An Algorithm for Automated Fractal Terrain Deformation

S. Stachniak
Alias
Toronto, Canada
sstachniak@alias.com

W. Stuerzlinger
York University
Toronto, Canada
www.cs.yorku.ca/~wolfgang/

Abstract

Fractal terrains provide an easy way to generate realistic landscapes. There are several methods to generate fractal terrains, but none of those algorithms allow the user much flexibility in controlling the shape or properties of the final outcome. A few methods to modify fractal terrains have been previously proposed, both algorithm-based as well as by hand editing, but none of these provide a general solution.

In this work, we present a new algorithm for fractal terrain deformation. We present a general solution that can be applied to a wide variety of deformations. Our approach employs stochastic local search to identify a sequence of local modifications, which deform the fractal terrain to conform to a set of specified constraints. The presented results show that the new method can incorporate multiple constraints simultaneously, while still preserving the natural look of the fractal terrain.

Keywords: (according to ACM CCS): I.3.7 [Computer Graphics, Three-Dimensional Graphics and Realism]: *Fractals*, I.2.8 [Problem Solving, Control Methods, and Search] *Graph and tree search strategies*

1. Introduction

Terrain modeling plays an important role in computer graphics. The automatic generation of terrains has many applications in areas ranging from landscape generation for media, the generation of random environments for games, and the generation of terrains for various kinds of training simulators.

There are several ways to obtain terrain data. One source is digital elevation models (DEM's) generated by surveys, but this limits terrains to those that currently exist in reality, not those that lie in our imaginations. Another option is the automatic creation of terrains by fractal methods [Man77,Lew87]. However, these algorithms are very unpredictable in that the user has very little control over the resulting terrain height map. There is no easy way for the user to generate a terrain that has a particular shape, or provides the surroundings to a pre-specified road track, deform a terrain to have exactly one lake, a canyon, etc…

While several algorithms for terrain deformation have been presented [ST89, VML97, FOMC02], all suffer from the abovementioned drawbacks. Currently, the most widely employed method for fractal terrain deformation is still modification by hand. The application of terrain deformations, such as a 2D Gaussian multiplication on the terrain, requires a decision as to where to deform the terrain, and by how much. This task is not trivial, as most humans cannot reliably predict the exact effect of an operation let alone create a sequence of such operations to satisfy their goals. Often, modelers laboriously deform terrains using a trial and error approach. The computational power of today's systems can easily generate extremely complex models with millions (or even billions) of polygons. However the capacity of humans to generate and modify such models does not increase [Ebe96]. This means that although we can expect humans to skillfully touch-up small areas of a terrain, we cannot expect a user to model entire planets or vast landscapes without aid.

The existence of a robust algorithm for automatic terrain deformation is of considerable benefit to the graphics community. Whereas previously, artists often began by generating random fractal terrains, trying to find one that approximates the shape they are trying to model, they can now spend their time refining a deformed terrain.

This paper presents an algorithm for automated fractal terrain deformation. Because of its generality, it is highly flexible and easily configurable. The algorithm computes a near optimal sequence of local and global modifications to deform a fractal terrain so that it satisfies user-defined constraints. No work to date has solved the general problem of terrain deformation in such a flexible manner. A star shaped island, a landmass with a lake, self-tiling terrains, and thousands of other types of terrains can be generated automatically using a single unified deformation algorithm.

2. Review of Related Work

Ever since Mandelbrot proposed the use of fractals as a basis for simulating natural scenes and phenomena [Man77], researchers have tried to generate and render such complex models. Although methods for the synthesis of fractal terrains produce realistic-looking data [Lew87], they do not provide easy ways for modifying the results.

Solving the general problem of constraining fractal terrains has been studied previously. [ST89] present a method to approximate a coarse spline mesh with a fractal terrain. Due to the use of a coarse spline mesh, only large-scale modifications are possible. A more recent method has been proposed for fractal deformation using displacement vectors [FOMC02]. Given a displacement grid, a fractal is deformed to render into a particular shape. Although high quality 2D fractal deformations are presented, it is unclear if 3D deformations would retain their natural look. Displacement vectors in 3D are very similar to the coarse spline meshes proposed in [ST89] and have the same drawbacks.

Another approach based on a Gibbs sampler [VML97] constrains fractal terrains to pass through a set of pre-defined points. However, there is no guarantee as to the shape of the terrain between the points. Hence it is hard to produce a precise result (e.g. a completely flat road), without having to provide a large number of points, which in turn defeats the purpose of automated deformation.

Solutions to specific types of deformations have also been studied previously. One approach for this uses a squig-curve model to generate rivers during the fractal terrain generation process [PH93]. Procedural attempts to generate erosion effects that simulate water flow in an existing terrain have also been presented. There, physically based models of hydraulic and thermal erosion and sediment movement to simulate the erosion due to water flow are used to modify the terrain [KMN88, MKM89, BF01, CMF98]. None of these solutions extend past their specific domain.

Another related topic is texture synthesis. These approaches use procedural techniques for the generation of textures with varying properties (see e.g. [HB95, GM85, NC99, TZL*02]). However, all these approaches work at local scales, whereas the problem introduced in this paper may require changes at all scales. Initial work to blend multiple texture types for more global control has been presented [ZZV*03], but even this work is clearly not general enough to address the problem introduced. Furthermore, texture synthesis is not designed for the creation of three-dimensional terrains and in general textures visualized as three-dimensional terrains do not necessarily look realistic. One exception is the work on hypertextures [PH89], which are targeted at three-dimensional applications. However, it is unclear how one can modify this approach to adapt the result to specified constraints.

Several three-dimensional modeling suites allow the user to work with meshes. Although such software allows the user to modify a mesh at the vertex level, there is very little support for modifying a fractal terrain to conform to a predefined shape, such as a road. This means that the user has to adapt a trial and error approach as one must first find a terrain that approximately conforms to the desired criteria and then modify that terrain by hand. Although one can generate very realistic and convincing results with such software, this is a laborious endeavor.

In the artistic community, manual approaches that use image-processing methods to modify a terrain so that a road can be inserted are well known. For example [Fry04] employs posterization of fractal terrains to insert a road into the terrain. While the results look great, the approach is limited by the random shape of the fractal terrain and does not allow the insertion of a road of a predefined shape (e.g. a road that

doesn't exceed a certain gradient) into an arbitrary terrain.

2.1.Contribution

In summary, we can say that no previous work provides a method for general deformation of fractal terrains. Although algorithms exist, which can generate approximate solutions to some of the presented problems, no work to date has successfully addressed general constrained terrain deformation.

This paper presents a new method for automatically deforming a fractal terrain to satisfy a wide array of possible constraints. Constraints are defined by using a highly expressive mathematical framework. Our algorithm's strength lies in its generality, and can produce a wider array of results than any other work to date.

3. General deformation of terrains with constraints

Our approach begins with a terrain defined as a height map. This terrain is usually obtained by fractal terrain generation and is then deformed according to a set of constraints. These constraints are expressed as functions that define how close the terrain has come to satisfying the constraint.

Our method requires two inputs. The first is the terrain, T represented as a two-dimensional array of height values. Secondly, we require a fitness function F defined over the terrain T, which expresses the constraints to be imposed on the terrain. More specifically, $F(T)$ is a measure that intuitively describes "how far" a terrain T is from satisfying the constraint description. All constraints in our algorithm are encoded in this general fitness function, which allows us to specify multiple constraints simultaneously.

Our method also requires a deformation operation. In general, it is desirable to produce natural looking deformations and to preserve the original terrain as much as possible. While there are many options for this, we choose multiplication with a truncated Gaussian kernel as our deformation operator to reduce potential artifacts, while still allowing for efficiency – the algorithm operates on terrains with thousands or even millions of vertices. This deformation is then applied to a region of the terrain by first scaling the Gaussian kernel to the desired amplitude and radius. More precisely, given a vertex v, a desired amplitude a, and a modification radius r, the operator multiplies the terrain with a Gaussian function of radius r centered at terrain vertex v with amplitude $(a–v.height)$. This operation was chosen because the distribution forces point v to the desired height, while maintaining the overall shape of the original terrain without introducing artifacts. Other deformation operations could be employed, but we have found that the Gaussian kernel adequately satisfies our needs.

Our algorithm is based on deforming the original terrain T into a new terrain T'. T' is defined as a height map that fits the constraints, i.e. it has the property:

$$F(T') = 0$$

Because we have restricted deformations to a single operation (the Gaussian multiplication) the problem of finding a general solution to terrain deformation is reduced to a search for a sequence of Gaussian multiplications that attempt to minimize F. To further simplify this search we parameterize a general Gaussian kernel as a triple (location, amplitude, and radius). Our goal is to search for an optimal sequence of parameter triples that deform

the original terrain into *T'*. However, even with these assumptions, the search space is infinite, and a bounded search of any practical size is intractable. There is also no guarantee that a terrain can be deformed to satisfy the fitness function. Therefore, instead of concentrating on finding the global optimum, we concentrate on finding approximations that are within epsilon of the ideal solution:

F(T') <= epsilon

Our approach makes use of stochastic local search to find a sequence of operations that converge the fitness towards epsilon. In the following sections, we discuss how we define constraints via fitness functions, how we search for deformation sequences, and finally how we optimize the search.

3.1.Constraint Definition

Our algorithm searches for a sequence of "good" deformations, where the definition of a good deformation is how it relates to the evaluation of the fitness function. In this section we present how constraints are encoded via fitness functions. Three distinct examples are presented, illustrating the generality and power of this method. First, we discuss how a terrain can be modified to match a predefined shape, then we modify a terrain to match a predefined road, and finally we discuss how we can adapt a terrain to fit the edge of a second terrain.

While we present three specific examples, they serve to emphasize the generality of the approach. The range of deformation examples presented in this paper should make the generality of the approach clear.

For reasons of efficiency, we prefer fitness functions that are computable in constant time per vertex, because fitness functions will be evaluated for a large number of operation parameters during the search. For practically interesting terrain sizes this necessitates fast computations and makes functions that take (much) more than constant time per vertex undesirable. Hence, we choose to define *F* as a distribution of penalties per vertex, which is highly expressive, yet easily computable in constant time. More precisely we define *F(T)* to be the sum of all penalties over all vertices for the given terrain.

$$F(T) = \sum_{x,y \in T} F(x,y) \quad (1)$$

A simple example is to modify a terrain to match an existing height map. If the desired heights are stored in a two-dimensional array, we can simply penalize vertices by their distance from the desired shape.

$$F(x,y) = abs(T[x][y] - height[x][y])^2 \quad (2)$$

To present a more interesting example for shape modification we introduce a function that constrains a terrain to fit a particular form (such as the shape of an island). If the shape is defined as a two-dimensional bit-mask that specifies which areas should be above a water level, we define *F* to penalize each vertex whose height does not conform to the desired shape.

$$F(x,y) = \begin{cases} (T[x][y] > waterLevel) \wedge (mask[x][y] = 1) \Rightarrow 0 \\ (T[x][y] < waterLevel) \wedge (mask[x][y] = 0) \Rightarrow 0 \\ otherwise \Rightarrow (T[x][y] - waterLevel)^2 \end{cases} \quad (3)$$

Every vertex is penalized if it is above or below the water level when it should not. Bit masks for more than one height threshold could be applied to produce more complicated terrains.

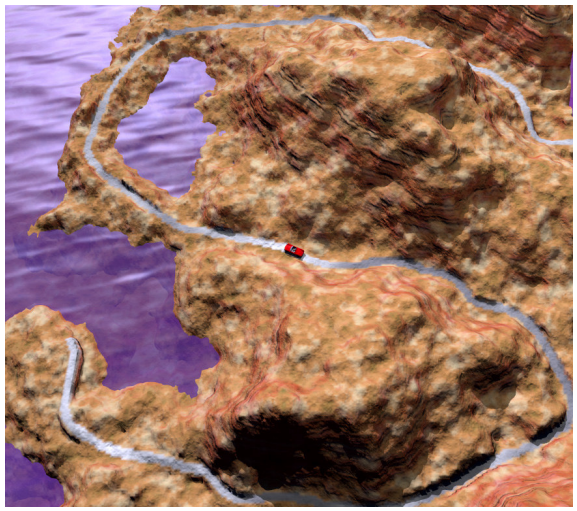


Figure 1: *An arbitrary terrain deformed to accommodate a flat S-shaped road using fitness function 4.*

Another class of constraints is the creation of terrains for modeling specific roads or rivers. The simplest idea is to generate a random terrain, and then search for a section that can easily accommodate a road. This was already presented by previous work (e.g. [Fry04]). A much more difficult task is to modify a terrain to fit a particular predefined road. As an example consider a completely flat road with a predefined shape that is set into a very hilly terrain (Figure 1). The fitness function for this terrain required that all vertices that form the road be constrained to a constant height. All other vertices are to retain their original shape as much as possible (to avoid global flattening of the terrain!). We consider all vertices within a certain distance from the road as path vertices. The constraint function then takes the following form:

$$F(x,y) = \begin{cases} if\,(path(x,y)=1) \Rightarrow (T[x][y] - roadHeight)^2 \\ if\,(path(x,y)=0) \Rightarrow (T[x][y] - origT[x][y])^2 \end{cases} \quad (4)$$

Vertices along the road are penalized by the square of their divergence from the desired height of the road, whereas all other vertices are penalized by the square of their divergence from the original terrain. For a more general application, one can define arbitrary roads by creating a function that looks at a constant set of neighboring path nodes, while incorporating slope and oscillation as a penalty measure (e.g. to get a road with a predefined gradient).

Finally, we discuss how to adapt a terrain T_1 to match the edge of a second terrain T_2, e.g. to extend a terrain further or to create a self-tiling terrain patch. To solve this task, we need a function that compares height values along the edge of T_1 to height values along the edge of T_2 and penalizes badly matched vertices.

$$F(x,y) = W(y) * (T_1[xMAX][y] - T_2[xMIN][y])^2 \quad (5)$$

A simple weighting function *W(y)* is used to ensure that values close to the edges are scaled appropriated to yield an equal distribution.

Fitness functions provide a mechanism for defining terrain deformations with utmost versatility. They can be as complex as explicitly defining exact vertex heights for a majority of terrain vertices, to simple functions defining the approximate position of just one point.

3.2.Multiple Simultaneous Constraints

The power of our algorithm lies in the definition of fitness functions as terrain constraints. It is simple to combine fitness functions via mathematical means such as multiplication to combine multiple constraints. Fitness functions must be normalized in order to ensure that the algorithm equally addresses all constraints.

3.3.Stochastic-Local-Search for Deformations

In our search for a sequence of deformation operations, the fitness function is treated as a quantitative measure that specifies how far a terrain is from conforming to the desired result. Solutions to automated terrain deformation are computed with a search for a series of deformation operations to transform a terrain *T* to another *T'*, which minimize the fitness function *F*. Given that any one of thousands of vertices can be modified by a deformation of arbitrary radius and amplitude the search space is clearly infinite. Sampling a finite set of amplitudes and radii still produce search trees that are have enormous numbers of children.

A good method for approximating search in such cases is stochastic local search (SLS) [Gu92, SLM92], a method whose roots are based in simulated annealing [KGV82]. Though SLS will not guarantee the perfect result (a terrain that matches the constraints perfectly), the method guarantees high quality approximations using a greedy approach. However, a naïve greedy search can reach local minima. Also, in our case, where the number of deformations is limited by processing speed, a greedy search may not yield uniform deformation distribution. Furthermore, in rare cases deformation may stall if deformation actions ping-pong between each other. Some SLS techniques solve these problems by introducing random noise to the search. Our algorithm uses noise to avoid these problems as well. We introduce an empirically set constant, *p,* which determines the amount of noise to be added to the search.

Our method generates a search space by first performing a large number of possible deformations, i.e. all vertices multiplied by a Gaussian kernel given for a set of amplitudes and radii. A table of potential deformations with resulting fitness values is computed per iteration. SLS chooses a deformation resulting in the best fitness with probability *p*, whereas a sub-optimal choice is selected with probability *1-p*. To prevent unrecoverable deformations, we limit sub-optimal choices to a certain percentage of best deformations. Unlike satisfiability problems (for which SLS has been shown to be quite effective and where many iterations can be performed), the search for terrain deformations must be limited to a relatively small number of iterations because the generation of even one level of the search tree is computationally expensive. Fractal terrain deformation, however, does not impose the restrictions of satisfiability as the definition of a satisfying terrain is one that conforms to the high-level constraint, and therefore we seek solutions only within epsilon. The goal is to satisfy the fitness function's high-level constraint. In practice, we either terminate iteration when the approximation has reached epsilon, or when the iteration count has breached some threshold.

It is vital to reduce the search space in any way possible. Given that the set of all sequences of all possible operations is infinite, we first reduce the infinite search space to a finite one by sampling a finite subset of possible amplitudes and radii. By controlling the amount of sampling we can empirically control the tradeoff between accuracy and speed.

Since a deformation can be centered on any terrain vertex, brute force search of any reasonable domain would be prohibitively slow. We therefore reduce vertex counts by pruning vertices in terms of their ability to influence *F*. For this we use a general method for determining a set of good

candidates. We begin by sampling the grid to provide a minimal and uniform distribution across the terrain. We then add vertices to this set by performing quad-tree subdivision in areas of interest. Here, we define certain regions to be modifiable, such as vertices along a path or all vertices surrounding a shape mask. We then subdivide the domain one level further if the partitioned space contains any area marked modifiable. Figure 2 demonstrates how this procedure identifies areas of interest.

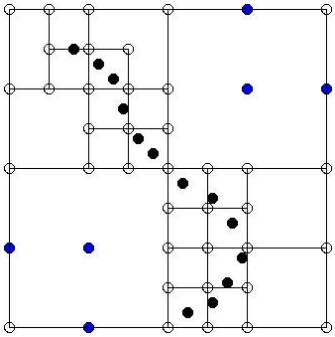


Figure 2: *We select candidate vertices by first adding all vertices along interesting features, in this case along a defined path (black dots). Quad tree subdivision then provides vertices concentrating around that path (empty circles). A coarse sampling of the entire grid ensures a uniform distribution of vertices over the remaining terrain (blue dots).*

In this way we significantly reduce the number of deformable vertices, while ensuring that vertices capable of greatly influencing deformation are included in the candidate set. Even with this technique, the search tree remains large. As an example, consider that this technique may reduce the potential number of deformations on a terrain with 257^2 vertices to only thousand modifiable vertices. With 25 possible amplitudes and 7 possible radii (empirical values that we have found to produce good results), a search tree of height 3 will still contain roughly 30 billion nodes. The size of this search tree justifies our use of stochastic local search as a viable means of search.

3.4.Frequency Limitation

Although our method produces results that are aesthetically sound, we hope that in the future, systems that employ our method will run at interactive rates. Without reduction in computation time through pruning of candidate vertices, even the simplest deformation tasks can take upwards of several hours to complete.

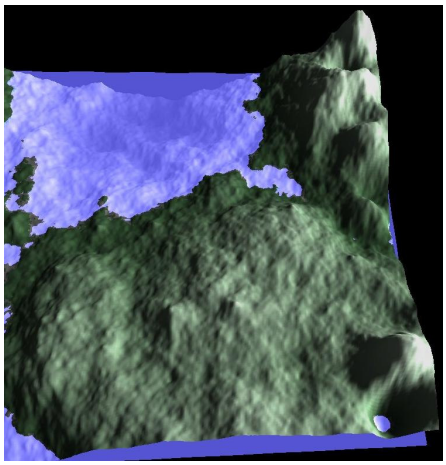


Figure 3: *This terrain shows artifacts that can occur if high frequency operations are performed. On the right side of this terrain the algorithm applied several local, high frequency, operations, which result in artificial bumps and dips.*

Initial implementations of our algorithm preferred to modify small regions (corresponding to high frequencies), because such modifications have little influence on surrounding vertices and consequently improve the fitness function at little cost. However this is a problem, as high frequency modifications may introduce

unwanted spikes in the terrain. Figure 3 demonstrates the corresponding artifacts. We practically eliminate all such artifacts by pruning all parameter combinations whose displacement is greater than a fixed proportion of the radius. A side effect of this pruning is a large reduction in overall search time.

3.5.Optimization with Prediction

In each iteration, a lot of work is performed when computing the fitness values for each possible deformation operation. The probability that a deformation resulting in a poor fitness value will be a top candidate in the next iteration is very small. However, removing such a deformation operation from the set of candidates is not acceptable, as that operation may be beneficial in future iterations. In order to reduce the overall amount of work when computing deformation fitness values, we introduce a technique that uses prediction to reduce computation.

We introduce a confidence value mapped to each possible deformation. This confidence value represents the probability that the specific operation will improve the terrain enough that it will be considered as a top candidate in the search. The first iteration initializes the confidence of all deformations to 1.0. It is unknown which deformations rank highly, and which deformations are poor. The confidence value for the deformation in the next iteration is computed as follows:

$$conf_{t+1}[i] = (1 - \min(1, \frac{C + (F[i] - \min F)}{\max F - \min F})) * conf_t[i] \quad (7)$$

where $conf_t[i]$ is the confidence of the deformation operation i at iteration t. The variables max$F$ and min$F$ are the best and worst fitness values computed for the previous iteration. C is an empirically set constant, which artificially increases the probability, ensuring that top candidates are always re-computed, and forcing all deformations to be re-computed after some time.

4. Results

The algorithm presented in this paper can deform arbitrary terrains to conform to specified constraints. In this section we demonstrate results generated with the constraints introduced in section 3. All presented results were generated with a noise value p = 0.65, which we found to solve all attempted deformation tasks. Amplitudes and radii were sampled to 25 heights and 7 radial distances.

In general, the use of predictions to speed up the computations results in a speed-up of approximately 400%. Furthermore, we found that the use of this optimization results in deformations that are nearly identical to those generated without prediction, and unnoticeably different to the naked eye.

4.1.Shape Conforming Terrain Deformation

The first example demonstrates the ability of our method to modify terrain data at global scales. Using fitness function (3), we can generate terrains that fit arbitrary shapes. Figure 6 shows a star shaped terrain automatically deformed using our algorithm. The bit-mask used was a five-pointed star shape. Generation of this model took 45 minutes on a 1.7Ghz Intel PC.

4.2.Deforming a Terrain to Match a Path

Some manual modeling techniques can produce natural looking terrain with particular features, such as a path or road [Fry04]. However these features are always dependent on the terrain they modify. The

terrain has always dictated the shape of the road.

Our method decouples this dependence. As an example, we show how to deform a terrain so that it has an S-shaped path. The road shapes the terrain. For this we use the fitness function defined in equation (4). The result is shown in Figure 1. Effectively, the method carved a space for the road to pass through in the right hand side of the terrain (which was a hill), and created a land bridge in the top left corner (where there was sea), to ensure a realistic looking road. Deformation of this terrain model was accomplished in under 25 minutes.

4.3.Terrain Blending

Current implementations of terrain blending work on the principle of texture blending. They involve the application of blending functions to one or more textures. Our algorithm can merge terrains by defining a fitness function as in equation (5). This not only produces terrains that merge seamlessly but can also reduce artifacts along edges. Our approach deforms terrains to match terrain edges instead of terrain blending which simply blends height maps and creates artifacts on severely disjoint terrains. Figure 7 depicts two terrain-merging results, each of which took under 6 minutes to compute.

4.4.Merging Fitness Functions

The generality of our approach is best demonstrated by the fact that we can combine fitness functions. To demonstrate the ability to merge fitness functions, we generated a star shaped terrain with a road network passing through each star end-point. Note that the fitness function for road generation requires that the original terrain be preserved as much as possible.

Figure 4: *A star shaped terrain with roads extending to each endpoint. Deformed using a fitness function that combines shape and road constraints.*

We ran the combined fitness function for 200 iterations with prediction. The results are presented in figure 4, and the terrain took 1.25 hours to generate. As expected, road paths are generated and hills are split to allow the road to pass through to the end points of the star.

Combined fitness functions clearly result in more computational effort, as multiple features need to be accommodated. This requires that the search space be explored more widely and in general more iterations of the SLS algorithm are required.

4.5.Terrain compression

Another benefit of our algorithm is that it provides an efficient way to store deformed terrains. Compression requires the storage the original terrain (e.g. via its random seed and size) and the sequence of deformations (location, amplitude, and radius) that produced the final result. Reconstruction involves parsing a list of Gaussian deformations, and applying them to the terrain. Such an operation is computed

extremely quickly, and therefore, terrain reconstruction speeds are a non-issue.

We compared our compression scheme to JPEG compression in Photoshop™ on setting 9 of 12 (high compression rate). Figure 5 depicts image compression sizes for average terrain images using our compression scheme, and the jpg compression scheme.

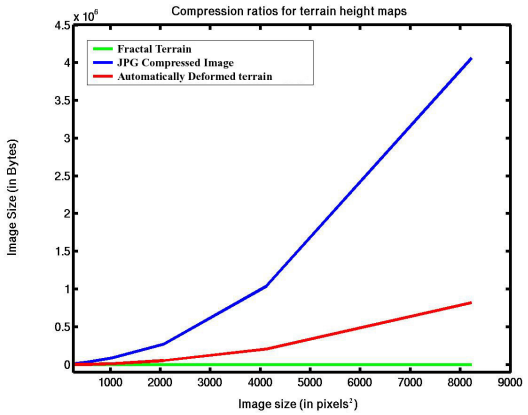


Figure 5: *File sizes for different methods of terrain data compression.*

Regenerating the original fractal terrain from its seed and applying the recorded sequence of deformations to the terrain decompress it. A fixed random lookup table is required for fractal terrain compression to be effective.

5. Conclusion and Future Work

In this paper we present a novel algorithm for automated deformation of fractal terrain data using stochastic local search. We have shown how simple functions can be used to describe complex constraints on terrains. Furthermore, we have demonstrated how intelligent search methods can be employed to minimize these constraint functions. A side effect of this search is a deformed terrain.

One of the drawbacks of our current implementation is execution speed. This reflects the tradeoff between the generality of our algorithm and the speed of specific solutions. In the future, we hope to exploit the fact that our approach is highly parallelizable. Because the fitness values computed for each vertex are solely dependant on the terrain at the given iteration, fitness computations can be computed independently of each other. With modern advances in hardware and multi-core processors, this algorithm should be able to deform terrains with highly complex constraints at interactive rates.

References

[BF01] Beneš B., Forsbach R.: Layered Data Structure For Visual Simulation Of Terrain Erosion. *IEEE Spring Conference on Computer Graphics*, 2001, 80-86.

[CMF98] Chiba N., Muraoka K., Fujita K.: An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery. *Journal of Visualization and Computer Animation* vol. 9 (1998), 185-194.

[Ebe96] Ebert D.S.: Advanced Modeling Techniques for Computer Graphics. *ACM Computing Surveys*, 1996, 153-156.

[Fry04] Fry R., Calyxa Bryce tutorials, http://calyxa.best.vwh.net/~calyxa/pearl/tutor.html.

[FOMC02] T. Fujimoto, Y. Ohno, K. Muraoka, N. Chiba: Fractal Deformation Using Displacement Vectors Based on Extended Iterated Shuffle Transformation. *The Journal of the Society for Art and Science*, Vol.1, No.3, pp.134-146, 2002

[GM85] Gagalowicz A., Ma S.: Model Driven Synthesis of Natural Textures for 3-D Scenes, *Eurographics*, 1985, 91-108.

[Gu92] Gu J.: Efficient Local Search for Very Large-Scale Satisfiability Problems. *ACM SIGART Bulletin* 1992, 3(1):8-12.

[HB95] Heeger D.J., Bergen J.R.: Pyramid Based Texture Analysis/Synthesis, *SIGGRAPH '95*, 229-238.

[HN01] Hoffmann J., Nebel B.: FF: The Fast-Forward Planning System. *AI magazine, 22(3)*, 2001, 57-62.

[KMN88] Kelly A.D., Malin M.C., Nielson G.M.: Terrain Simulation Using a Model of Stream Erosion. *SIGGRAPH '88*, 263-268.

[KGV82] Kirkpatrick S., Gelatt, C.D., Vecchi M.P.: Optimization by Simulated Annealing. *IBM Technical Research Report RC 9335,* 1982.

[Lew87] Lewis J.P.: Generalized Stochastic Subdivision. *ACM Transactions on Graphics*, 6(3), 1987, 167-190.

[Man77] Mandelbrot, B. The Fractal Geometry of Nature. *Freeman, San Francisco,* 1977.

[MKM89] Musgrave F.K, Kolb C.E, Mace R.S.: The Synthesis and Rendering of Eroded Fractal Terrains, *SIGGRAPH '89,* 41-50.

[NC99] Neyret F., Cani M.P.: Pattern-based texturing revisited, *SIGGRAPH '99*, 235-22.

[PH89] Perlin K., Hoffert E.: Hypertexture, *SIGGRAPH '89*, 253-62.

[PH93] Prusinkiewicz P., Hammel M.: A Fractal Model of Mountains with Rivers. *Graphics Interface 1993*, 174-180.

[SLM92] Selman B., Levesque H., Mitchell D.: A New Method for Solving Hard Satisfiability Problems. *Proceedings of AAAI 1992*, 440-446.

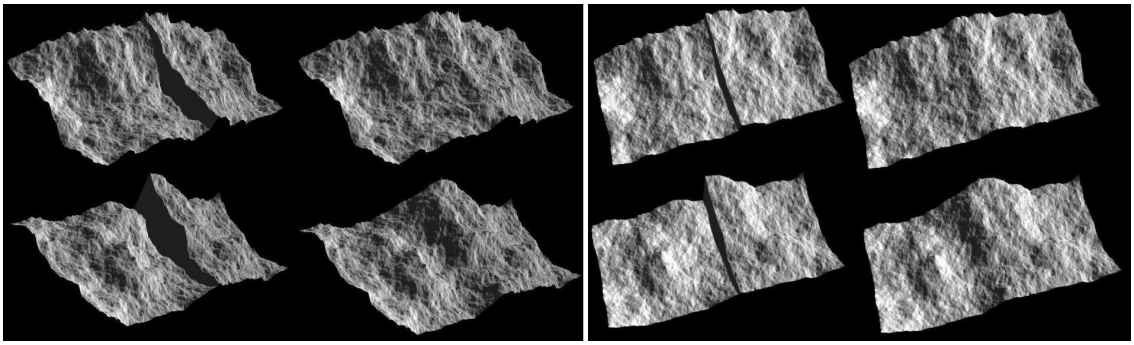
[ST89] Szeliski R., Terzopoulos D.: From Splines to Fractals. *SIGGRAPH '89*, 51-60.

[TZL*02] Tong X., Zhang J., Liu L., Wang X., Guo B., Shum H.Y.: *SIGGRAPH* 2002, 665-672.

[VML97] Vermuri B.C., Mandal C., Lai S.: A Fast Gibbs Sampler for Synthesizing Constrained Fractals. *IEEE Transactions on Visualization and Computer Graphics*, 3(4), 1997, 337-351.

[ZZV*03] Zhang J., Zhou K., Velho L., Guo B., Shum H.Y.: Synthesis of Progressivley-Variant Textures on Arbitrary Surfaces. *SIGGRAPH 2003,* 295-302.

Figure 6: *A Star shaped island obtained by deforming a randomly generated fractal terrain.*



(a) (b)

Figure 7: *3D views of merged terrain data, each consisting of* 257^2 *vertices. (a) Side view of merged terrain, before and after merging. (b) A different view of the same terrain models.*