Terrain Generation Using Genetic Algorithms

Teong Joo Ong
Center for the Study of Digital Libraries
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
mong@csdl.tamu.edu

Ryan Saunders
John Keyser
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
{rls8901,keyser}@cs.tamu.edu

John J. Leggett
Center for the Study of Digital Libraries
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
leggett@csdl.tamu.edu

ABSTRACT
We propose a method for applying genetic algorithms to create 3D terrain data sets. Existing procedural algorithms for generation of terrain have several shortcomings. The most popular approach, fractal-based terrain generation, is efficient, but is difficult for a user to control. Other methods tend to require too much user input. In this paper, we provide an alternative method of terrain generation that uses a two-pass genetic algorithm approach to produce a variety of terrain types using only intuitive user inputs. We allow a user to specify a rough sketch of terrain region boundaries, and we refine these boundaries using a genetic algorithm. We then couple this with a database of given terrain data to generate an artificial terrain, which we optimize using a second genetic algorithm.

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

General Terms
Algorithms

Keywords
Terrain generation, genetic algorithms, geographic information systems (GIS), height field, image processing

1. INTRODUCTION
Artificial terrain generation involves the creation of a set of elevation values over a two dimensional grid, such that the resulting model appears to be the surface of a real region of land. While games, movies, and similar forms of entertainment are the most popular applications for terrain generation algorithms, these algorithms also find application in a variety of other contexts, including simulation and training environments.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'05, June 25-29, 2005, Washington, DC, USA.
Copyright 2005 ACM 1-59593-010-8/05/0006…$5.00.

Current terrain generation algorithms span a wide range. These are described in more detail in section 2.2. All of these current techniques have key limitations. Many of the simpler methods are suitable for creating only a narrow range of terrain types. Some require the user to possess arcane parameter tuning skills in order to get acceptable results. Most of them provide no localized control over the landscape, and those that do require the user to invest a significant amount of skilled labor in order to get results.

In contrast, for a terrain generation tool to be maximally useful to artists, game developers, architects, and simulation designers (i.e., the user base of such a tool), it ought to enable its users to design the terrain visually using intuitive controls, producing realistic results across a variety of terrain types.

The approach we present here aims to address many of the shortcomings of these existing methods. We give the user a manageable level of control in specifying the types of terrains, and the approximate regions that each terrain type should occupy. We then make use of genetic algorithms in two stages to create a realistic terrain model. The resulting method is capable of generating realistic terrains with only intuitive inputs and limited knowledge required of the user, and could be implemented as a standalone, CAD-style tool, or as a plugin for any of the major 3D modeling packages commercially available.

2. BACKGROUND
2.1 Terrain Representation
While terrain data can be represented in a number of ways, by far the most common structure for terrain representation is the *height field*. In mathematical terms, a height field is a scalar function of two variables, such that every coordinate pair (*x, y*) corresponds to an elevation value *h*:

$$h = f(x, y)$$

In practice, a height field is normally implemented as a two-dimensional, rectangular grid of height values, and is equivalent to a grayscale image. Height fields have the limitation that they cannot represent structures in which multiple surfaces have the same (*x,y*) coordinates (such as caves and overhangs), but are sufficient for most uses and can be highly optimized for rendering and object collision detection [3] [14].

2.2 Existing Terrain Generation Techniques

Existing terrain generation techniques can be grouped into several categories.

The most flexible class of techniques is the family of *sculpting techniques*, in which a human artist "paints" or "sculpts" the terrain manually, using an image editing program (e.g., Adobe Photoshop), a 3D modeling program (e.g., Maya or 3D Studio), or a specialized "terrain editor" program, (e.g., *Terragen* [13] or the editors that ship with the recent game titles *Unreal Tournament 2004* [15] and *SimCity 4* [12]). The set of terrain modification operations available to the artist will differ depending on which type of editor is being used, but the general principle is the same. These techniques have the advantage of offering the user almost unlimited control over the details of the terrain. But this advantage is also a disadvantage – by leaving most or all of the details up to the user, these techniques place high requirements on the user in terms of time and effort, and the realism of the resulting terrain is completely dependent on the skill of the user.

At the opposite end of the spectrum are *Geographic Information Systems (GIS)-based techniques*, in which elevation data is derived from real-world measurements (e.g., satellite imagery, land surveys). GIS data can be acquired from a number of sources (e.g., [6]) and in several formats, such as the U.S. Geological Survey's DEM (Digital Elevation Model) format [16]. GIS approaches have the advantage of offering highly realistic terrains with very little human effort, but at the expense of user control. If the user has specific goals for the layout of the terrain and the kinds of features present, GIS approaches may actually be very time-consuming, as the user might have to search extensively to find real-world data that meets specific criteria.

A third category of terrain generation methods is the class of procedural techniques, in which the terrain is generated programmatically. These methods can be further separated into the *physically-based techniques* and the *fractal techniques*.

Physically-based techniques simulate the effects of physical processes such as erosion by wind [17] or water [4] [7], or plate tectonics. These approaches can generate highly realistic terrains, but require a thorough understanding of the physical laws to implement, and possibly even to use effectively. They can also be extremely costly in terms of processing time, and may not generalize well to accommodate new terrain types.

Fractal techniques exploit the self-similarity property exhibited (to a limited extent) by some types of terrain. An object is said to be self-similar when magnified subsets of the object are similar (or identical) to the whole object and to each other [10]. For example, the jagged edge of a broken rock might appear similar to the ridgeline on a distant horizon. One can thus use certain forms of fractals to generate height fields that somewhat resemble real terrains[JK6] [11]. Variability in the resultant terrain can be introduced by incorporating randomness into the fractal algorithm; even so, the self-similarity characteristic of fractals makes fractal-generated terrain often easily recognized as such. To get believable terrains, the various (non-intuitive) fractal parameters used by the engine may have to be tweaked extensively. This class of techniques is the current favorite of the computer game industry, largely due to their speed and simplicity of implementation [2] [5]. There are a number of specialized terrain generation tools available that are based predominantly on fractal techniques, such as *Terragen* [13] (which is really a hybrid fractal/sculpting tool) and *MojoWorld*, a sophisticated program for creating entire fractal worlds [8].

Based on our observations of the strengths and weaknesses of each type of technique, our goal is to explore an alternative approach in terrain generation using genetic algorithms (GA[JK7]).

2.3 Characteristics of a Good Terrain Generation Algorithm

Since terrain comes in many different geometric shapes, forms and characteristics, a good, general algorithm for terrain generation should be: 1) adaptive – performing well across different terrain types and permitting new terrain types to be added; 2) innovative – exhibiting emergence characteristics such that new and original constructs not necessarily specified in the input are created; 3) scalable – capable of handling datasets and results at varying terrain resolutions; and 4) intuitive – allowing the human user to exercise control over the terrain generation process using easily understood and predictable parameters. While prior methods may meet some of these characteristics well, they tend to perform poorly in others; we believe that our approach does well at meeting all of these characteristics. Evolutionary computation is well suited for the terrain generation problem because it requires searching through a huge number of possibilities (all possible terrains) to find good approximate solutions (meeting both realism and user-specified constraints)[9].

3. METHODS

Our approach breaks down the terrain generation process into two stages: the terrain silhouette generation phase, and the terrain height field generation phase. The input to the first phase is a rough, 2D map laying out the geography of the desired terrain that can be randomly generated or specified by the user. This map is processed by the first phase to remove any unnaturally straight edges (Figure 1), and then fed to the second phase, along with a database of pre-selected height field samples representative of the different terrain types. The second phase searches for an optimal arrangement of elevation data from the database that approximates the map generated in the first phase.

3.1 Terrain Silhouette Generation

The input to this phase is a 2D "map" of polygonal terrain "regions" specifying the approximate size, shape, and position of different terrain types (for example, a large, elliptical region of mountains surrounded on all sides by rolling hills). The user could create these regions manually, using a simple, CAD-style interface, or they could be generated randomly. In either case, for the linear boundaries of these rough polygonal shapes not to be noticeable as artifacts in the generated terrain, they must be broken down into more natural-looking, uneven boundaries. We perform this edge modification by subdividing each edge into a sequence of points and applying a GA to produce an acceptable boundary shape.

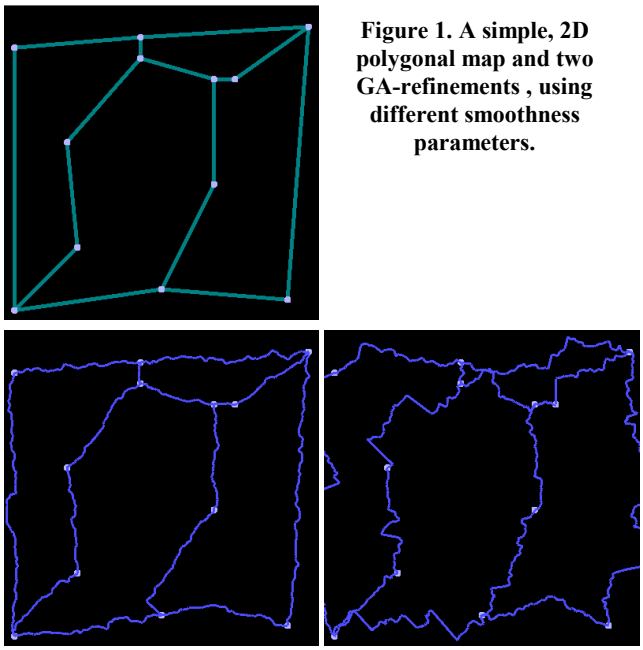


Figure 1. A simple, 2D polygonal map and two GA-refinements , using different smoothness parameters.

Individuals in the initial population are created as follows:

1. For a particular input boundary segment, let P_0 be the starting point and P_n be the end point. Define P_0 to be at the origin, and P_n to lie on the positive x-axis.
2. We will generate a series of n intermediate points, P_i, which will be connected to create a new set of line segments connecting P_0 to P_n, evenly spaced in terms of their x-coordinates. Create the first line segment connecting P_0 to P_1 by choosing an arbitrary angle, θ from the x-axis, lying within a predetermined range of angles, and place P_1 along that angle at a fixed x-value.

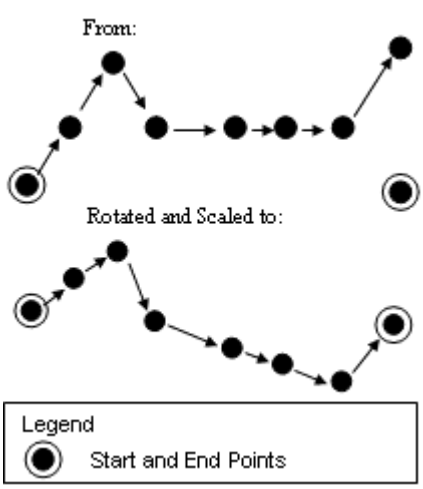


Figure 2. Rotate and scale the line segments to match the specified start and end points

3. Now create a line segment connecting P_i to P_{i+1} (whose y-coordinate is still unknown) by generating an arbitrary angle θ such that the resultant angle formed from P_{i-1}, P_i, and P_{i+1} falls below the user-defined threshold[JK15]. This user-defined threshold gives a measure of the boundary's local *smoothness* (small values indicate slow variation, and thus smoother boundaries).
4. Repeat step 3 n - 1 times. Note that P_n is probably not at the same position the original end point was in.
5. Since the resultant boundary's endpoints will not, in general, line up with those of the original shape, apply a rotation and scaling to make the generated boundary fit into the desired location (Figure 2).

3.1.1 Encoding/Decoding and Fitness Evaluation

Floating point encodings are used in the genes to record the angle θ (in radians) between P_i and P_{i+1} such that the encoded values may range from -1 to 1 where a value of 1 indicates that the line segment heads in the same direction as P_{i-1}. Figure 3 provides two examples of how the angle between two line segments is measured. A value of 1 in a gene represents an angle of π radians (i.e., the two segments have the same direction), with positive values representing "upward turning" angles and negative values representing "downward-turning" angles. The smaller-magnitude angle is always chosen, such that no angles larger than π radians are possible. The angle at P_0 is encoded slightly differently since there is no P_{-1}; instead, the angle measure is calculated with respect to the imaginary line connecting the start and end points. The decoding process is essentially the reverse of the encoding process (and therefore will not be described). The merit of such an encoding scheme is that it allows us to evaluate the fitness of an individual without explicitly decoding the floating point numbers to Cartesian coordinates. S is the smoothness parameter, which ranges from 0.0 to 1.0 such that 1.0 indicates perfect smoothness (i.e., a perfectly straight line). The floating point values f in the genes have a direct relationship with the value of S since a higher value of f indicates a smoother transition between the line segments (and thus, a higher S value).

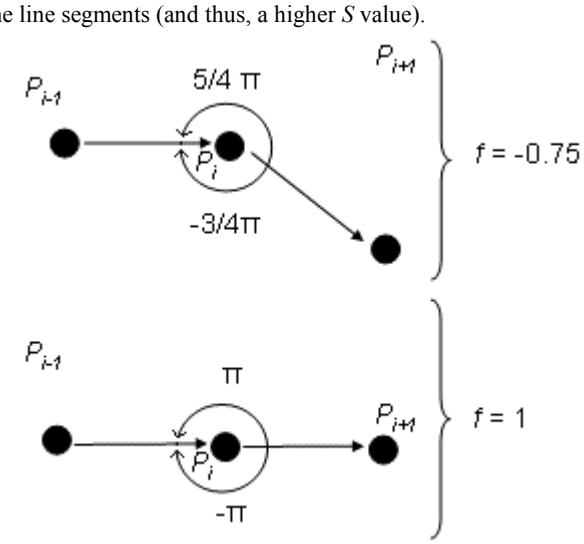
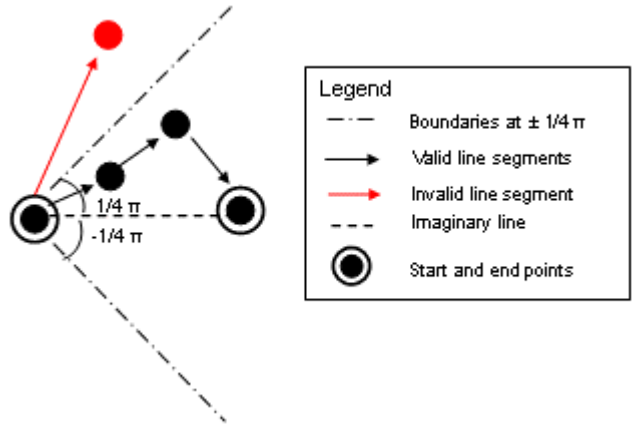


Figure 3. Angle between two line segments will be encoded into a gene as a floating point number

Unconstrained mutations or generation of line segments in the silhouette can yield results that are difficult or impossible to

transform to fit the specified start and end points (consider the case where the generated boundary's start and end points are identical). In light of this observation, we impose a global constraint on the initialization stage and mutation operators such that the segments in the silhouette line can only fall within the region spanned by two lines projected at ± 1/4 π (with respect to the baseline) (Figure 4). The value ± 1/4 π was chosen somewhat arbitrarily, but works well in practice – any value significantly smaller than ± 1/2 π can be used, since this ensures that progress is always being made towards the end point, and that the generated boundary can be easily transformed to fit.



[JK21]

Figure 4. Line segments can only fall in between the region spanned by the boundaries

Fitness of the individuals is determined by evaluating the smoothness of the resultant silhouette. A smoothness measure is computed based on the ideal range of *f* for a particular smoothness parameter *S*, calculated as:

$$1-(\frac{1-S}{4}+\delta) \le |f| \le 1 \text{ since } \Pi \text{ is encoded as } 1$$

where δ is a small, predefined floating point value. This range is reduced as *S* increases so that higher fitness values are assigned only to line segments with smoother transitions and vice versa. [JK22]As the deviation of a gene's *f* from this ideal range increases, the gene is penalized with a lower fitness value. The fitness value for a gene *i* is computed as follows:

$$\text{Fitness value } v_i = Sin\,(\frac{\Pi}{2} f_i\,(1.1 - S))$$

Where v_i is the fitness value for a particular gene *i* and f_i is the encoded angle for gene *i*. The function exhibits a horizontal asymptote at $f_i = 1$ for low smoothness level due to the term 1.1 – *S* (The value 1.1 is chosen instead of 1 to handle cases when *S* = 0) so that the assignment of fitness values favors sharper transitions between the line segments. The function slowly turns into a linear function for higher smoothness values in order to penalize sharper transitions between the line segments. Finally, the fitness value from each gene is combined to obtain a value that contributes to the individual's fitness value.

Similarly, the smoothness at coarser granularities of the resultant silhouette is computed by taking a subset of the *f*s from the line segments at certain fixed intervals, such as f_1, f_4, f_8 and f_{12}, and computing their fitness value based on the evaluation scheme discussed above. Lastly, the fitness values at each different granularity level are combined based on a weighted average function. Thus, the final fitness value *F* (overall smoothness) is determined by the following equation:

$$F = \sum_{i=1}^{m} \frac{N(A_i)}{\sum_{j=1..m} N(A_j)} * F_i$$

Where *m* is the number of granularity levels we are computing for the silhouette line fitness and the *N* function provides the number of elements in the A_i set of *f* values.

To simplify calculations, fitness values of individuals in the implementation of our GA are computed at only two granularity levels. As an example, we assume that our line segments at two different granularity levels are $A_1 = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ and $A_2 = \{f_1, f_3, f_5, f_7\}$ respectively. The total fitness values for the corresponding granularity level are F_1 and F_2 respectively.

3.1.2 Genetic Operators

The GA used in the silhouette generation process utilizes the standard crossover and mutation operators [1]. The crossover operator (Figure 5) randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. The mutation operator randomly alters the floating point value encoded in the genes (Figure 6). The effects of these operators on the candidate solutions can be very drastic because the direction (and indirectly, the length) of a particular line segment is dependent on the previous radian values *f* encoded in the genes before it.

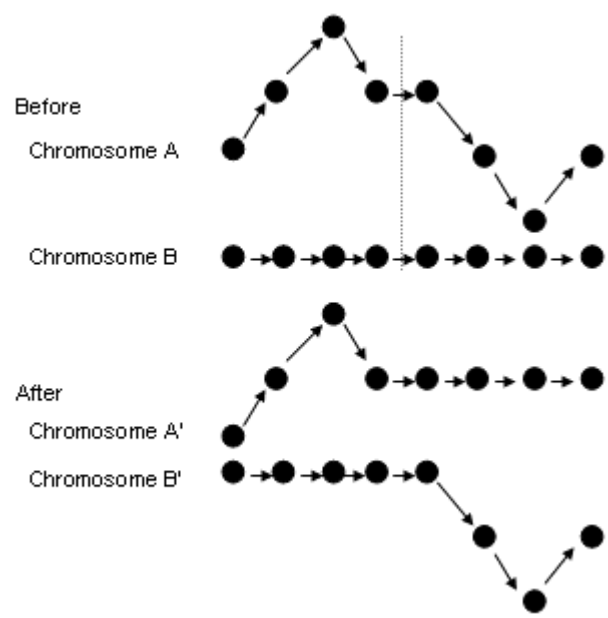


Figure 5. The effects of crossover on the end points

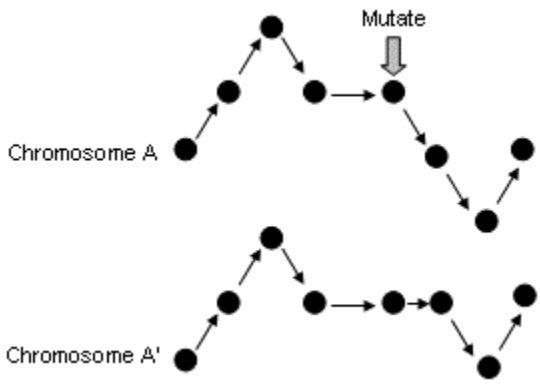


Figure 6. Mutation of the value encoded in a gene, resulting in a different end point for a silhouette line

3.1.3 Discussion

Depending on how much detail we want in the resultant silhouette, the number of line segments used can be adjusted. The maximum amount of useful detail is limited by the resolution of the height field that will be generated in the second phase – if the silhouette has details finer than this limit, they will have no contribution to the output height field.

Since, in real-world terrain, a single region may be adjacent to several other regions of different types, and they may transition differently into each other, we allow the individual boundary segments in the silhouette of a particular region to have different *S* values for their fitness evaluation criteria.

3.2 Terrain Height Field Generation

Terrain silhouettes generated from the processes described in the section 3.1 are used as guidelines in this stage. Each region enclosed by a set of boundaries is assigned by the user a particular terrain type. Instead of trying to generate terrain features and height fields from scratch, such as is the case in most fractal-based algorithms, we use a collection of terrain type sample height fields and features as building blocks for our GA population, and manipulate these samples to generate new configurations. These input examples can be taken from real-world GIS data scans, or from user-created samples. Since the input samples may be taken directly from real-world surveying data, our approach has an inherent advantage with respect to realism over fractal-based methods. For example, if a terrain type is set to be "mountain foothills", we can use satellite-scans of regions identified as mountain foothills to initialize our data.

3.2.1 Encoding/Decoding

Each chromosome represents a candidate solution (height field) corresponding to the silhouettes generated in the earlier phase. A single gene represents a small, localized cluster of height values with a "center point" in the generated height field and an "area of influence" (for example, a 16-pixel-diameter circle) as shown in Figure 7.

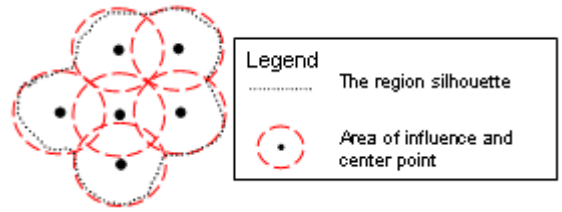


Figure 7. Overlapping areas of influence for a particular terrain silhouette

The genes are organized into a uniform, rectangular grid, such that their areas of influence overlap with one another and together they cover the entire region (height values in overlapped areas are calculated by combining the height values of each overlapping gene, according to a blending function). The main reason for allowing overlap among the genes is to minimize visual artifacts: if the region enclosed by the silhouette were partitioned into non-overlapping sub-regions, there would be inevitable, unnatural discontinuities at the gene boundaries. Although different shapes could be used for the area of influence, we chose a circular Gaussian function because of its good blending characteristics. The GA used in this phase is implemented as follows:

1. Initialize *m* chromosomes for the population, giving each gene in the grid a randomly selected chunk of height field data of the appropriate terrain type from the database.
2. Evaluate the fitness of each chromosome in the population.
3. Perform crossovers with probability p_c and mutations with probability p_m.
4. Replace the current population with the new population.
5. Go to step 4 until the end of the evolution process
6. Pick the strongest individual from the population and generate the result height field from it by applying all of the operations encoded in the chromosome.

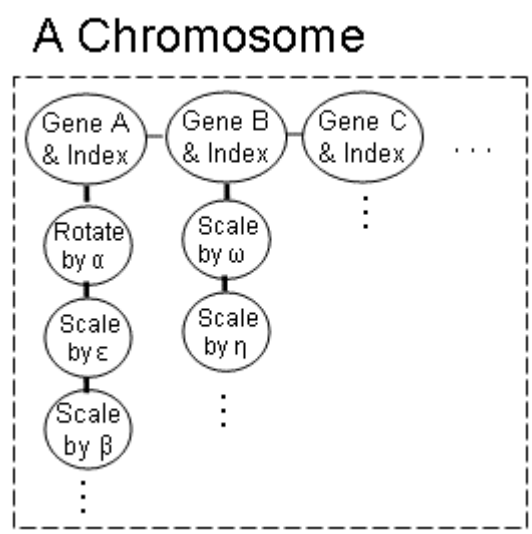


Figure 8. Encoding of a chromosome

3.2.2 Genetic Operators

The crossover and mutation operators are similar to the standard GA operators except that they operate on lists of terrain transformation operations, such as rotating a patch of terrain to a different orientation and scaling the height values of a terrain patch (Figure 8). From the chromosomes' perspective, these operators function identically to the standard operators in that they facilitate the exchange or mutation of encoded genetic material. From the perspective of the generated height field, they are best understood as sequences of image processing instructions, such as translating, rotating and scaling the height data (Figure 9).

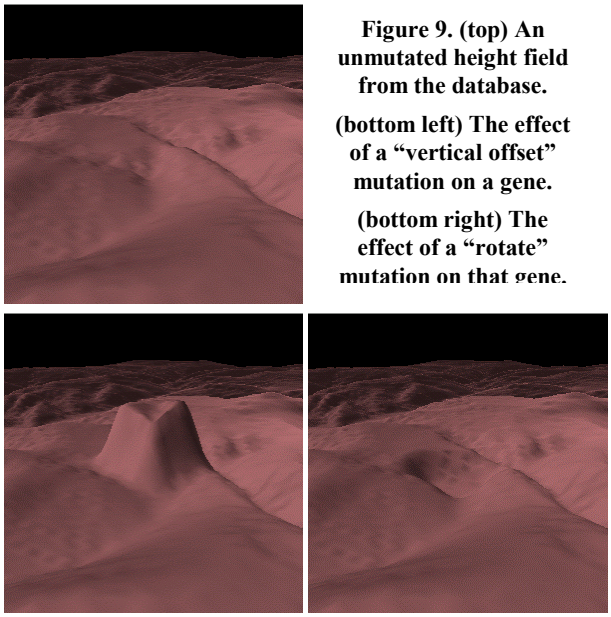


Figure 9. (top) An unmutated height field from the database.

(bottom left) The effect of a "vertical offset" mutation on a gene.

(bottom right) The effect of a "rotate" mutation on that gene.

The crossover operator effectively swaps regions of pixels between chromosomes (Figure 10). The various mutation operators implement image transformations, such as translation, rotation and scaling of the elevation pixel data[JK31]. The mutations can alter the transformation parameters of the image processing operations, or insert new image processing operations into the list of sequence carried by a gene (Figure 11).

3.2.3 Fitness Evaluation[DO32]

The fitness evaluation is somewhat more complicated in this phase than in the previous, because in order to be "good", a height field must satisfy a number of constraints. In particular, the following will be true of a "good" height field:

1. Each region in the terrain resembles the example terrains in the terrain database for its terrain type (i.e., "mountainous" regions look like the examples of "mountains" that we already have, "hills" look like our example "hills", etc.).
2. The boundaries between regions of different terrain types are reasonable. There are no sharp "drop offs" at the boundaries between terrain types.

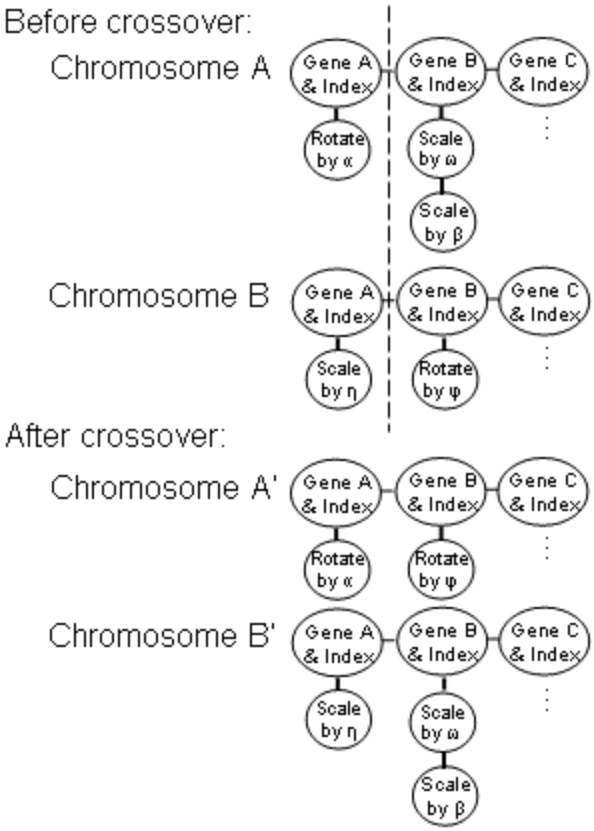


Figure 10. Crossing over the image processing instructions between genes

Each terrain type possesses a number of measurable characteristics, against which a region of that type can be validated. The degree to which these characteristics extracted from the generated region match those measured from the examples in the terrain database gives a measure of how "like" the source examples our generated region is. The more our generated height field resembles the sample inputs in these characteristics, the higher its fitness will be. The minimum, maximum, mean, and variance of the sample elevations are good candidates for meaningful measurements, as are the slope, frequency content (derived from the Fourier transform of the height field) and the density and size of semantic "features", such as valleys, plateaus, ridges and edges (extracted using conventional computer vision techniques).

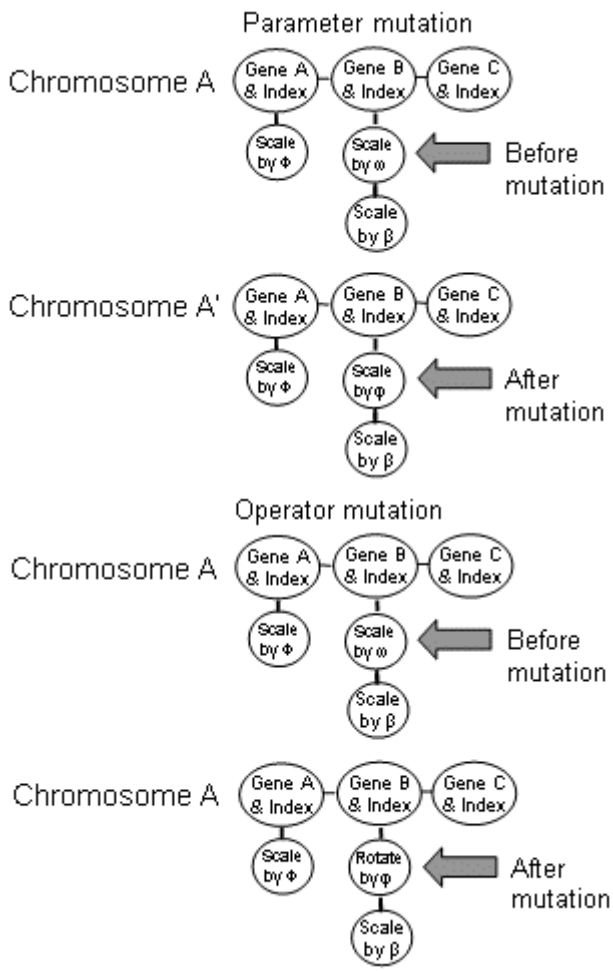


Figure 11. Mutation operators can alter the parameters of the image processing operations.

One difficulty of applying a GA to optimize terrain height fields is that, since the terrain can be adjusted in so many ways, and since the local adjustment of a single gene makes such a small contribution to the global, overall fitness of the terrain, convergence is likely to be very slow using a scalar-valued fitness evaluation, since so much information is lost in aggregating the regional fitness values. To address this, rather than keeping only the aggregate (global) fitness value, we also retain the individual fitness values for each region, in order to provide some probabilistic guidance to the mutations and crossovers in the next evolution cycle.

3.2.4 Discussion

Encoding all of the terrain-related data into each chromosome would be expensive since each chromosome would then contain a complete 3D terrain (actually, *more* than a complete terrain since the genes overlap one another); this is a primary reason for encoding only the sequence of transformation operations. By allowing the chromosomes to contain only the *instructions* for generating a height field, it is feasible to use a larger population for the evolution process. In our experiments, we used between 5 and 15 chromosomes, evolved over 10 to 20 cycles.

Since (like all genetic algorithms) the height field generation algorithm is inherently random, the terrains generated from two separate runs of the algorithm will not, in general, be the same, even if they use the same map. While this has the benefit of allowing an infinite number of variations to be created, it would also inhibit the user's design process, as each successive run would produce an entirely new terrain, even if the user had made only a small tweak to the map between runs. To control this, the seed for the random number generator can be kept the same across separate runs of the algorithm, allowing the same terrain to be regenerated as many times as desired.

4. FUTURE WORK

The silhouette generation process has several shortcomings, in that a single run of the GA is unable to generate some types of terrain features that can be found in nature due to the constraint we placed on the allowable range of θ. We could further improve the performance of the GA used in this phase by first generating silhouette lines of arbitrary length and θ, then extracting from it line segments having the desired length to initialize our population. A complexity parameter could also be incorporated into the GA used in the generation processes to better approximate the simplicity or complexity of certain terrain silhouettes, such as that exhibited by deserts or shorelines.

Currently, the terrain samples for the different terrain types used in the terrain height field generation phase are all hand-picked. We are experimenting with various image and pattern analysis techniques such as Fourier analysis, image feature detection, and perceptron-based classification in an attempt to automate this process such that the system can automatically or semi-automatically categorize a given set of terrain samples into different logical terrain types and determine the relevant characteristics of each, further simplifying the process from a user's perspective. We are also considering methods of identifying the presence of multiple terrain types within a single source sample. Such improvements in our methods of analyzing terrain types will allow us to refine our fitness evaluation algorithm in the height field generation stage.

We believe that the fitness evaluation and mutation operators can be further improved to better track the formation of terrain features spanning multiple genes, allowing entire features to be crossed over *en bloc* and resulting in faster convergence properties. More extensive feature tracking would also enable the formation of rivers and such (which impose additional constraints, such as being continuous across region boundaries and always flowing downhill).

5. CONCLUSIONS

We have presented a new approach to terrain generation using genetic algorithms. Instead of trying to solve the problem as a whole, we break it down into two phases: terrain silhouette generation, and terrain height field generation. In contrast to fractal-based algorithms, our approach provides better control over the shape and layout of the resultant terrain, through the placement of regions of different terrain types. We provide a balance between user input and real-world data capture unmatched by previous approaches.

There is still much room for improvement, as discussed in the future work section. Our experimentations in automatic terrain sample categorization can further improve the usability of this approach and the fitness evaluation function for the chromosomes.

6. ACKNOWLEDGMENTS[JK36]

This research is supported in part by the Humanities Informatics Initiative, Telecommunications and Informatics Task Force, Texas A&M University, and by NSF grant CCR-0220047.

7. REFERENCES

[1] Baeck, T., Fogel, D. B., Michalewicz Z. and Back, T. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing; 2000.

[2] Burke, Carl. *Generating Terrain*. http://www.geocities.com/Area51/6902/terrain.html.

[3] Duchaineau, Mark et al. *ROAMing Terrain: Real-time Optimally Adapting Meshes*. In proceedings of IEEE Visualization '97. IEEE Computer Society Press, 1997.

[4] *Erosion 3D*. Schmidt, Walter A. (dev). http://www.geog.fu-berlin.de/~erosion/

[5] Fernandez, António Ramires. *Lighthouse 3D - Terrain Tutorial*. http://www.lighthouse3d.com/opengl/terrain/index.php3.

[6] *GeoCommunity*. ThinkBurst Media, Inc. http://data.geocomm.com/

[7] Kelley, A., Malin, M. and Nielson, G. *Terrain Simulation Using a Model of Stream Erosion*. ACM Computer Graphics Vol. 22, No. 4, August 1988, pp.263-8

[8] *MojoWorld*. Pandromeda, Inc. (dev). http://www.pandromeda.com/

[9] Mitchell M. *An Introduction to Genetic Algorithms*. The MIT Press, MA, 1999.

[10] Peitgen, H., Jurgens, H., Saupe D., Jnrgens H. and Jurgens H. *Chaos and Fractals*. Springer-Verlag, 02-edition, 2004.

[11] Prusinkiewicz, Przemyslaw and Hammel, Mark. *A Fractal Model of Mountains with Rivers*. In proceedings of Graphics Interface '93. , 1993.

[12] *SimCity 4*. Electronic Arts (pub). 2003.

[13] *Terragen*. Planetside Software (dev). http://www.planetside.co.uk/terragen/

[14] Ulrich, Thatcher. *Rendering Massive Terrains Using Chunked Level of Detail Control*. In proceedings of ACM SIGGRAPH 2001. , 2002.

[15] *Unreal Tournament 2004*. Epic Games, Inc. (dev). Atari, Inc. (pub). http://www.unrealtournament.com/

[16] USGS. *National Mapping Program Standards*. http://rockyweb.cr.usgs.gov/nmpstds/demstds.html. United States Geological Survey, 2003.

[17] WERU. *Wind Erosion Simulation Models*. http://www.weru.ksu.edu/weps.html. Wind Erosion Research Unit, Kansas State University, 2003.