A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms

William L. Raffe, Fabio Zambetta, and Xiaodong Li
School of Computer Science and Information Technology
RMIT University
Melbourne 3001, Australia
Email: {william.raffe, fabio.zambetta, xiaodong.li}@rmit.edu.au

***Abstract*—This paper provides a review of existing approaches to using evolutionary algorithms (EA) during procedural terrain generation (PTG) processes in video games. A reliable PTG algorithm would allow game maps to be created partially or completely autonomously, reducing the development cost of a game and providing players with more content. Specifically, the use of EA raises possibilities of more control over the terrain generation process, as well as the ability to tailor maps for individual users. In this paper we outline the prominent algorithms that use EA in terrain generation, describing their individual advantages and disadvantages. This is followed by a comparison of the core features of these approaches and an analysis of their appropriateness for generating game terrain. This survey concludes with open challenges for future research.**

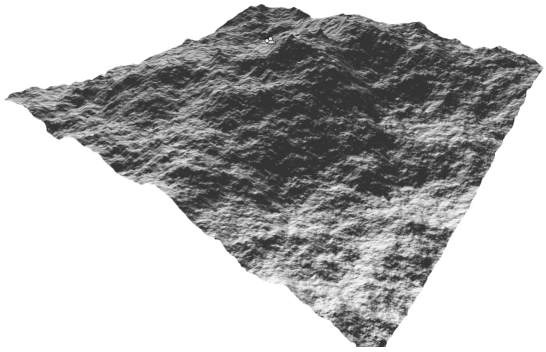


Figure 1. A terrain generated through a fractal subdivision method

I. Introduction

Procedural terrain generation (PTG) has been successfully applied in many fields and has enjoyed ongoing research by both industry leaders and academics for almost three decades [1]. Most existing research has been devoted to creating vast sprawling landscapes that are typically too large and too detailed to be created manually. These methods have been improved in recent years and powerful commercial tools, such as *Terragen* (Planetside Software), are capable of producing renderings of detailed terrain that resemble professional photographs of landscape.

One of the most prominent applications for the Procedural Terrain Generation is for use in video games [2]. To cater to players who enjoy a wide variety of experiences and exploring new environments, game developers can use PTG techniques to provide players with large terrains to play on, as can be seen in games such as *Worms 2* (Team17, 1997), *Civilization V* (Firaxis Games, 2010), *Minecraft* (Markus Perrson, 2009). Alternatively, a PTG system can also be used as a creative aid for designers or to simply streamline the map creation process by providing a base terrain for developers to work with, as in the game *Darwinia* (Inversion Software, 2005).

One of the oldest and most common PTG techniques is that of Fractal Subdivision [3] in which the resolution of a terrain is recursively increased and the height of each new point on the terrain is adjusted. A typical fractal terrain is shown in Figure 1. These fractal techniques have catalyzed research interest over many years as much as terrain simulation techniques (e.g. Erosion Simulation [4]). However, both of these techniques provide little control over the generation process, such as the layout of terrain features (mountains, valleys, cliffs, etc.), and in recent years there has been a push towards techniques with increased control. This has predominantly been achieved by increasing the number of parameters in the generative algorithm [5], [6], by allowing the user to provide a basic sketch of feature layout [7], [8], or by allowing users to quickly paint on height values [9].

More recent techniques have used Evolutionary Algorithms (EA) to drive the terrain generation process. These techniques fall into the broader field of *Search-Based Procedural Content Generation* (SBPCG), in which search algorithms (primarily EA) are used to procedurally generate a wide variety of game content. An introduction to SBPCG and a taxonomy of some approaches can be found in [10], [11]. While other techniques, such as fractals, are well established, their random output and the style of terrain generated means that there are few game types that can utilize them. By using EA, more control can be exerted over the generation process, allowing developers to get more desirable results or for maps to be automatically generated to meet player preferences, as was recently done in the procedural generation of virtual race tracks [12].

The rest of this paper is organized as follows: in Section II we first state requirements for terrain to be useful in games followed by summaries of six algorithms that use EA in PTG, along with advantages and disadvantages of each of the approaches. In Section III we provide more direct comparisons between the different algorithms, starting in Section III-A with an analysis of visual results and then in Section III-B with a

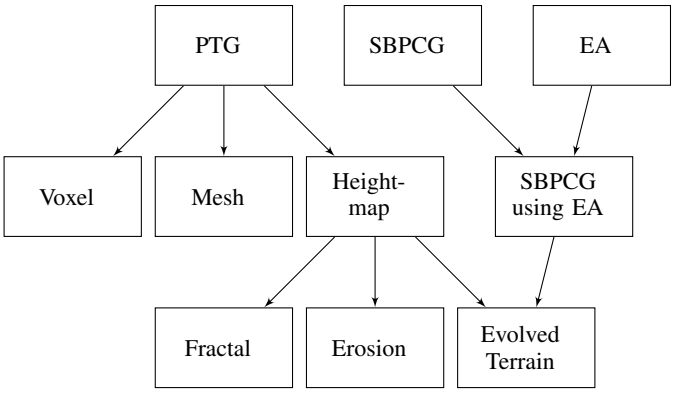


Figure 2. Relationship between the fields of PTG, SBPCG, and EA.

comparison of features and capabilities. Finally, in Section IV we state some areas for improvement within this growing field.

II. Evolutionary Approaches to Terrain Generation

The use of evolutionary techniques for PTG is an emerging field, with the first research on this topic being carried out only few years ago and only a handful of approaches attempted since. Figure 2 shows the relationship between the fields of PTG and evolutionary algorithms and shows how the techniques in this section relate to other techniques in PTG. It should be noted that all of the papers reviewed in this section use *height-maps*, which represent the terrain as a two dimensional array of height values and can be rendered as a mesh. Height-maps are popular due to their computational efficiency and their ease of use in complex algorithms. The use of EA in terrain generation is not limited to using height-maps but, to the best of our knowledge, there are no existing techniques that use other terrain representations, such as voxels or triangulated irregular networks.

In this section we first outline some basic requirements for terrain to be useful in games that will be used extensively when analyzing the different approaches to be reviewed. This is followed by description of known techniques that use EA during a PTG process.

A. Requirements for Terrain in Games

The genre of the game that a piece of virtual terrain is to be used in will dictate the terrain features. Olsen [13] outlines and describes some requirements in order for a PTG technique to be useful in the Real Time Strategy genre. Primarily, Olsen states that the terrain must contain enough flat areas for game characters to move on realistically, as well as having a high connectivity such that players can traverse as much of the map as possible. However, the terrain also needs hills, cliffs, valleys and other features to promote strategic game-play and to make maps visually appealing. Though those criteria are true for many game genres, each genre will still have their own requirements for scale and feature arrangement. For example, a First Person Shooter game, e.g. *Call of Duty: Modern Warfare 3* (Infinity Ward, 2011), will typically have small scale maps, containing high densities of terrain features and virtual objects to impede line of sight, while Role Playing Games, e.g. *The Elder Scrolls V: Skyrim* (Bathesda Game Studios, 2011), typically have large environments with dispersed townships, thus encouraging exploration of an expansive world. Further more, some game genres, such as Flight Simulators, have no need for the same requirements of traversability but rather require the terrain to appear realistic from an aerial perspective. However, as these types of terrain can be generated well by existing fractal algorithms, in this paper we focus on games that have similar terrain requirements as those laid out by Olsen [13].

B. Algorithm Summaries

In this section, descriptions of six approaches to applying EA to PTG are provided, along with general advantages and disadvantages of each. To the best of our knowledge, these six algorithms are the only publications in this field and each group of authors attempts to use EA in PTG in different ways, thus highlighting the infancy and the potential for growth in this field. As not all of the resulting system have been given titles, we label each approach by the authors' names.

1) Ong et al. [14][15]: The first known proposal of using EA in PTG was put forth by Ong et al. [14] and their work culminated in a terrain generation program called *Terrainosaurus*, implemented by Saunders [15]. Their algorithm consists of two phases: the first is to generate a 2D outline of the terrain by distorting a user-provided sketch of the terrain boundaries. The second phase constructs the height-map, with the initial population being created from sample height-map data provided by the user. In order to manipulate the terrain, numerous control points are selected on the surface of the terrain. The genotype representation here is a list of operators for each control point, such as raising, lowering, and rotating the point. These control points also have an area of influence around them, thus the operators will affect surrounding vertices of the height-map to a lesser degree. One-point crossover is used, whereby parents exchange parts of their operator lists. For the fitness function in this phase, the authors recommend measuring how similar the resulting terrain is to the sample terrains provided by the user, thus enforcing the behavior of candidate solutions resembling the initial population.

Advantages and Disadvantages: The *Terrainosaurus* algorithm is useful for making a family of similar terrains. Thus, if an existing database of game maps for a specific genre can be acquired then they can be used as sample terrains and this would be an effective way to ensure that the maps that are generated meet the requirements of the genre. It can also be used after the game is released to provide slight variations to map layouts. This is beneficial in genres such as multiplayer first person shooters where exploration of new maps is not desirable and players prefer to learn maps in-depth and develop strategies. Thus, slight variations to a map would require a player to alter their strategies without requiring them to explore an entirely new map. However, this advantage can also

be a disadvantage if no appropriate samples can be provided, as it would be unlikely that a suitable terrain will be generated.

2) Ashlock et al. [16]: Ashlock et al. apply EA to L-systems in order to generate fractal style terrains [16]. They use an L-System that expands in two dimensions such that if each symbol is translated into a square then an even grid of vertices for a height-map is produced. A displacement value is assigned to each symbol in the L-System grammar so that when an expansion occurs, the new vertices are given height values that are modified by the displacement values of the symbols that they border. The authors use EA to find an appropriate set of replacement rules and displacement values. Two genotype representations are used; the first is a string of replacement rules for all of the symbols in the grammar and the second is a list of height displacement values, again one for each symbol. A two-point crossover method is used for both genotypes and the fitness function used in this algorithm is a comparison to a target terrain provided by the user, measuring the difference in height of each height-map vertex between a candidate and the target.

Advantages and Disadvantages: The results provided by Ashlock et al. show that an L-System can be effective at generating fractal style terrains and in later papers these results are improved further. However, the disadvantage, in respect to this algorithm being used in games, is that it is not known whether this algorithm can produce terrains that differ from typical fractal terrains and that do not exhibit the symmetrical qualities that are shown in results provided by Ashlock et al. This L-System approach also suffers from the same problem as Ong et al. where if an appropriate target terrain cannot be provided then the system cannot perform well and even if a target terrain is available, all resulting candidates will be closely related and the solution space will not be explored.

3) Walsh and Gade [17][18]: In their approach, Walsh and Gade use EA to automatically adjust parameter values of a terrain generator [17] . The terrain parameters that are evaluated include feature scale, feature spikiness, and water level, as well as some atmospheric features, namely sun angle and cloud coverage. It should be noted that this approach does not generate an entirely new terrain; rather it adjusts the features of an existing one. The genotype representation in Walsh and Gade's approach is a group of parameter values, each being an 8 bit string. Mutation is in the form of flipping a single bit in one or more of the strings, while one-point crossover is used to swap segments of two parents' bit strings for one parameter. An interactive EA is used in which the fitness is judged purely by the user. A tournament selection process is then used to decide which members of the current generation will be the parents of the next, giving a higher probability of selection to candidates the user has chosen. Walsh and Gade primarily use crossover operations, allowing for fast convergence towards a desirable terrain. Walsh and Gade have recently investigated a fitness function for their approach that uses a measurement based on the ratio between order and complexity of an image of a candidate terrain to judge the aesthetic qualities of the terrain [18].

Advantages and Disadvantages: The current implementation of Walsh and Gade's algorithm has limited applicability to games. This is because adjusting parameters such as atmospheric effects may provide a visual variety for player but will probably have very little impact on the entertainment value of a game map and may actually distract the player. Also, adjusting parameters such as feature scale require a pre-designed map to be available and the parameters used by the authors may not provide much variety in strategic game-play or player exploration. However, Walsh and Gade's idea of applying EA to parametrically controlled procedural generation techniques holds a lot of potential if a suitable PTG is used. For example, EA could be applied to work by Kamal and Uddin [5], who present a parameter controlled fractal generation method, or Doran and Parberry [6], who use highly parametrized cellular automata to generate island terrain.

4) Frade et al. [19][20][21][22]: Frade et al. used genetic programming to create height functions in an algorithm they named *GenTP* [19]. A height function is an equation that is applied to the value of each vertex in an existing height-map to produce a new height-map. The genotype representation in this approach is a tree of operators and is evolved by adding, removing, or substituting operators through genetic programming. The operators can be any basic numerical operator, trigonometric function, or custom made terminal function. The terminal functions create a base height-map for the rest of the height function to operate on and in early works by Frade et al. [19] these were stochastic noise algorithms that provided different base height-maps with every call. However, in later work [20] the terminal functions were changed to repeatable number sequence generators that would allow for multiple calls to the same height function to result in the same terrain. The authors initially use an interactive evolution approach, however their later work investigates two fitness functions to generate terrain for use in games. The first was an *Accessibility* measurement [20] which favored candidate terrains that have large quantities of flat terrain for the player to traverse and for separated areas of flat terrain to have at least one path between them. The authors later introduced the *Obstacle Edge Length* measurement [21], which would ensure that there were obstacles in the terrain for the player to navigate around.

Advantages and Disadvantages: The advantages of the *GenTP* algorithm is that the use of three different types of mutation (adding, removing, and substituting segments of the operator trees) leads to a good exploration of the solution space while also doing well to prevent the height function from becoming too long. In earlier works [19], it was possible to generate a family of similar terrains through multiple calls to the same height function. Also, the generated terrains were visually intriguing and possibly useful as a development aid for level designers. However, these earlier results do not have much flat and connected terrain, limiting the genre they are applicable to. Meanwhile, terrains that are generated through the use of the two fitness functions allow for players to traverse much of the map and the authors have used this approach in the development of the game *Chapas* [22]. However, many of the

results provided are overly flat and exhibit fairly predictable patterns, which also limits their applicability to many game genres.

5) Togelius et al. [23]: The creation of terrain for use in video game can be seen as an inherently multi-objective problem, balancing between interesting games and fair games. Thus, Togelius et al. propose the use of a *Multi-Objective Evolutionary Algorithm* (MOEA) in the creation of maps for Real-Time Strategy (RTS) games [23]. In their algorithm, Togelius et al. create terrain as a sub-component of the overall goal of generating a complete and playable game map, which also includes the placement of game-play elements and objectives. To generate the terrain itself, Togelius et al. use a flat height-map from which mountain peaks are raised out of and ridges are constructed along Gaussian curves. The genotype for the terrain is represented by five values: two for the standard deviations of the Gaussian distribution, two for x and y coordinates of the mountain peak and one that adjusts the height of the mountain. Multiple fitness functions are used that attempt to promote balanced game-play by favoring maps with dispersed placement of player bases, equal access to resources from each player, and that have traversable paths between bases. For the genetic operators, the authors use probability based mutation and simulated binary crossover (SBX), which suggests that childrens' genetic strings closely resemble one parent or the other.

Advantages and Disadvantages: Initial results from Togelius et al. show promise in the generation of complete maps for RTS games and the authors have successfully applied this approach to the well known game *StarCraft* (Blizzard Entertainment, 1998) where terrain features are represented by two dimensional textures of water and rock rather than raised and lowered height-map data [24]. However, compared to many of the other algorithms demonstrated in this paper, the terrain that is produced in their initial algorithm that uses height-maps contains only basic detail, due to the genotype representation restricting the style of features to soft and rounded looking peaks.

6) Raffe et al. [25]: Recently, Raffe et al. applied EA in generating terrain that is constructed of patches of smaller terrain height-maps [25]. In their algorithm, sample terrains are decomposed into smaller, uniform sized patches, which can then be recombined to make new candidate terrains. The genotype representation in the approach by Raffe et al. is a two dimensional array of patch identification numbers. A uniform crossover mechanism is used where a child duplicates one of the parent's genetic structure and then each patch is given a probability of being switched for the patch in the same position on the other parent. Mutation is similar, giving each patch a probability of being switched with a randomly chosen patch. For fitness evaluation, Raffe et al. use a two-leveled interactive evolution mechanism in which the user first selects parents and then selects which patches in those parents they would like to exclude from genetic operations.

Advantages and Disadvantages: The results provided by Raffe et al. [25] show how by combining multiple runs of their algorithm with different parameter settings, a user with little experience can create terrain similar to a map from the game *Halo* (Bungie, 2001). This is aided by the Two-Level interactive evolution system that retains desirable features in a parent and focuses mutation onto undesirable features. However, due to the patches being extracted from user provided sample terrains, the exploration of the solution space of possible terrains would be limited by the patches that are available for use. Also, due to the uniform grid based patching approach, some features, such as the triangular shape of a peninsula, can be difficult to achieve, especially if the feature isn't present in any of the sample terrains. This is primarily due to the square shape of the patches and can only be overcome by using smaller patch sizes.

III. Discussion

A. Visual Comparisons

Figure 3 shows visual results of five out the six primary approaches that have been reviewed. Note that each author has used a different rendering algorithm and so when comparing these figures to each other we should exclude texturing or lighting effects and focus on the shape and distribution of terrain features.

Figure *3*a shows the results produced by the *Terrainosaurus* program by Saunders [15], which is improved upon earlier work with Ong et al. [14]. The terrain is divided into three areas, each with a different terrain type that has been generated using different sample terrains. The terrain that is produced relies on the sample terrains provided to the program and here the authors have used low resolution satellite imagery, resulting in soft terrain features. Terrain is smoothed even more by the way in which the circular area of effect around the genetically modified control points are raised or lowered.

Unfortunately, no figure could be acquired from the work of Ashlock et al. on evolvable L-systems for terrain generation [16]. However, because their approach is similar to a fractal approach, Figure 1 can be seen as an approximation of their results, with one of the primary differences being that the author's focus on terrains with a dominant single feature in the middle of the map. For more accurate results, please see [16].

Figure *3*b shows two terrains produced by Walsh and Gade's algorithm [17]. These two terrains are from the same base terrain but have visibly different feature height, feature noise, and sun angle parameter settings. This image shows how the parameters chosen by Walsh and Gade lead to aesthetically different terrain but would not result in many differences in game-play.

Figure *3*c and *3*d are from the works of Frade et al. Figure *3*c is from their earlier work that uses interactive evolution [19] and shows that, while the terrain is visually appealing it would be difficult to incorporate into a game without designer intervention. Figure *3*d is from Frade et al. later work with the obstacle length and accessibility fitness functions [21][20] and shows how the terrain is much more useful for many game genres due to the increased quantity of playable surfaces. but

also how it may suffer from overfitting the fitness function and resulting in flat, featureless terrain.

Figure *3*e shows a game map by Togelius et al. [23]. The map contains not only terrain features but also circular markers indicating game-play assets such as resources and player bases. This map is immediately playable and the map layout has been optimized to provide a balanced game for all players of a Real-Time Strategy game. However, this figure also shows how the terrain features are limited to softly rounded hills and mountains.

Finally, figure *3*f was produced through multiple runs of the program created by Raffe et al. [25]. Their algorithm is capable of a variety of terrain features, from steep cliffs, rolling hills, and smooth playable surfaces. However, the approach is dependent on what terrain patches are provided to the system and struggles to produces sharp angled edges such as the cliff peninsula on the left side of the image.

B. Feature Comparisons

Table I provides a condensed summary of the approaches reviewed in this survey. As there is no standardized metric for evaluating these types of PTG algorithms, the results in Table I are stated through our own opinions, with explanations and justifications given in this section. No time efficiency values are provided in this table because not all of the published papers provide reliable performance test data.

The columns of Table I list the capabilities by which the approaches are compared. The first two columns list the approach and a synopsis of the fitness function used. The *Refinement* column indicates whether an algorithm is good at creating a family of similar terrains (exploiting the solution space), while the *Variety* column evaluates how well an algorithm can generate substantially different terrains (exploring the solution space). The column labeled *Control* refers to how well the system can be steered by user criteria. In other words, we examine how easy it is for a user to get a desirable result from the system, especially with regards to the arrangment of terrain features. In the *Game Integration* column we give a judgment on how easy it would be to incorporate the approach, as they exist now, into a game. This is then followed by the *Ideal Use* field that states what applications each approach would be most suited for in interactive media.

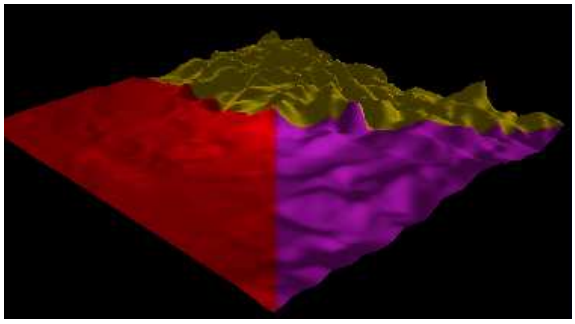
The most suitable approaches for *Refinement* of terrains are those by Ong et al., Walsh and Gade, Togelius et al., and Raffe et al. The approach by Ong et al. [14] is seeded by the sample terrains, with mutations only being the raising and lowering of selected points on the height-map, while the fitness functions used encourage candidate terrains that do not diverge much from the samples. The parameter-based technique by Walsh and Gade [17] is also seeded by an initial terrain and narrows the refinement of the terrain to feature scale, rigidity, and water level, none of which affect the overall shape of a terrain. The use of SBX crossover means that the technique by Togelius et al. [23] favors slight changes in candidate terrains from one generation to the next. Meanwhile, the two-level parent selection mechanism used by Raffe et al. [25], coupled with the patch size parameter, allows for the user to specify small individual patches of terrain to be mutated. Both the technique by Frade et al. [19] and the L-system algorithm by Ashlock et al. [16] do not perform as well in Refinement because, while they are good at generating families of similar terrains, it is not feasible to make minor alterations to an existing terrain.

Not many of the techniques perform well for *Exploration* of the solution space. The best techniques for this are the early approaches by Frade et al. in which the near infinite combinations of function operators can result in some highly varied terrains. However, this benefit appears to be reduced in later works as the candidate terrains become more flat. Both the technique by Togelius et al. and that of Raffe et al. allow for good exploration of their individual solution space but are limited in exploring the overall solution space due to their genotype representation restricting the types of terrain features that can be produced. The approaches by Ong et al., Walsh and Gade, and Ashlock et al. all use an idealized terrain, either for seeding or to drive the fitness function, and are therefore limited in their ability to deviate from that style of terrain.

The algorithms presented by Walsh and Gade and by Raffe et al. score the highest in the area of *Control*. This is primarily due to their use of interactive evolution that gives the user full control over the feature layout or appearance in the respective genotype spaces. Ong et al. and Ashlock et al. use a fitness function that compares candidate terrains to a target terrain and thus if a target terrain exists a user can generate a suitable feature layout. However, as mentioned before, if a target terrain cannot be provided, the user's ability to generate a suitable terrain is limited. The early approaches by Frade et al. also use interactive evolution, however, a small mutation in the genotype height function can result in extensive changes to the phenotype terrain, which means it can be quite difficult for a user to start a run with a desired goal in mind and for them to achieve it. On the other hand, the technique from Togelius et al. evolves towards terrains that provide specific game-play but precise control over terrain features is limited.

When analyzing the feasibility of integrating these approaches into games, it is important to recognize that Frade et al. and Togelius et al. are in fact the only authors to have their terrains tested in an actual game. The difference between these approaches however is that Frade et al. only generate the terrain, requiring a designer to populate the world with virtual objects, while Togelius et al. include the placement of game-play objectives as part of the map generation process and thus require very little intervention from a designer before being used in a game. It has been shown that the patch-based approach by Raffe et al. is capable of generating terrain similar to that used in commercial games, however these terrains have not been tested in an actual game and the use of interactive evolution requires designer interaction. While we can imagine ways that systems by Ong et al., Ashlock et al., and Walsh and Gade can be extended for use in games, no research has yet been conducted to show the feasibility of this.

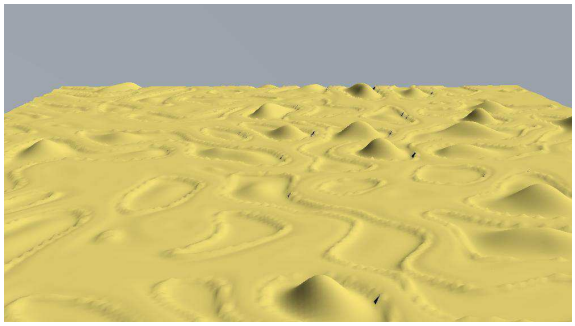
The Ideal Use column does not indicate the original authors' intention for the use of their algorithm but rather how we feel



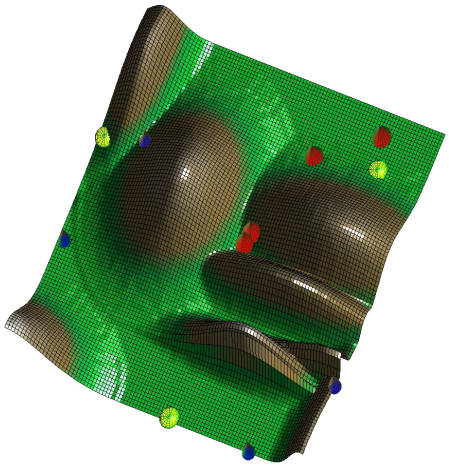
(a) Ong et al. and Saunders [15]

(b) Walsh and Gade [17]

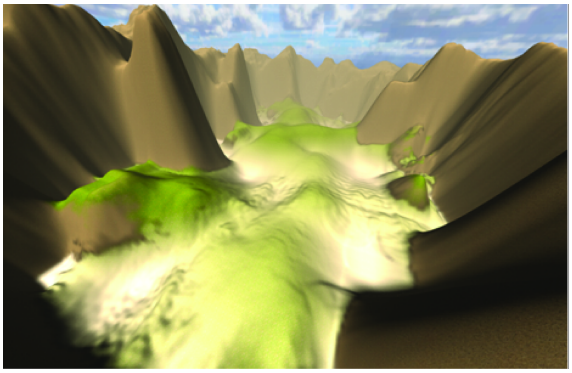
(c) Frade et al. [19]



(d) Frade et al. [21]



(e) Togelius et al. [23]



(f) Raffe et al. [25]

Figure 3. Visual results from five of the reviewed papers. All images used with permission from the respective authors.

it can be used in relation to games and other interactive media. The technique by Togelius et al. is currently the most suitable for use in games due to a focus on the real-time strategy genre of games. Both the height function and patch-based techniques by Frade et al. and Raffe et al. are targeted at being used in games, however both techniques would benefit from a genre focus so that the algorithm, primarily the fitness evaluation method, can be designed to make maps for certain types of games. Finally, the approaches used by Ong et al., Ashlock et al., and Walsh and Gade create large terrains that may only be suitable for genres such as flight simulators, however it is currently acceptable in these types of games to use basic fractal terrain generation methods.

IV. Open Challenges

Through the analysis of the existing techniques we were able to identify future challenges for the field of PTG using EA. Three of these challenges are to:

- Develop a genotype representation that can explore much of the solution space for a chosen game genre.

Table I
ALGORITHM COMPARISON

Approach	Fitness Evaluation	Refinement	Variety	Control	Game Integration	Ideal Use
Ong et al. [14], [15]	Compared to example terrains.	High	Low	Medium	Low	Simulated natural terrain. Could be used in games such as flight simulators that need large, natural terrain.
Ashlock et al. [16], [26]	Compared to idealized terrain.	Medium	Low	Medium	Low	Where single feauture terrains and fractal terrains are applicable. Simulation applications.
Walsh and Gade [17]	Interactive evolution.	High	Low	High	Low	Evolutionary art where a single screen capture is more desirable then a playable game.
Frade et al. [27] - [22]	Interactive evolution. Accessibility metric. Obstacle length metric.	Medium	High	Low	Medium	Early approaches for evolutionary art or games with eccentric terrain. Later appraoches for games that require prodominately flat terrain.
Togelius et al. [23], [24]	Multiobjective evolution for base and resource distances and asymmetry of terrain.	High	Medium	Low	High	Real-time strategy games that use player bases and collectable resources.
Raffe et al. [25]	Two-leveled interactive evolution.	High	Medium	High	Medium	As a development aid for game maps of all sizes.

- Further investigate fitness evaluation methods for PTG, with more emphasis placed on techniques that evaluate the map based on how the player interacts with it.
- Investigate metrics to allow for direct comparisons between PTG techniques.

All of the techniques mentioned in this survey paper have very different genotype representations. As of the writing of this paper, none of these genotype representations have yet been proven to be substantially better than the others; they each come with their own advantages and disadvantages and no technique has yet shown to be reliable enough to be used in commercial games. For a representation to be useful to game developers it should be able to generate a variety of terrains, at least within the same game genre, and also allow for small, controlled changes to be made to terrains. Having both of these features would allow players to experience entirely new maps as well as refined or slightly modified version of their favorite maps. With no existing techniques being able to achieve this delicate balance of features yet, the primary challenge in coming years will be to find a suitably powerful genotype representation and we expect to see the emergence of new approaches as well as improvements to existing ones.

Another important challenge is to find an appropriate fitness evaluation method. Nearly half of the approaches mentioned in this survey have used interactive evolution in early prototypes. However, the uses for such a system are limited to development aids and we believe that the future of this field lies within automating the fitness evaluation process. Togelius et al. [11] describe three categories of automated fitness evaluation methods for Search-based Procedural Content Generation (SBPCG). From these three categories we believe that the most useful for PTG algorithms would be a combination of a *Direct* fitness function that measure the terrain itself on an appropriate metric, as can be seen with the accessibility measure used by Frade et al., and *Interactive* fitness functions that utilizes data on how a player interacts with the terrain when it is used in a game. The combination of these two types of fitness measures should ensure that maps are playable and, once they have been evaluated in a game, evolve towards being more entertaining for players.

A major limitation of all PTG research is a lack of solid metrics to analyse the performance of individual algorithms. The most common methods for reporting PTG research is to present visual results, which are influenced by the rendering technique used, and time performance results. Smith and Whitehead have suggested the use of an *Expressive Range* metric for two dimensional level generators [28]. In this approach, an algorithm is tested to see how much of a two dimensional solution space can be generated and how often each area of the solution space is likely to be visited. The solution space here is defined by two or more custom metrics, for example the amount of flat area on a terrain and the connectivity of the terrain, and is independent of the fitness measures used. Thus, if appropriate metrics can be found, the Expressive Range technique can be used for PTG algorithms and may even be extended for use with many *Search-based Procedural Content Generation* algorithms.

V. CONCLUSION

Evolutionary algorithms (EA) can add control to the process of procedural generation of terrain (PTG), providing an alternative to the typically stochastic traditional approaches. This allows for not only a wide variety of terrains to be generated but also for families of similar terrains to be created and refined. Many PTG techniques for creating game maps aim at automating the terrain generation process to provide players a wide range of maps to play on. EA techniques have been applied to the generation of other game content with success

and have demonstrated how they can be used to tailor content to individual players and thus its use in PTG should be further investigated.

There are currently six known approaches to using EA in PTG, each one with its own advantages and disadvantages. While none of these approaches has yet been able to generate suitable terrain for use in games, each of them provides new ideas and highlights new challenges to overcome. The main challenge facing this body of research is to find a robust genotype representation, capable of working well with an EA and also of being translated into a detailed terrain. The second most important challenge is to find a fitness evaluation method, focused on the use of generating terrain for games. Finally, further investigation will need to be undertaken into metrics that can be used by authors to better evaluate their PTG algorithms.

ACKNOWLEDGMENT

The authors would like to thank R. Saunders, P. Walsh, M. Frade, and J. Togelius for their permission to use figures from their respective papers.

REFERENCES

[1] R. Smelik, K. de Kraker, S. Groenewegen, T. Tutenel, and R. Bidarra, "A Survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009.

[2] A. Doull. (2008) Death of the Level Designer: Procedural Content Generation in Games. ASCII Dreams. [Online]. Available: http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html

[3] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, no. 6, pp. 371–384, 1982.

[4] F. Musgrave, C. Kolb, and R. Mace, "The synthesis and rendering of eroded fractal terrains," in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. ACM, 1989, pp. 41–50.

[5] K. Kamal and Y. Uddin, "Parametrically controlled terrain generation," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. ACM, 2007, p. 23.

[6] J. Doran and I. Parberry, "Controlled procedural terrain generation using software agents," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 111–119, 2010.

[7] H. Zhou, J. Sun, G. Turk, and J. Rehg, "Terrain synthesis from digital elevation models," *IEEE Transactions on Visualization and Computer Graphics*, pp. 834–848, 2007.

[8] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based terrain generation using diffusion equation," in *Computer Graphics Forum*, vol. 29, no. 7. Wiley Online Library, 2010, pp. 2179–2186.

[9] G. de Carpentier and R. Bidarra, "Interactive GPU-based procedural heightfield brushes," in *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009, pp. 55–62.

[10] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne, "Search-based procedural content generation," *Applications of Evolutionary Computation*, pp. 141–150, 2010.

[11] ——, "Search-based Procedural Content Generation: A Taxonomy and Survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, no. 99, pp. 1–1, 2011.

[12] J. Togelius, R. De Nardi, and S. Lucas, "Towards automatic personalised content creation for racing games," in *IEEE Symposium on Computational Intelligence and Games (CIG)*. IEEE, 2007, pp. 252–259.

[13] J. Olsen, "Realtime procedural terrain generation," *Department of Mathematics And Computer Science (IMADA) University of Southern Denmark*, 2004.

[14] T. Ong, R. Saunders, J. Keyser, and J. Leggett, "Terrain generation using genetic algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2005, pp. 1463–1470.

[15] R. Saunders, "Realistic terrain synthesis using genetic algorithms," Ph.D. dissertation, Citeseer, 2006.

[16] D. Ashlock, S. Gent, and K. Bryden, "Evolution of l-systems for compact virtual landscape generation," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3. IEEE, 2005, pp. 2760–2767.

[17] P. Walsh and P. Gade, "Terrain generation using an Interactive Genetic Algorithm," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–7.

[18] ——, "The use of an aesthetic measure for the evolution of fractal landscapes," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 1613–1619.

[19] M. Frade, F. F. de Vega, and C. Cotta, "Breeding terrains with genetic terrain programming: the evolution of terrain generators," *International Journal of Computer Games Technology*, 2009.

[20] ——, "Evolution of artificial terrains for video games based on accessibility," *Proceedings of the European Conference on Applications of Evolutionary Computation*, vol. 6024, pp. 90–99, 2010.

[21] M. Frade, F. de Vega, and C. Cotta, "Evolution of artificial terrains for video games based on obstacles edge length," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[22] M. Frade, F. F. de Vega, and C. Cotta, "Development of Chapas an open source video game with genetic terrain programming," in *VII Congreso Espanol sobre Metaheuristicas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, 2010.

[23] J. Togelius, M. Preuss, and G. Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, pp. 1–8.

[24] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 265–272.

[25] W. Raffe, F. Zambetta, and X. Li, "Evolving patch-based terrains for use in video games," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 363–370.

[26] D. Ashlock, S. Gent, and K. Bryden, "Embryogenesis of artificial landscapes," *Design by Evolution*, pp. 203–221, 2008.

[27] M. Frade, F. F. de Vega, and C. Cotta, "Modelling video games' landscapes by means of genetic terrain programming: a new approach for improving users' experience," in *Proceedings of the 2008 conference on Applications of evolutionary computing*. Springer-Verlag, 2008, pp. 485–490.

[28] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 4.