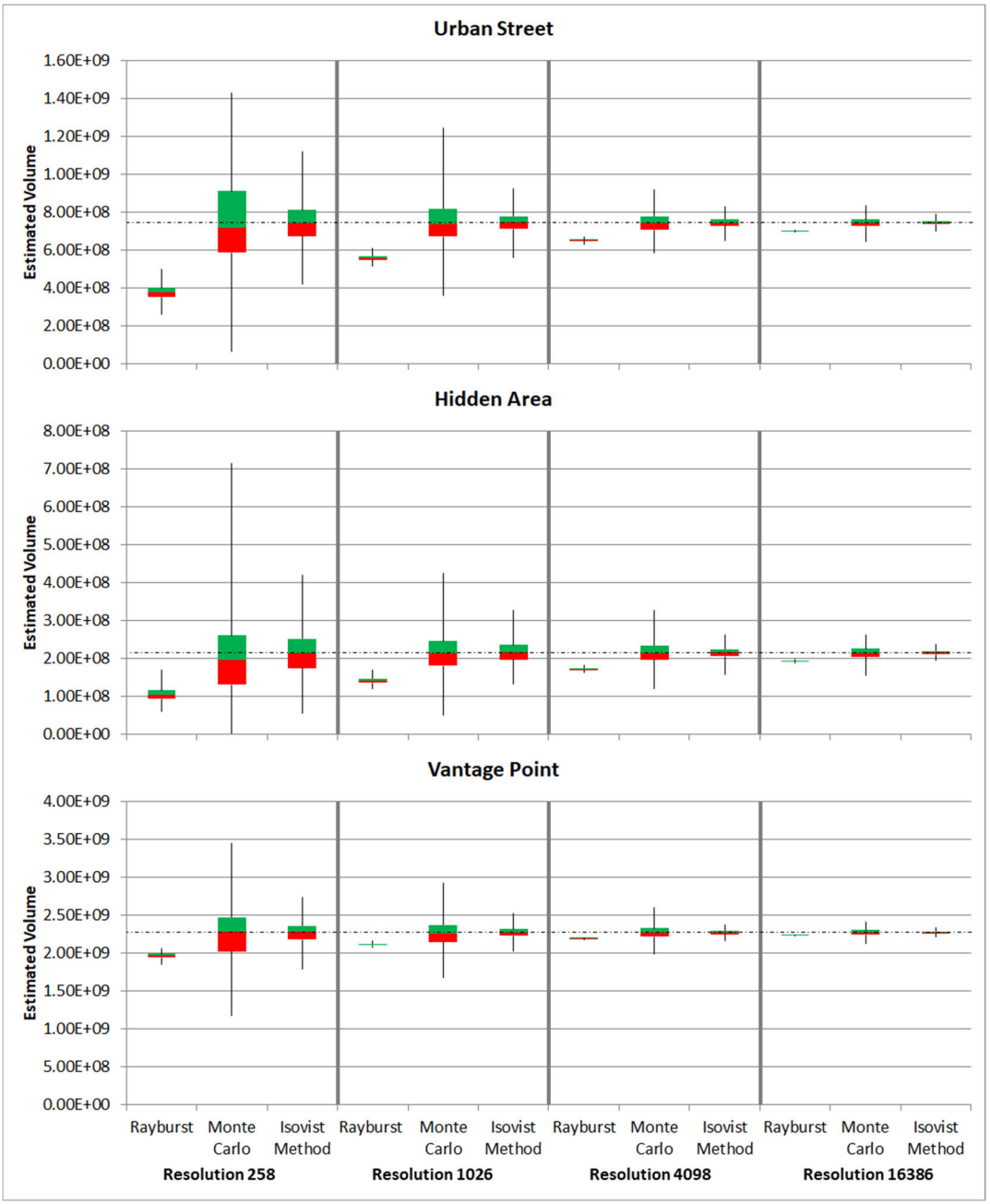

FIGURE 4. The box and whisker plots of the three urban scenarios, dotted horizontal line indicated actual volume.

A dotted horizontal line indicates the 'ground truth' volume. These figures show that, the Rayburst Sampling method was the least accurate in terms of median volume estimations. Even at the highest resolution, Rayburst was typically less accurate than Monte Carlo based method and the Isovist Method, although it does have the least variance in its estimations as shown by its small interquartile range. Monte Carlo based method had produced the greater variance, with the largest interquartile ranges, but produced accurate median values similar to those of the Isovist Method. The Isovist Method had estimations with smaller interquartile ranges than the Monte Carlo based method, making its estimations more reliable, and also produced the most accurate median estimations.

Fig. 5. illustrates why the Rayburst method is less accurate at lower resolutions. This figure shows a 2D example of how the Rayburst method triangulates space. The Rayburst method sums the area of each triangle to calculate the area of the isovist, but this leaves pockets of space that do not have their area included in the calculations. This means that

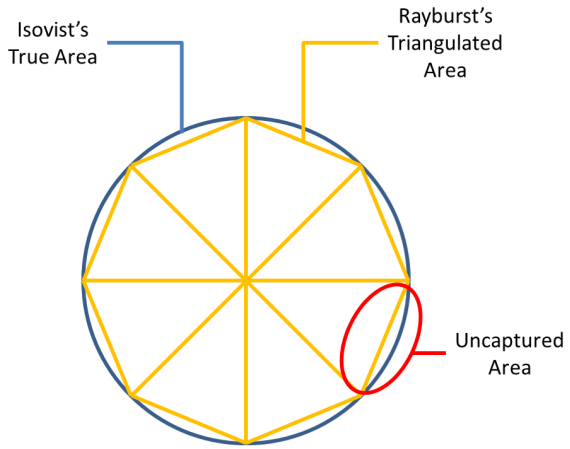

FIGURE 5. Example of how the Rayburst volume estimation method cannot capture an isovist's true volume.

higher resolutions are required to minimize the amount of uncaptured space.

To summarize, the proposed Isovist Method produced the most accurate median values with smaller interquartile ranges than Monte Carlo integration. The Rayburst method produced the smallest interquartile ranges, but its volume estimations were less accurate than either the Isovist Method or Monte Carlo integration. From the plots (Fig. 4.) it can be seen that all three methods (Rayburst, Monte Carlo, and Isovist) showed more variability in their respective volume estimation at lower resolutions and increasing accuracy as the resolution increases. The Isovist Method consistently outperformed the other methods across the four resolutions and scenarios with increasing complexity.

Besides evaluating the methods in terms of their accuracy in volume estimation, computation efficiency of these methods is examined. For a 'fair' comparison where each volume estimation method was given the same four inputs (the origin of the isovist, the resolution, the maximum view distance, and the environment), Monte Carlo integration was typically the fastest method, on average taking just under half the time of the next fastest method, the Isovist Method. The Rayburst method was the slowest, especially at high resolutions. At resolution 16386 the Rayburst method took, on average, 7.6836 seconds to calculate the volume of a single isovist, compared to the Isovist Method and Monte Carlo method, which took 0.261, and 0.1122 seconds respectively.

In the 'fair' comparison, the time taken for the Isovist and Rayburst methods include the time required to generate the isovist vectors. To make the comparison more meaningful to our application, where the isovist vectors need to be computed anyway so as to obtain the other isovist-based measures, the comparison was re-done, removing the isovist calculation times. This reduced the computation time for the Isovist Method and Rayburst method, at resolution 16386, to 0.0012 and 0.0039 seconds respectively. As such, we have used the Isovist Method in our work.

C. RESULTS FROM USING PROPOSED MEASURES IN AN EVOLUTIONARY TERRAIN GENERATION APPROACH

To demonstrate how the proposed measures can be used to automatically generate a terrain containing a specific set of gameplay elements, we incorporated the measures in an evolutionary approach [48]. The Genetic Algorithm (GA) based approach evolves a set of modifications that, when applied to a specific initial terrain, incorporates areas of user-specified types (examples include: Hidden Area, Strongholds, Vantage Point and Open Area) and associated constraints, also specified by the user. This approach takes three inputs, an initial terrain, an input graph, and an area type dataset. The initial terrain is a terrain represented as a height map to which the evolved modifications are applied. The input graph consists of a collection of nodes, where each node corresponds to a desired area type, and a collection of edges, where each edge represents a desired constraint between two areas, which influences the layout of areas in the terrain. The constraints used were: geographic (straight line) distance, path distance, traversability (whether its possible to traverse from one area to the other), and line-of-sight (whether one area is visible from the other). The area type dataset used was the same as developed for the initial testing of the measures, summarized in Table 3. The area type dataset is used in the fitness evaluation of the approach, where evolved areas are characterized using the 16 proposed measures and compared to the measures corresponding to area instances in the dataset using the Euclidian distance to determine how similar the evolved area is to its desired type.

A series of experiments was conducted to demonstrate the use of the proposed measures for generating a terrain incorporating game designers' intent using our evolutionary approach. Each of the experiments attempted to evolve a terrain with a certain number of area types with constraints between them. For each experiment, the GA run was repeated 30 times, with each repetition starting from a random initial population for 2000 generations. This repetition of the same experiment gives a balanced insight into the performance of the approach, as GA is stochastic. The GA parameters used for these experiments are summarized in Table 8.

TABLE 8. Genetic Algorithm Parameters for the experiments.

Parameter	Setting
Population Size	30
Mutation Probability	3.3% (per gene)
Crossover Probability	60%
Mutation Type	Gaussian + Uniform
Crossover Type	Single-Point
Selection Method	Binary Tournament
Elitism	Keep top 2 individuals unchanged.
Termination Condition	2000 generations

In general, it was found that the approach produces terrains that meet the target requirements, though as the number of requirements (area types and constraints) increases it

becomes more likely that some of the requirements are not met in totality in a particular solution. Taking the terrains corresponding to the highest fitness for each run, for each experiment we now provide a brief analysis of the accuracy.

For the geographic distance constraints between areas, our results showed that on average, it attained within 4% of specified values in approximately 75% of the evolved terrains across experiments involving these constraints.

In terms of the path distance constraints between areas, the difference between evolved and desired path distance was on average less than 2% in approximately 75% of the evolved terrains.

The line of sight and traversability constraints are both Boolean, so these are either met or not. For each experiment, out of the 30 terrains resulting from each run, between 1-4 of the terrains did not meet all of these constraints, meaning they were all met in the majority of cases.

In terms of the look of the final terrain, we present here the results for three of them, with the goal for each of these experiments summarized in Table 9. Experiment 1 was a simple test, to see if a single area type, a hidden area, with no constraints could be evolved into a terrain. Experiment 2 and 3 add more areas and constraints in order to provide a more comprehensive test of our approach.

TABLE 9. Experiments and their desired areas and constraints.

Experiment	Desired Area Types	Number of Constraints
EXP1	1 Hidden Area	No Constraints
EXP2	2 Strongholds 2 Hidden Areas	30 Constraints
EXP3	4 Vantage Points 1 Open Area	50 Constraints

In all experiments, the initial terrain that modifications were applied to was generated using Perlin noise [57] and is showing in Fig. 6. (a). The generated terrain for EXP 1 is shown in Fig. 6 (b), viewed from above in order to showcase the entire terrain. This terrain corresponds to the solution with the highest fitness out of the 30 runs of the experiment. The generated terrain is annotated with a set of spheres, each corresponding to a distinct area type that was evolved into the original terrain. The hidden area is represented as a yellow sphere, with grey spheres representing other areas that were evolved into the terrain in the process. To enable evaluation of how hidden the area really is, Fig. 6.(c) shows the hidden area from the view point of a player standing on the terrain. The hidden area is shown to be a small area with a single access point that is surrounded by elevated terrain to keep it hidden. These are desirable traits for a hidden area. It should be noted that whilst incorporating the hidden area, the evolved terrain has maintained the overall aesthetics of the original terrain.

EXP2 attempted to evolve two strongholds and two hidden areas into the same initial terrain as EXP1 (as was shown in Fig. 6. (a)). This experiment also attempted to maintain geographical distance, path distance, traversability, line-of-sight, and reverse line-of-sight constraints between all areas,

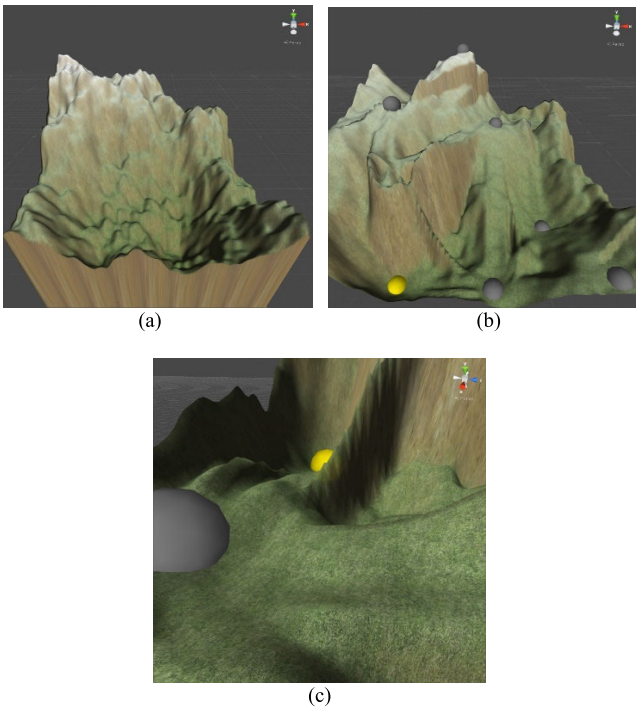

FIGURE 6. The input and output terrain of a simple experiment that evolved a single hidden area into an initial terrain. (a) initial terrain generated using Perlin noise. (b) the generated terrain, with the hidden area represented as a yellow sphere. Grey spheres represent other areas that were evolved into the terrain. (c) A close-up of the hidden area from a player character's point of view.

resulting in a total of 30 constraints. Fig. 7.(a) shows the evolved terrain, where orange spheres represent strongholds and yellow spheres represent hidden areas. Fig. 7. (b) shows the terrain once it has been populated with objects, such as trees and rocks, to demonstrate how a game designer may make use of the evolved terrain. Fig. 7. (c) and Fig. 7. (d) show the terrain from the perspective of a player character. Fig. 7. (c) is looking towards a path leading to a hidden area. Fig. 7. (d) is a view from the path leading to the small hidden area where helpful items may be placed. The geographical distance constraints between the two strongholds were deliberately set high in order to force the strongholds to opposite sides of the terrain, which is evident in the evolved terrain. Further geographic constraints were put in place to ensure each stronghold was located near a hidden area. The strongholds are afforded protection by elevated terrain and have limited access points making them easier to defend. Both hidden areas are small, out of the way, and well covered, making them more difficult to see, but within reach of a stronghold making them suitable places to place items that a player defending a stronghold may venture out to collect (such as health power ups or ammunition).

EXP3 attempted to evolve five gameplay elements, four vantage points and an open area, into the same initial terrain as used in EXP 1 and EXP 2, while maintaining all five types of constraint (geographical distance. Path distance,

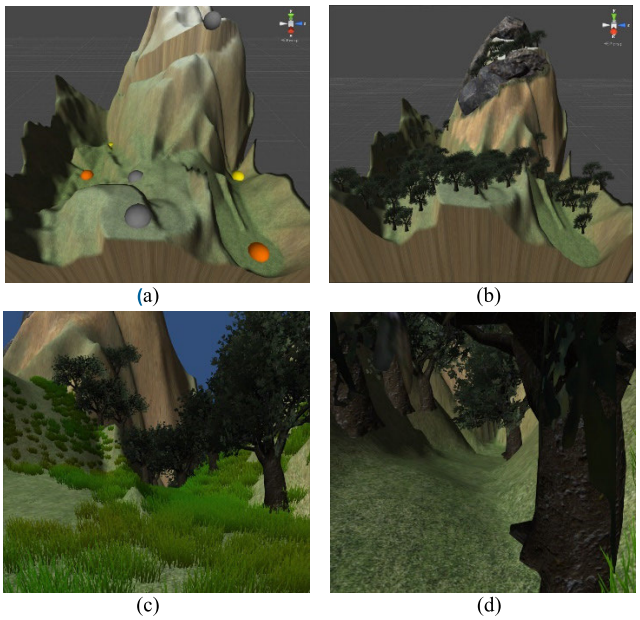

FIGURE 7. (a) A generated terrain that incorporates two strongholds and two hidden areas. (b) the generated terrain populated with trees and rocks. (c) A first-person view of the entrance to a path leading to one of the hidden areas. (d) A first-person view of the path leading to the hidden area.

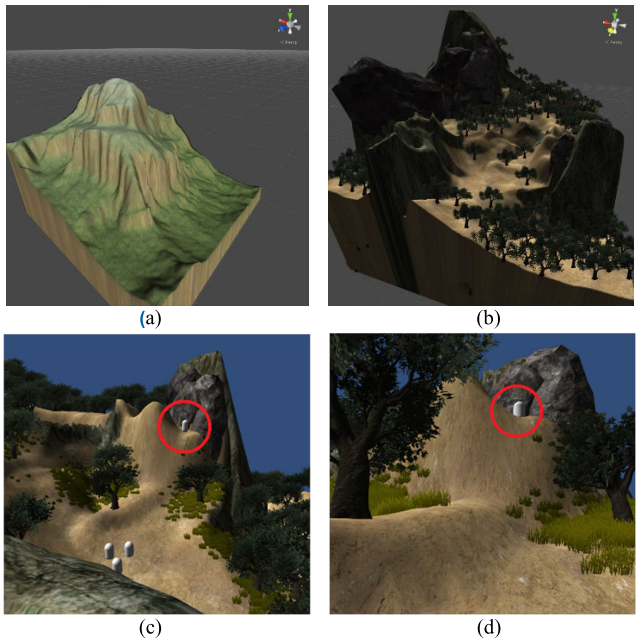

FIGURE 8. (a) The initial terrain used in EXP3. (b) A generated terrain that incorporates four vantage points and one open area, populated with trees and rocks. (c) A first-person view from one of the vantage points overlooking three other players (represented as white capsules) and another player at another vantage point circled in red. (d) A first-person view from one of the three players at the open area, looking up at a player positioned at a vantage point (circled in red).

traversability, line-of-sight, and reverse line-of-sight constraints between all areas) between each gameplay element. This makes for a total of 50 individual constraint values. These constraints forced the vantage points to surround the open area, while maintaining line-of-sight to allow players to utilize the vantage points to gain combat advantages over players in the open area. Fig. 8. (a) shows the initial terrain used in EXP3. Fig. 8. (b) shows the evolved terrain for this experiment, populated with objects similar to the terrain shown in Fig. 7. (b). Fig. 8. (c) shows the open area from the perspective of a player who is positioned at one of the vantage points. This figure shows the player has good vantage over three other players (white capsules) who have been positioned in the open area. A fourth player (white capsule) has been positioned in another of the vantage points and is circled in red. Fig. 8. (d) shows the terrain from the perspective of one of the three players in the open area, looking up at the fourth player positioned in the vantage point, again circled in red. These images show that the vantage points were placed at high locations that overlook the hidden area, as is desired.

We now provide some insight into the computation time taken to evaluate a terrain using the proposed measures in our experiments. The time taken to calculate the 16 proposed measures for a terrain varied between the experiments due to the differing complexity of the level layouts. Factors affecting this performance include the number of areas (nodes) in the terrain, as the isovist measures are calculated for every node in the layout graph, and the complexity of the layout graph, which affects the time taken to calculate the graph-based measures. Average perfomance times, as obtained on an an Intel Core i7-4700HQ CPU with 32 GB RAM, for each of the three experiments are presented in Table 10.

TABLE 10. Time taken to calculate all 16 measures for an entire terrain.

Experiment	Time (Seconds)
EXP1	0.0029
EXP2	0.0144
EXP3	0.0209

V. DISCUSSION AND CONCLUSION

This article has described a collection of isovist and graph measures capable of characterizing an area of terrain and how they are calculated. Importantly, this article also introduced two efficient methods for estimating the volume of an isovist, one based on Monte Carlo integration, and a completely novel technique where the volume is calculated from the isovist radials. Analysis of the proposed isovist volume estimation methods against an additional, known volume estimation method revealed that the proposed volume estimation methods are accurate and efficient to compute. These methods are useful for any isovist based application, such as in urban space analysis and design, or any applications that use star-convex sets.

Use of the measures was demonstrated in an evolutionary approach towards generating terrains that automatically incorporate gameplay elements into an initial terrain. Across

three scenarios of increasing complexity, the measures were used to guide evolution to add gameplay elements, in the form of distinct areas to an initial terrain. The initial terrain was chosen for its aesthetic value, without incorporating the needed gameplay elements. The final evolved terrain kept the aesthetics of the original terrain whilst incorporating the desired gameplay elements.

Time taken for the calculation of the measures depends on the complexity of the level layout, but is fast in general, being in the order of 2 milliseconds for a terrain with five gameplay elements with 50 constraints between them.

The proposed measures have a variety of applications due to their ability to differentiate between gameplay element types with low computational overhead. We have focused on gameplay elements that are typically desired in games of the FPS genre, making this approach particularly useful for games of this genre. The same elements however, are also typically used in other game genres. For example, a puzzle or exploration game may use a vantage point to view the layout of a level so that a player has an idea on how to get to their destination. As another example, open areas are used in RTS games for placement of player bases, or in role playing games (RPG) to place buildings or small towns.

Besides being able to be incorporated in the evolutionary process, as demonstrated in our work, the measures can also be incorporated into a manual terrain creation process. For example, the measures could be integrated into a terrain editing tool, so as to display the suitability of areas within the terrain to particular gameplay types. The artist can then proceed to shape the terrain, using the measures for guidance, to incorporate a desired gameplay element into the terrain.

Applications also exist for these measures outside of the computer games domain, for example in the fields of urban planning, landscape design and mission planning. Such applications in the real world are possible as detailed topographic data, in the form of height maps, exists for much of the surface of the Earth. By evaluating the terrain height maps of actual real-world locations, geographic areas that suit a particular purpose could be identified. For example, in a military scenario when seeking to establish a base an area may be sought that displays the properties of a stronghold, and does not have any line of sight with vantage points that the enemy could occupy.

Ultimately, the versatility of our approach, comes from the ability for the measures to be used to categorize gameplay types using examples of those gameplay types in existing terrains. This means that when wishing to categorize any new gameplay element besides the ones that we have investigated, one does not need the exact values of the measures for the new element, just examples of terrain that include the particular element.

REFERENCES

[1] K. Hullett and J. Whitehead, "Design patterns in FPS levels," in *Proc. 5th Int. Conf. Found. Digit. Games*, 2010, pp. 78–85, doi: 10.1145/1822348.1822359.

[2] A. Pech, P. Hingston, M. Masek, and C. P. Lam, "Identifying attributes for characterizing game area types in virtual Terrain," in *Proc. 7th Workshop Procedural Content Gener.*, 2016, pp. 1–12. [Online]. Available: https://www.dropbox.com/s/0zb9qx1mphdfdbo/PCG2016_paper_3.pdf?dl=0

[3] A. D. King, "The social logic of space," *Landscape Urban Planning*, vol. 13, pp. 247–249, Jan. 1986, doi: 10.1016/0169-2046(86)90038-1.

[4] M. L. Benedikt, "To take hold of space: Isovists and isovist fields," *Environ. Planning B, Planning Des.*, vol. 6, no. 1, pp. 47–65, 1979, doi: 10.1068/b060047.

[5] M. Toy and G. Wichman. (1980). *Rogue*. Epyx, Coventry, U.K. Accessed: Dec. 1, 2020. [Online]. Available: https://store.steampowered.com/app/1443430/Rogue/

[6] D. Ashlock, C. Lee, and C. McGuinness, "Search-based procedural generation of maze-like levels," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 260–273, Sep. 2011, doi: 10.1109/TCIAIG.2011.2138707.

[7] J. Dormans, "Level design as model transformation: A strategy for automated content generation," in *Proc. 2nd Int. Workshop Procedural Content Gener. Games*, 2011, pp. 1–8, doi: 10.1145/2000919.2000921.

[8] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Aug. 2011, pp. 297–304, doi: 10.1109/CIG.2011.6032020.

[9] R. Der Van Linden, R. Lopes, and R. Bidarra, "Designing procedurally generated levels," in *Proc. AAAI Workshop*, 2013, p. 15.

[10] R. Lopes, T. Tutenel, R. M. Smelik, K. J. de Kraker, and R. Bidarra, "A constrained growth method for procedural floor plan generation," in *Proc. 11th Int. Conf. Intell. Games Simulation*, 2010, pp. 13–20.

[11] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. De Kraker, "Rule-based layout solving and its application to procedural interior generation," in *Proc. CASA Workshop 3D Adv. Media Gaming Simulation*, 2009.

[12] J. Taylor and I. Parberry, "Randomness + structure = clutter: A procedural object placement generator," in *Proc. Int. Conf. Entertainment Comput.*, Sep. 2011, pp. 424–427, doi: 10.1007/978-3-642-24500-8_57.

[13] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin, "Procedural generation of roads," *Comput. Graph. Forum*, vol. 29, no. 2, pp. 429–438, May 2010, doi: 10.1111/j.1467-8659.2009.01612.x.

[14] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proc. 28th Annu. Conf. Comput. Graph. Interact. Techn.*, 2001, pp. 301–308, doi: 10.1145/383259.383292.

[15] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin, "Procedural generation of villages on arbitrary Terrains," *Vis. Comput.*, vol. 28, nos. 6–8, pp. 809–818, Jun. 2012, doi: 10.1007/s00371-012-0699-7.

[16] G. Cordonnier, "Large scale terrain generation from tectonic uplift and fluvial erosion," *Comput. Graph. Forum*, vol. 35, no. 2, pp. 165–175, 2016, doi: 10.1111/cgf.12820.

[17] G. Cordonnier, E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie, and M.-P. Cani, "Authoring landscapes by combining ecosystem and terrain erosion simulation," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017, doi: 10.1145/3072959.3073667.

[18] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou, "Arches: A framework for modeling complex terrains," *Comput. Graph. Forum*, vol. 28, no. 2, pp. 457–467, Apr. 2009, doi: 10.1111/j.1467-8659.2009.01385.x.

[19] B. Rusnell, D. Mould, and M. Eramian, "Feature-rich distance-based terrain synthesis," *Vis. Comput.*, vol. 25, nos. 5–7, pp. 573–579, May 2009, doi: 10.1007/s00371-009-0332-6.

[20] K. Golubev, A. Zagarskikh, and A. Karsakov, "Dijkstra-based Terrain generation using advanced weight functions," *Procedia Comput. Sci.*, vol. 101, pp. 152–160, 2016, doi: 10.1016/j.procs.2016.11.019.

[21] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based terrain generation using diffusion equation," *Comput. Graph. Forum*, vol. 29, no. 7, pp. 2179–2186, Sep. 2010, doi: 10.1111/j.1467-8659.2010.01806.x.

[22] E. Michel, A. Emilien, and M.-P. Cani, "Generation of folded Terrains from simple vector maps," *Eurographics*, vol. 2015, pp. 77–80, May 2015, doi: 10.2312/egsh.20151019.

[23] G. A. Bradbury, I. Choi, C. Amati, K. Mitchell, and T. Weyrich, "Frequency-based controls for terrain editing," in *Proc. 11th Eur. Conf. Vis. Media Prod.*, 2014, pp. 1–10, doi: 10.1145/2668904.2668944.

[24] F.-X. Talgorn and F. Belhadj, "Real-time sketch-based terrain generation," in *Proc. Comput. Graph. Int.*, 2018, pp. 13–18, doi: 10.1145/3208159.3208184.

[25] J. Gain, P. Marais, and W. Straßer, "Terrain sketching," in *Proc. Symp. Interact. 3D Graph. Games*, 2009, pp. 31–38, doi: 10.1145/1507149.1507155.

[26] W. L. Raffe, F. Zambetta, and X. Li, "Evolving patch-based Terrains for use in video games," in *Proc. 13th Annu. Conf. Genetic Evol. Comput.*, 2011, pp. 363–370, doi: 10.1145/2001576.2001627.
[27] I. Antoniuk and P. Rokita, "Procedural generation of adjustable terrain for application in computer games using 2D maps," in *Proc. Int. Conf. Pattern Recognit. Mach. Intell.*, 2015, pp. 75–84, doi: 10.1007/978-3-319-19941-2_8.
[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680, doi: 10.3156/jsoft.29.5_177_2.
[29] É. Guérin, J. Digne, É. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive example-based terrain authoring with conditional generative adversarial networks," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 1–13, Nov. 2017, doi: 10.1145/3130800.3130804.
[30] R. J. Spick, P. Cowling, and J. A. Walker, "Procedural generation using spatial GANs for region-specific learning of elevation data," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2019, pp. 1–8, doi: 10.1109/CIG.2019.8848120.
[31] A. Wulff-Jensen, N. N. Rant, T. N. Møller, and J. A. Billeskov, "Deep convolutional generative adversarial network for procedural 3D landscape generation based on DEM," in *Proc. Interactivity, Game Creation, Des., Learn., Innov.*, 2018, pp. 85–94, doi: 10.1007/978-3-319-76908-0_9.
[32] Qiu, Yue, and Liu, "Void filling of digital elevation models with a terrain texture learning model based on generative adversarial networks," *Remote Sens.*, vol. 11, no. 23, p. 2829, Nov. 2019, doi: 10.3390/rs11232829.
[33] J. Klein, S. Hartmann, M. Weinmann, and D. L. Michels, "Multi-scale terrain texturing using generative adversarial networks," in *Proc. Int. Conf. Image Vis. Comput. New Zealand (IVCNZ)*, Dec. 2017, pp. 1–6, doi: 10.1109/IVCNZ.2017.8402495.
[34] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proc. Workshop Procedural Content Gener. Games*, 2010, pp. 1–8, doi: 10.1145/1814256.1814259.
[35] J. Olsen, "Realtime procedural terrain generation," Univ. Southern Denmark, Odense, Denmark, Tech. Rep. 2004.10.31, 2004. [Online]. Available: https://web.mit.edu/cesium/Public/terrain.pdf
[36] M. Frade, F. Fernandez de Vega, and C. Cotta, "Breeding terrains with genetic terrain programming: The evolution of terrain generators," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 1–13, Dec. 2009, doi: 10.1155/2009/125714.
[37] M. Frade, F. F. De Vega, and C. Cotta, "Evolution of artificial terrains for video games based on accessibility," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2010, pp. 90–99, doi: 10.1007/978-3-642-12239-2-10.
[38] M. Andereck, *Procedural Terrain Generation Based on Constraint Paths*. Columbus, OH, USA: The Ohio State Univ., 2014.
[39] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Declarative terrain modeling for military training games," *Int. J. Comput. Games Technol.*, vol. 2010, Jul. 2010, Art. no. 360458, doi: 10.1155/2010/360458.
[40] R. Smelik, K. Galka, K. J. De Kraker, F. Kuijper, and R. Bidarra, "Semantic constraints for procedural generation of virtual worlds," in *Proc. ACM Int. Conf. Proc. Ser.*, 2011, pp. 1–4, doi: 10.1145/2000919.2000928.
[41] R. Conroy-Dalton and N. Dalton, "OmniVista: An application for Isovist field and path analysis," in *Proc. 3rd Int. Space Syntax Symp.*, 2001.
[42] A. Van Bilsen and R. Poelman, "3D visibility analysis in virtual worlds: The case of supervisor," in *Proc. 9th Int. Conf. Construct. Appl. Virtual Reality (CONVR)*, 2009, p. 5.
[43] U. Pyysalo, J. Oksanen, and T. Sarjakoski, "Viewshed analysis and visualization of landscape voxel models," in *Proc. 24th Int. Cartographic Conf.*, Santiago, Chile, vol. 15, 2009, pp. 1–15.
[44] D. Fisher-Gewirtzman and I. A. Wagner, "Spatial openness as a practical metric for evaluating built-up environments," *Environ. Planning B, Planning Des.*, vol. 30, no. 1, pp. 37–49, Feb. 2003, doi: 10.1068/b12861.
[45] P. P.-J. Yang, S. Y. Putra, and W. Li, "Viewsphere: A GIS-based 3D visibility analysis for urban design evaluation," *Environ. Planning B, Planning Des.*, vol. 34, no. 6, pp. 971–992, Dec. 2007, doi: 10.1068/b32142.
[46] N. S. Dalton, R. C. Dalton, P. Marshall, I. Peverett, and S. Clinch, "Three dimensional isovists for the study of public displays," in *Proc. 10th Int. Space Syntax Symp.*, 2015, pp. 13–17.
[47] A. Rodriguez, D. B. Ehlenberger, P. R. Hof, and S. L. Wearne, "Rayburst sampling, an algorithm for automated three-dimensional shape analysis from laser scanning microscopy images," *Nature Protocols*, vol. 1, no. 4, pp. 2152–2161, Nov. 2006, doi: 10.1038/nprot.2006.313.
[48] A. Pech, "Evolving gameplay elements into virtual terrains," Ph.D. dissertation, Edith Cowan Univ., Joondalup WA, Australia, 2018.
[49] M. Newman, *Networks: An Introduction*. Oxford, U.K.: Oxford Univ. Press, Mar. 2010.
[50] S. R. Lay, *Convex Sets*. New York, NY, USA: McGraw-Hill, 1964.
[51] N. Metropolis and S. Ulam, "The Monte Carlo method," *J. Amer. Stat. Assoc.*, vol. 44, no. 247, pp. 335–341, Sep. 1949, doi: 10.1080/01621459.1949.10483310.
[52] G. Peter Lepage, "A new algorithm for adaptive multidimensional integration," *J. Comput. Phys.*, vol. 27, no. 2, pp. 192–203, 1978, doi: 10.1016/0021-9991(78)90004-9.
[53] *Savage: The Battle for Newerth*, S2 Games, Kalamazoo, MD, USA, 2003.
[54] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[55] J. Cohen, "A coefficient of agreement for nominal scales," *Educ. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, Apr. 1960, doi: 10.1177/001316446002000104.
[56] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, p. 159, Mar. 1977, doi: 10.2307/2529310.
[57] K. Perlin, "An image synthesizer," *ACM SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, Jul. 1985, doi: 10.1145/325165.325247.

ANDREW PECH received the B.Sc. (Hons.) and Ph.D. degrees in computer science from Edith Cowan University, Perth, WA, Australia, in 2013 and 2018, respectively.

Since 2019, he has been a Software Developer with Micromine, Intuitive Mining Solutions, Nedlands, WA, Australia. His research interest includes procedural content generation for computer games. He was a recipient of the Australian Postgraduate Award Scholarship from the Australian Government.

CHIOU PENG LAM received the Ph.D. degree in computer vision from Curtin University, Perth, WA, Australia, in 1995.

From 1995 to 1997, she was a Postdoctoral Research Fellow with Curtin University. In 1997, she was a Lecturer with Murdoch University, WA, Australia. She is currently an Associate Professor in software engineering with the School of Science, Edith Cowan University. She has more than 110 refereed publications to date. Together with her collaborators, she has developed computational intelligence-based techniques for addressing problems in data mining, defense, bioinformatics, and search-based software testing. Her research interests include pattern recognition, evolutionary algorithms, and software testing.

MARTIN MASEK (Member, IEEE) received the B.E. degree in information technology engineering and the Ph.D. degree in computer vision from The University of Western Australia, Perth, WA, Australia, in 1998 and 2004, respectively.

From 2003 to 2005, he was an Associate Lecturer with the School of Electrical, Electronic, and Computer Engineering, The University of Western Australia. In 2005, he joined as a Lecturer of computer science with Edith Cowan University, Perth, where he was promoted to a Senior Lecturer in 2008, where he has been an Associate Professor since 2018. His research interests include the development of computer vision, artificial intelligence, and computer game technology to solve problems in areas, such as health, education, and defense.