Evolving Patch-based Terrains for use in Video Games

William L. Raffe
william.raffe@rmit.edu.au

Fabio Zambetta
fabio.zambetta@rmit.edu.au

Xiaodong Li
xiaodong.li@rmit.edu.au

School of Computer Science and Information Technology
RMIT University
Melbourne, Australia

ABSTRACT

Procedurally generating content for video games is gaining interest as an approach to mitigate rising development costs and meet users' expectations for a broader range of experiences. This paper explores the use of evolutionary algorithms to aid in the content generation process, especially the creation of three-dimensional terrain. We outline a prototype for the generation of in-game terrain by compiling smaller height-map patches that have been extracted from sample maps. Evolutionary algorithms are applied to this generation process by using crossover and mutation to evolve the layout of the patches. This paper demonstrates the benefits of an interactive two-level parent selection mechanism as well as how to seamlessly stitch patches of terrain together. This unique patch-based terrain model enhances control over the evolution process, allowing for terrain to be refined more intuitively to meet the user's expectations.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

General Terms

Algorithms

Keywords

Terrain generation, evolutionary algorithms, video games

1. INTRODUCTION

The expanding video game consumer market is driving game developers to design an abundance of creative in-game content to entice and engage players. The time and financial costs associated with the production of this content are large, typically requiring teams of highly skilled artists and programmers [3]. With this in mind, there is increasing research by both academics and industry leaders [2] to reduce the costs associated with the production of content while still providing the consumers the high quality and varied experiences players have come to expect.

Evolutionary algorithms have recently been proposed as a means of increasing control over the procedural generation of video game content. Togelius et al. [19] provide a review of existing contributions to this field, including the use of evolution to generate game rules [18], AI controllers [14], and projectile physics [8]. Many of these techniques use evolutionary algorithms in combination with player profiling to customize content to a player's preferences and playing style. In this paper we focus on the use of evolutionary algorithms in the generation of 3D terrains that can be used as levels in a game. Terrain plays a pivotal role in the user's experience of the game; setting the scene and mood in story based game-play and providing strategic and competitive game-play in multiplayer games. Research in this field is emerging with seminal techniques being proposed for the generation of levels in racing games [16] and two dimensional platforming games [11].

In this paper, we use evolutionary algorithms during the generation of 3D terrains. Existing research in this area has produced systems that use evolutionary algorithms to vary the height of control points on the height-map [10], alter mathematical functions that define the height of each vertex on the height-map [6], to search for the optimal values of parameters that define general terrain properties [20], and to place terrain features and game resources on a map [17]. Our novel approach divides sample terrains into equal sized square patches and recombines patches from one or more of these samples to produce a new terrain. Evolution can be applied to a patching technique by carrying out crossover and mutation on patch arrangement. One of the primary benefits of using a patching approach is the added control over exploration versus exploitation of the search space. This control is enhanced by the use of an interactive two-level selection scheme involving *Parent Selection* and *Gene Selection*, which allows every patch of each parent to be separately marked for crossover and mutation. Patch-based terrains also present their own set of challenges, such as the seamless sewing of patches to reduce artefacts and to increase the believability of the terrain.

This paper is organized as follows. In Section 2 we provide a background on existing research into the use of evolutionary algorithm on terrain generation. In Section 3.1 we describe our patch-based algorithm for generating terrain,

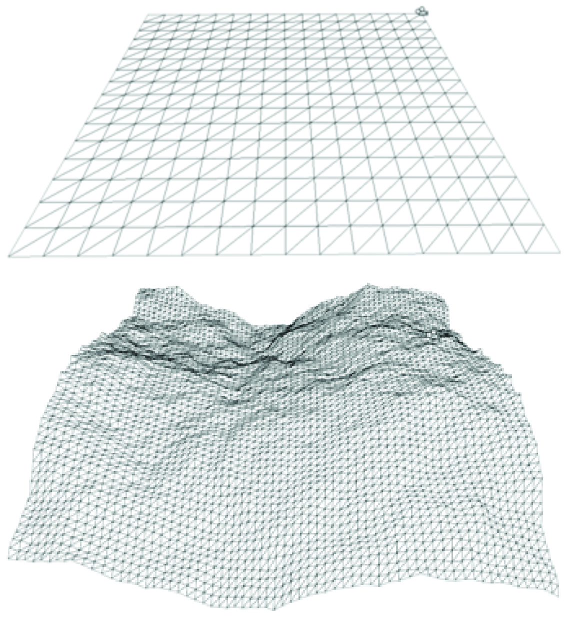


Figure 1: Flat height-map and height-map after fractal terrain generation.

including the important methods and parameters involved. In Section 3.2 we show how evolutionary operators, such as crossover and mutation, can be applied to our terrain generation approach. Section 4 describes the initial experiments we have carried out on our prototype, including justification for specific implementation details, observations on parameter settings, and a demonstration of how the system can be used to generate terrain for games. Finally, Section 5 provides closing remarks and the possible directions of our future work.

2. RELATED WORK

2.1 Procedural Terrain Generation

The most common method for rendering terrain is a height-map [13]. A height-map is usually represented as a three-dimensional array detailing the x,y and z coordinates of vertices. The vertices are evenly spaced across two of the coordinate planes and the third coordinate's value specifies the height of each vertex. The vertices are joined to their neighbours to form triangles and from this a solid surface can be rendered. Examples of height-maps in use are shown in Figure 1.

Procedural terrain generation is the process of programmatically creating terrain for virtual worlds. In the case of height-maps, this involves a system to assign the height value of each vertex. Fractal subdivision was one of the earliest procedural terrain generation techniques [5] and it remains one of the most popular [13] due to its ability to efficiently replicate natural terrain patterns. An example of a fractal terrain is shown in Figure 1. Another popular technique is to use an erosion simulation algorithm [9] to refine an existing terrain, such as one generated fractal subdivision. While erosion simulation is usually computationally expensive, it can also add an increased sense of realism to the terrain. However, the drawback of both fractal subdivision and erosion simulation is that they are highly stochastic and thus there is very limited control over the type of terrain that is generated.

While there are many other approaches to terrain generation [13], we use a patch-based mechanism in this paper. Zhou et al. [21] demonstrate a successful use of a patch-based terrain generation system. Their algorithm allows for a user to provide a basic sketch of feature layout and for terrain to be generated matching it. This is accomplished by first discovering features, such as mountains and valleys, on a sample height-map and extracting them in the form of patches. These patches are then arranged to match the flow of the users sketch. Section 3.1 explains patch-based terrain more clearly, describes how our system utilizes patches and how our approach differs from that of Zhou et al. [21].

2.2 Evolutionary Algorithms on Terrain Generation

Evolutionary algorithms can be employed to add more control to the terrain generation process and can produce terrains that meet the user's preferences of feature arrangement, such as the position and orientation of mountains and valleys. The first notable contribution to this field is by Ong et al. [10]. Their algorithm proposed a two phase process with both phases using evolution to manipulate user provided examples. The first stage was to generate the silhouette of the terrain by mutating a basic user sketch of the terrain's borders into a more complex and rough outline. The second phase involves using a geospatial imagery sample height-map as the initial seed and mutating the height of control points and their surrounding area. The authors propose using fitness functions that compare the resulting candidate to that of the seed height-map, making sure it has a similar design but with unique placement of features. The advantage to the algorithm created by Ong et al. [10] is that, in both stages of generation, the terrain can be seeded, thus introducing the user's preference at the start of evolutionary process. However, the disadvantage is that the user may not able to provide a geospatial imagery system (GIS) sample of the type of terrain they want to generate.

A significant contribution to the field of using evolutionary algorithms on terrain generation is made by Frade et al. with their *genTP* algorithm [6, 7]. The basis of their algorithm is to use noise functions in combination with mathematical operator strings to calculate the height value of each vertex on the height-map. Genetic programming adds, removes, or substitutes operators in the function that defines the height values. Initially, Frade et al. used interactive evolution [6] to demonstrate the algorithms explorative capabilities and the aesthetically appealing terrain that could be produced. They have since investigated automated fitness functions that favour smoother terrains that are well connected [7], which would allow the created terrain to be more appropriate for use in video games. However, so far this has caused the resulting terrains to be excessively flat, indicating the complexities in trying to control this approach programmatically.

Walsh and Grade [20] have used Genetic Algorithms to modify parameter values that describe the terrain, including feature height, feature roughness, water level, and atmospheric properties such as sun angle and cloud coverage. For the first implementation of their algorithm, Walsh and Grade use an interactive evolution mechanism to optimise

Algorithm 1 Roof-Tiling

```
1: set vert_pos to zero
2: while(vert_pos+patch_length)<heightmap_length do
3:   set hori_pos to 0
4:   while(hori_pos+patch_width)< heightmap_width do
5:     select a new patch from database
6:     place new patch at (hori_pos-overlap_size, vert_pos)
7:     stitch new patch with any existing patch to the left
8:     hori_pos=hori_pos+patch_width
9:   end while
10:  shift row to (0, vert_pos-overlap_size)
11:  stitch new row with any existing row above it
12:  vertical_position = vertical_position + patch_length
13: end while
```

these parameters. It should be noted that this algorithm does not create a new terrain, rather it alters the appearance of an existing one. A sample terrain must be supplied to the algorithm and the feature arrangement and orientation of this terrain is not changed, rather the features just increase or decrease in size and rigidity. However, this approach of optimising terrain parameters could be combined with a parameter based procedural terrain generation method, such as those by Doran and Parberry [1], which will give a significant increase in control of the type of terrain that is created.

Recently, Togelius et al. [17] have used multi-objective fitness functions to generate terrain that is appropriate for use in real-time strategy (RTS) games. Unlike the approaches mentioned earlier that only address terrain generation, this algorithm focuses more on creating playable game levels. This includes the placement of player bases, collectable resources, and terrain obstacles on the map. Thus, terrain generation becomes a process of object placement rather than direct height-map manipulation. Togelius et al. also put more emphasis on automating the generation process through the use of multi-objective fitness functions that strive to provide fair and fun game-play for all players. The disadvantage to this change of focus, however, is that the terrain that is generated lacks detail and variety, providing fair game-play at the cost of aesthetically pleasing terrain.

3. APPROACH

3.1 Patch-based Terrain Generation

Our algorithm uses a patch-based system to generate the terrain for all evolutionary candidates. In this system, smaller height-map samples, all of equal size, are stitched together to form a larger height-map. One benefit of patching is that it does not require the height of each vertex to be recalculated every time a new terrain is generated, rather just copying height values from the smaller patches to the larger terrain. Also, patching provides more control by allowing good local features to be kept and for undesirable features to be swapped out, whether a feature is contained in one patch or the result of combining multiple patches.

An example of a patching system for 3D terrain generation is shown by Zhou et al. [21]. Our approach differs in a few fundamental ways though. Firstly, Zhou et al. extract patches that have noticeable features on them, requiring a feature detection algorithm. Our system extracts patches from a sample height-map in a nested looping manner. The algorithm starts in the top left corner of the sample height-map, extracts a patch, moves to a full patch size to the right, and then extracts another patch. This is repeated for a full row of patches, then shifts down one patch size and repeats the process on a second row. As this is a fast process, we can do this for multiple sample terrains at the beginning of program execution. In our experiments we used eight sample terrains of 512x512 vertices, however the number of patches that are extracted is dependent on the size of each patch. It should be mentioned that for the experiments in this paper, the sample height-maps were generated through procedural fractal methods along with handcrafted terrains that contain elevations that had little thought put into them. Both of these types of sample terrain were used to give the system a variety of different features and average height values in the patches. Future experimentation will examine the affect that different patch sets will have on the resulting terrain generated by our system.

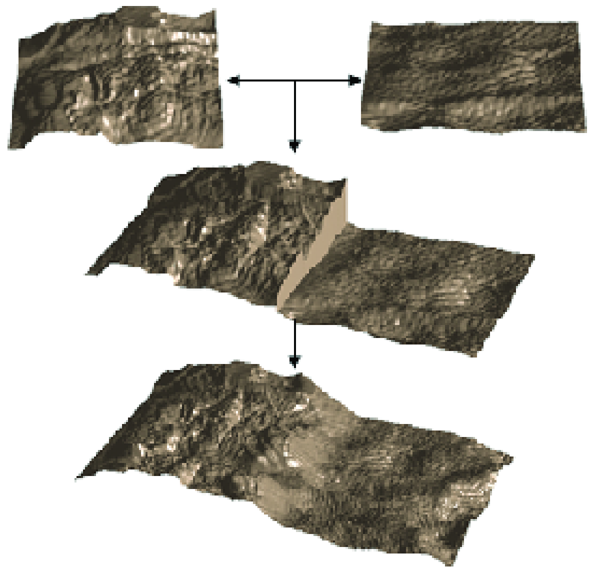


Figure 2: Two patches first joined with no overlapping and then with overlapping and cubic spline interpolation.

Zhou et al. also take care in how patches are placed, requiring searching for the best fit patch for each position and manipulating it to better match the user's feature sketch and neighbouring patches. Our algorithm places patches down in a grid like fashion more attune to texture synthesis algorithms [4]. Our approach to patch arrangement and seamless patch stitching is covered in detail in Section 3.1.1. Finally, while Zhou et al. use a feature sketch to control patch layout, control in our system is provided through the use of evolutionary algorithms, allowing for more interactive adjustments to the terrain.

3.1.1 Roof-Tiling

An important aspect of a patching technique is how the patches will be stitched together. If an inappropriate method is used then the borders between patches will be evident. Zhou et al. [21] use an algorithm they call Poison Seam Removal, which is a variation on the popular Poison Image

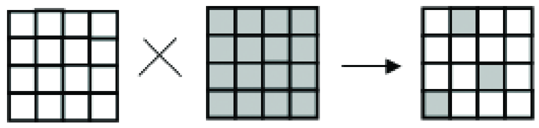


Figure 3: Example crossover operation. White parent is the base parent with a crossover rate of 0.2.

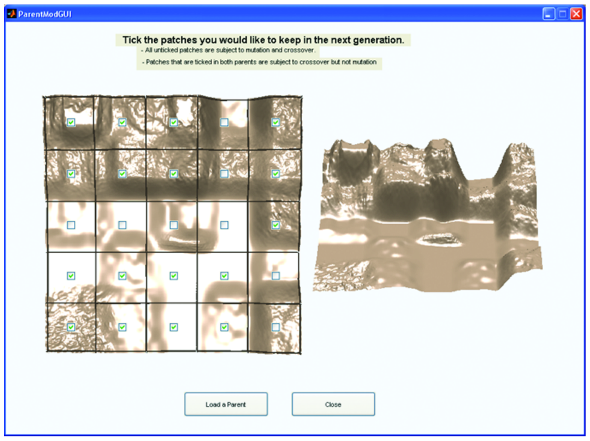
Editing [12] technique used in blending images or transposing features of one image onto another. The authors of the Poison Image Editing algorithm, Pérez et al. [12], state that their algorithm works best if the intensities of the pixel are within similar ranges for both images. In the case of heightmaps, this means that the area around the seam must have similar height values. This does not perform well in our system because two patches with very different height ranges, such as a mountain ridge and a valley, can be placed next to each other due to the stochastic selection of patches during mutation operations.

In order to stitch patches of varying heights together with minimal seams, we employ a *Roof-Tiling* overlapping algorithm, so called due to its likeness to laying tiles on the roof of a house. This process is outlined in Algorithm 1. It involves stitching rows of patches together and then stitching each entire row onto the one above it. All patches overlap all of their neighbours in order to aid in seam removal. The size of the overlap region and the overlapping technique used will determine the quality of the final terrain. We used an interpolation technique to calculate the height values of the vertices in the overlap region. That is, if we have a left patch and a right patch that are being stitched together, then the height values closer to the left side in the overlap region will be largely influenced by height values of the left patch and not much by the patch on the right side. This applies vice versa and height values in the centre of the overlap region should be influenced relatively equally by both patches. In our system we use a cubic spline interpolation function to smooth out the transition in the overlap region, which is discussed further in Section 4.3. Figure 2 shows two patches of terrain first separately, then joined with no overlapping, and then stitched together using our method of overlapping and cubic spline interpolation.

3.2 Evolving Patch Placement

In our genetic representation, each patch is a gene and the chromosome is structured as a fixed size grid. That is to say, if each candidate terrain is constructed of sixteen patches, then the candidate's chromosome is a four-by-four grid of genes. This means that evolutionary operators, such as crossover and mutation, only need to deal with substituting patches, the number of patches (or genes) cannot be increased or decreased. The initial population can either be seeded by the user provided sample terrains, from which patches are extracted, or it can be randomly generated. Unlike many of the previous contributions in this field, our system does not alter the topology of individual patches. The evolutionary algorithm changes the spatial placement of patches rather than manipulating height values of individual vertices.

Our system uses a uniform crossover technique [15], whereby patches from each of the two parents have respective probabilities of appearing in a child. One of the parents is randomly chosen to be the base parent, meaning that its entire chromosome is copied to the offspring. Each patch slot is then randomly given a crossover probability. If that probability is greater than a crossover rate parameter, which is provided at the start of the run, then the patch from the base parent is kept, otherwise it is switched for the patch in the same position from the other parent. Figure 3 shows an example of crossover with a white parent and a grey parent.


ParentModGUI

Tick the patches you would like to keep in the next generation.
- All unticked patches are subject to mutation and crossover.
- Patches that are ticked in both parents are subject to crossover but not mutation

Figure 4: The *Gene Selection* interface.

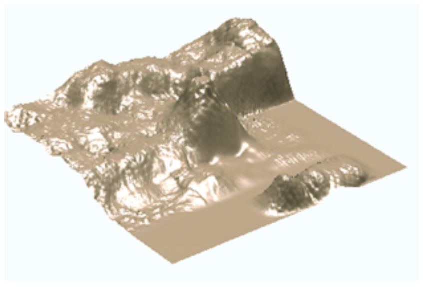


Figure 5: The ridge in the bottom right is undesirable. Removing this patch through *Parent Selection* alone can take many generations of waiting for a suitable mutation.

Mutation is carried out in a similar way to crossover. Each gene position is randomly given a mutation probability value; if this value is greater than the mutation rate parameter then the current patch is kept, otherwise it is substituted with a randomly chosen patch. Mutation occurs on a child after the crossover operation has completed. If only one parent is selected, mutation is carried out on that parent to generate all of the offspring.

For parent selection we use a hierarchical interactive evolution mechanism. The user is presented with fully rendered images of all of the candidate terrains and they are allowed to choose up to two parents, we refer to this simply as the *Parent Selection* stage. Subsequent to this, if the user wishes, they can also undertake *Gene Selection*, which allows them to specify which genes, or patches in this case, will be subject to crossover and mutation. The reason we chose to implement this type of hierarchical selection mechanism is discussed in Section 4.2.

Using a patch based terrain generation algorithm has multiple benefits over the other evolutionary terrain generation

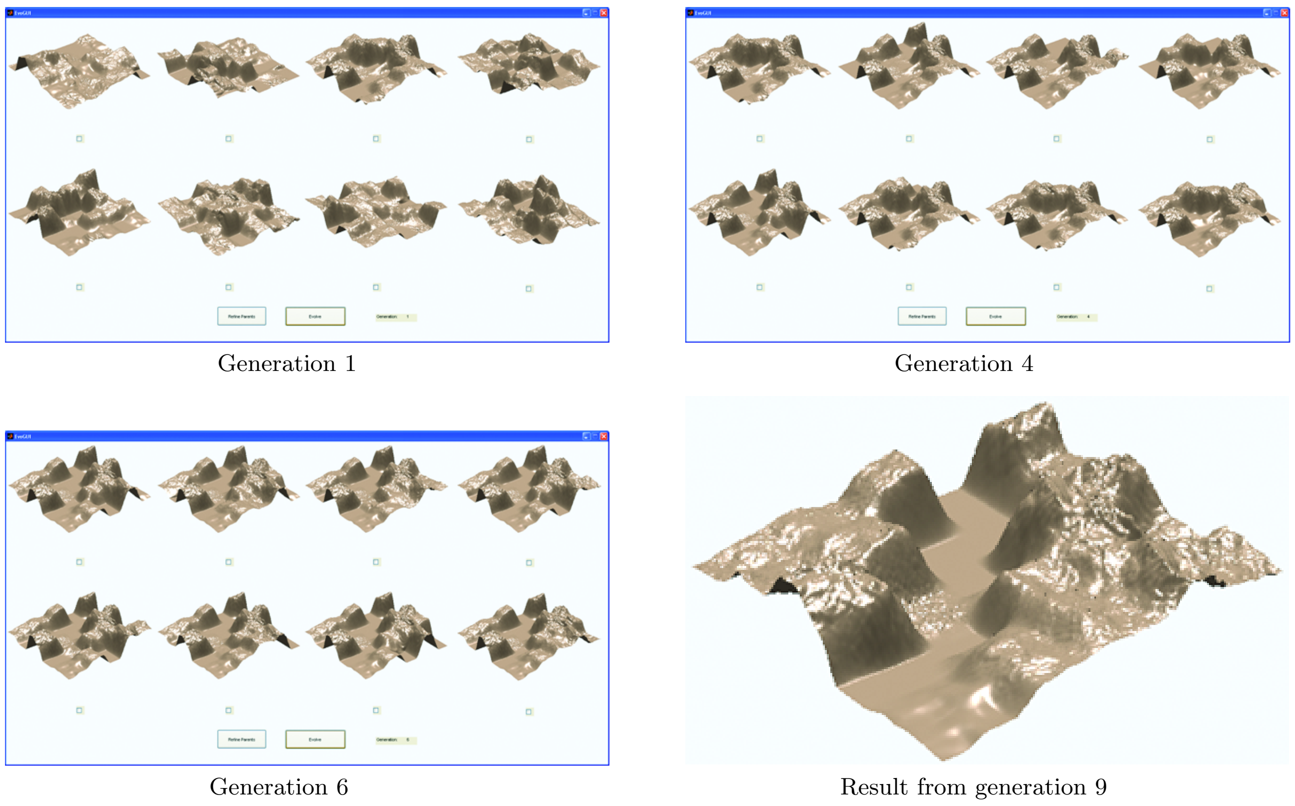


Figure 6: One run over ten generations using the hierarchical selection system.

methods [6, 10, 17, 20]. The use of patches along with the two-level parent selection mechanism offers a far greater level of interaction between the user and the evolutionary process, giving the user more control of feature arrangement. As for the use of the resultant terrain in video games, unlike Ong et al. [10] and Walsh and Grade [20] our system does not require one single existing height-map that exhibits similar feature layout to the user's desired terrain. Instead we allow the users to supply multiple sample terrains and to also change their desired goal throughout the evolutionary process. The terrains produced in early works of Frade et al. [6] contain many artefacts and steep transitions between features which prevents the terrain from being useful in video games. Their later work, involving accessibility fitness functions [7], ensures connectivity in the terrain but at the cost of producing terrain that is overly flat. Results in Section 4 show our algorithm can produce terrain that both contains rich features as well as connectivity, making it ideal for use in video games. We agree with Togelius et al. [17] that object placement is an important part of video game level creation but it should not come at a cost of terrain detail. Finally, the tool we have developed here is designed to be used as an aid for both experienced and inexperienced content developers, helping them explore the search space of possible terrains.

4. PRELIMINARY RESULTS

4.1 Parameter Settings

The prototype system that we created works through interactive evolution. As user fatigue is an issue, we seek parameter settings that allow the user to efficiently explore the search space and develop an idea of the type of terrain they want to create. Once the user has a rough idea of what they want, we need the system to converge quickly to produce a terrain that is acceptable to the user.

For all of our experiments we used a fixed height-map size of 512x512. This is the number of vertices per candidate terrain and, if the height-map was represented as a two dimensional image, this would be the number of pixels. Of more importance is the size of each patch. The patch size is determined by how many patches make up the terrain, or how many genes there are in the genetic representation. We found ideal values for interactive evolution to be either sixteen patches (4x4) or twenty five patches (5x5). Any fewer and there will not be enough variety in the candidates. Any more and candidate terrains become jagged due to an increased chance of two adjacent patches having a large difference in their average height values. Also, with more patches evolution will be harder to control because mutation will substitute an increased number of patches per offspring.

In all of the following experiments we used a population size of 8, allowed a maximum of 2 parents to be selected per generation, used a crossover rate of 0.5 to ensure both parents influenced the offspring equally, and used a mutation rate of 0.1. We found that a mutation rate above 0.2 lead to difficulty in controlling evolution because too many patches would change per offspring. Section 4.2 addresses the negative affects of mutation in our system while Section 4.3 explains why cubic spline interpolation was used to smooth out patch transitions.

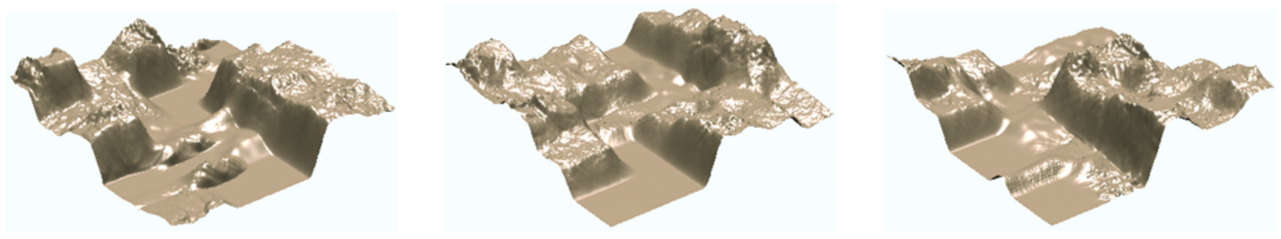


Figure 7: Results of seperate runs with a similar user goal as Figure 6.

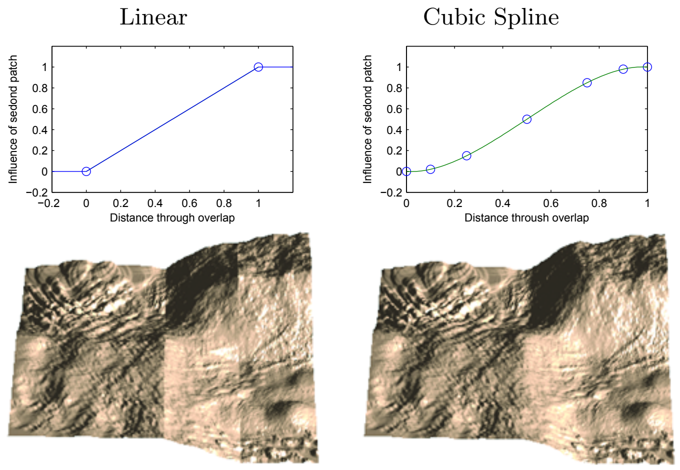


Figure 8: Linear interpolation and cubic spline interpolation at 50% overlap.

4.2 Two-Level Interactive Evolution

Our interactive evolutionary algorithm uses a hierarchical approach with two layers of selection. The first layer is a typical *Parent Selection* mechanism similar to those used in evolutionary art. Here, the user chooses which candidates are visually appealing to them and those chosen become the parents of the next generation. The second, more unique, layer is the *Gene Selection* process. During this stage the user is able to select which patches, or genes, they would like to exempt from the crossover and mutation operations. Figure 4 shows the interface used to allow the user to visually select which patches of a single parent are desirable and which are not. If no patches are selected then it is assumed that all patches are subject to crossover and mutation.

The *Gene Selection* mechanism was introduced after we perceived a flaw in our early experiments that only utilized the *Parent Selection* system. Figure 5 shows a terrain with an undesirable patch. In this scenario the user wants a mountain ridge on one side of the terrain and flat plains on the other side. Thus, the ridge on the bottom right in this figure is preventing the terrain from matching the user's desires. With a mutation rate of only 0.1 or 0.2 there is only a small chance that this one patch would change and when it does another patch may change too, leading to a new undesirable patch. This meant that it often took dozens of generations to substitute a single undesirable patch, increasing both user fatigue and frustration.

The *Gene Selection* mechanism aids in preventing the above situation from occurring. In this scenario, the user

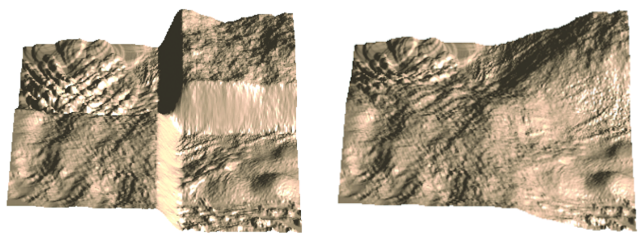


Figure 9: Cubic spline interpolation used with overlap sizes of 20% and 80%.

would indicate only one patch to be subject to mutation. An inverse mutation rate is used to ensure that this one patch is highly likely to change in all offspring. If all patches are subject to mutation then the base rate specified in the mutation rate parameter is used. As the number of patches affected by mutation decreases, the mutation rate increases towards 1.0.

The hierarchical selection system allows for users to utilise exploration and exploitation when they need them. Figure 6 shows one run of our prototype where the user desired a terrain with a canyon running between two ridges, a terrain that may be difficult to create with other terrain generation techniques. Screen shots are shown for generation one, four, and six, as well as a resulting terrain from the ninth generation. Up until the fifth generation, only *Parent Selection* was used, allowing the user to quickly get candidate terrains that exhibited some minor resemblance to the desired terrain. This can be seen in the fourth generation in Figure 6 as all of the candidate terrains have similar features. After this, *Gene Selection* was used to refine the terrain, swapping out only a handful of undesirable patches in each generation. Figure 7 shows how our algorithm can generate multiple terrains with similar feature layouts but with different details. Each of the terrains shown in Figure 7 was generated from a separate run of our prototype, all with the same user goal that was used in Figure 6.

4.3 Smooth Patch Stitching

In all of the examples shown in this paper, cubic spline interpolation was used to smooth out the overlap region between patches. That is, a cubic spline curve was used to determine the influence of both patches on each height value in the overlap region. In our initial implementation we used a simplistic linear interpolation algorithm. Figure 8 shows the same terrain, constructed by four patches (2x2) and with an overlap size of 50%. The beginning and end of the overlap region between patches can be seen in the linearly in-

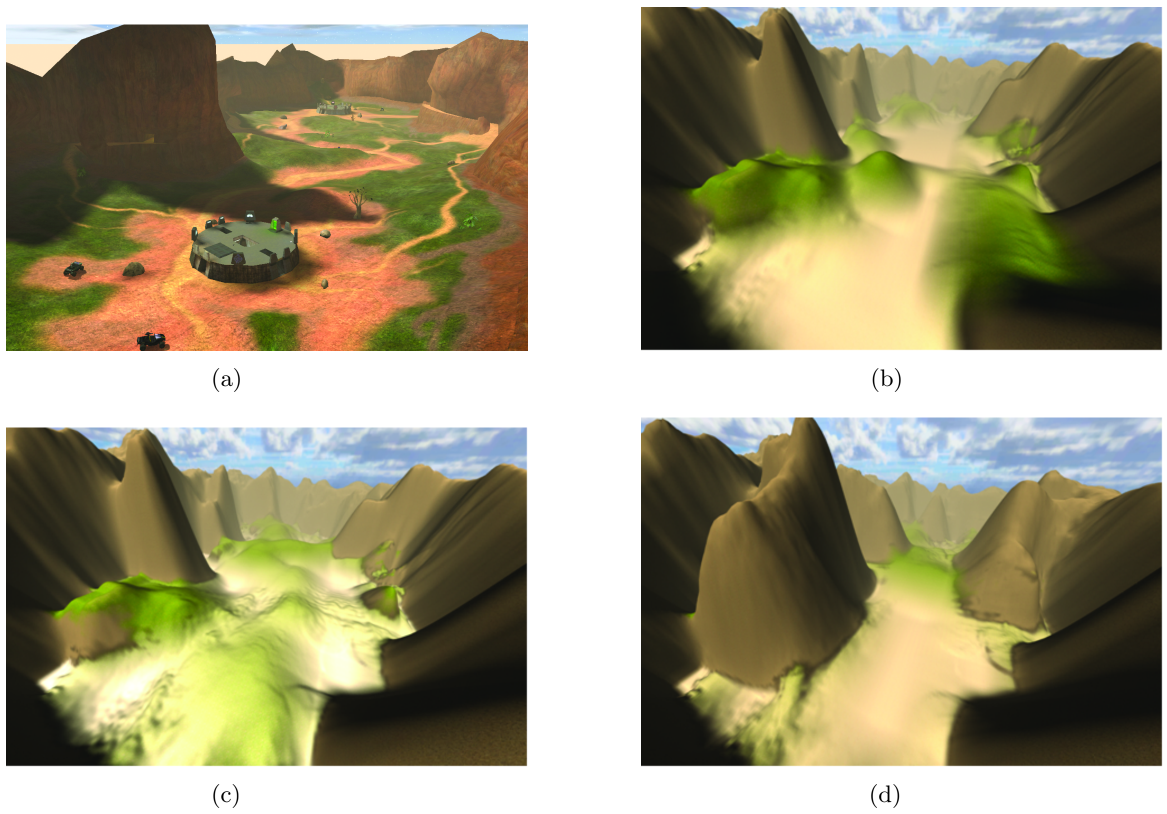


Figure 10: (a) is a map from the game *Halo*, by Bungie, 2001. (b), (c) and (d) were all generated by our algorithm.

terpolated terrain. The cubic spline interpolation method, however, gradually enters and exits the overlap region.

The overlap size value mentioned earlier indicates how much of each patch should be used when overlapping with another patch. When patches are being extracted from sample terrains, this extra amount is also extracted to the right and bottom of the patch, thus enlarging the overall patch size. This helps features to be maintained throughout the overlapping procedure and maintains the number of patches that are used to construct a terrain. However, because patches overlap with their neighbours in all directions in the Roof-Tiling algorithm, some feature detail is still lost when overlapping with patches to the left and above.

For linear interpolation to produce terrain that does not show the boundaries of the overlap region, an overlap size of 90% is needed, restricting variety in steepness in patch transitions. In contrast, cubic spline interpolation can be used at many overlap sizes to produce different terrain detail. Figure 9 shows the same 2x2 terrain using cubic spline interpolation at 20% and 80% overlap respectively. With an overlap size of 20% there are sharp transitions between patches while with an overlap size of 80% the raised area in the top right of the terrain is almost completely smoothed over. Both of these values may be desirable for different terrain effects, such as using small overlap sizes to represent cliff faces and large overlap sizes to create rolling hills.

4.4 Video Game Map Generation

Figure 10 shows how our system can be used to generate approximations and variations of terrain used in a video game. Image *(a)* in this figure is the original map from the game *Halo*[1], developed in 2001 by Bungie. The user did not desire to create a replication of this terrain but instead used it as inspiration for new terrain. Images *(b)*, *(c)* and *(d)* were all created in our prototype system and rendered in the *Unity 3* game engine[2]. It should be noted that our renderings lack detailed texturing and also do not have any vegetation, structures, or other virtual objects. Image *(b)* is the closest approximation to the original game map. Image *(c)* shows how the playable area can be changed slightly, which would cause players to adapt their strategies. Image *(d)* shows more drastic changes to the borders of the playable area, which would change game-play and require players to re-explore the map.

To achieve the terrains in Figure 10, we use multiple runs with the output from one run (the ideal terrain of that run) being used as a sample terrain and seed for the next run. This allowed us to change parameters such as patch size and overlap size between runs to achieve different terrain effects. The first and second runs used large patch sizes to generate a rough outline of the cliffs around the playable area. The third run used smaller patch sizes with steep transitions between patches to create the hard edges of the cliff face and protruding peaks. Finally, in all successive runs, the playable area was manipulated with small patch sizes to allow for more control but with a large overlap size to ensure smooth inclines and playable surfaces.

[1]http://halo.xbox.com/en-us
[2]http://unity3d.com/

5. CONCLUSION AND FUTURE WORK

In this paper we have introduced a novel algorithm for applying evolutionary algorithms to the generation of virtual terrain. A unique patch-based terrain generation system was used to allow for more control of the evolutionary process over existing evolutionary terrain algorithms [6, 10, 20]. The use of both *Parent Selection* and *Gene Selection* during interactive evolution gives the user control over how many patches are affected by crossover and mutation. This allows for quick browsing early in a run and greater control during the refinement stage. We also provided algorithms for stitching height-map patches to produce seamless, artefact free, 3D terrains. We believe that the terrain generated by our system is appropriate for use in video games due to the rich feature arrangement and the high connectivity that is possible.

Future work will involve investigating the automation of the evolutionary process. *Gene Selection* may play a pivotal role when interactive evolution is replaced with automated fitness functions, allowing each patch to have its own fitness relative to a single candidate. This will lead to more natural feature layouts and reduce the size of the overlap region. This may also be achieved by adding constraints on patch placement, requiring that a logical and natural terrain be generated before consideration for breeding. We hope to utilise our algorithm to customize content for players based on their patterns of play. Thus, once our algorithm is refined, we will pursue implementing it into a complete game, which will allow us to experiment with the algorithms effectiveness in generating video game terrain efficiently and reliably and to examine players' enjoyment of the generated terrain.

6. REFERENCES

[1] J. Doran and I. Parberry. Controlled procedural terrain generation using software agents. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):111–119, 2010.

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

[3] R. Edwards. The economics of game publishing. IGN Entertainment Inc., 2006, Online: http://au.games.ign.com/articles/708/708972p1.html , Accessed: 13 December 2010.

[4] A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, pages 341–346. Citeseer, 2001.

[5] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.

[6] 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.

[7] M. Frade, F. F. de Vega, and C. Cotta. Evolution of artificial terrains for video games based on accessibility. *Proceedings of the European Conference on Applications of Evolutionary Computation*, 6024:90–99, 2010.

[8] E. Hastings, R. Guha, and K. Stanley. Evolving content in the galactic arms race video game. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 241–248. IEEE, 2009.

[9] 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*, pages 41–50. ACM, 1989.

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

[11] C. Pedersen, J. Togelius, and G. Yannakakis. Modeling player experience in super mario bros. In *IEEE Symposium on Computational Intelligence and Games(CIG)*, pages 132–139. IEEE, 2009.

[12] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

[13] 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.

[14] K. Stanley, B. Bryant, I. Karpov, and R. Miikkulainen. Real-time evolution of neural networks in the NERO video game. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1671, 2006.

[15] G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9. Morgan Kaufmann Publishers Inc., 1989.

[16] 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)*, pages 252–259. IEEE, 2007.

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

[18] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 111–118. IEEE, 2008.

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

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

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