# Storage-Free Terrain Simulation

Maurice Danaher & Warren Creemers

***Abstract:*** *This paper presents a method for simulating terrain by visualising landscape data in real-time. Current storage-based simulators are limited in the size and detail of the terrain that they can produce due to the storage of the terrain data on the hard disk. A method for terrain simulation is presented that is "storage-free". The method has been implemented in a simulator and the results produced are compared with two popular simulation programs. While the execution as expected is slower the map sizes produced are enormous in comparison to the others.*

***Key words:*** *Terrain, Simulation, Real-time.*

## INTRODUCTION

Terrain in a typical simulation program is created prior to program execution and stored to the hard disk ready for use. When the simulator is executed the terrain is loaded into the simulator, which is responsible for determining which areas of the terrain are used in the rendering pipeline. Examples of simulators using this approach include "Tread Marks" [8] and "Draken" [4].

The size of the explorable environment used in the simulation is limited to what is stored on the hard disk. Also the graphical quality of the landscape used is limited by the quality of the landscape stored in the simulations dataset.

This work addresses a problem with the classical approach to terrain simulation. The classical simulator must make a trade-off between the amount of detail present in the landscape and the size of the landscape. This trade-off exists because conventional storage devices are not large enough to store a landscape that is both detailed and large.

## TERRAIN STORAGE

To understand limitations of current terrain simulators it is necessary to explore how they store the landscapes that they render. Three main data formats used for the storage of terrain data are defined by the "National Imagery and Mapping Agency" [9], namely: (1) Height Field Grid (Referred to as a Digital Elevation Matrix (DEM)); (2) Triangular Irregular Network (TIN); (3) Digital Contour Line. Bitters [2] presents a detailed evaluation of these three formats according to their performance and storage characteristics.

Many video games implement a Digital Elevation Matrix for passing data into the rendering pipeline. A primary reason for adopting the Digital Elevation Matrix format is the fact that it has the fastest processing speed of the three possible formats. However the memory requirements of this algorithm are comparatively large. The algorithms used by a simulator using this format offer the advantage of being simple to implement. Unfortunately limitations are imposed on this format because as the matrix holds only one value at each field there is no way to store non-extruded features such as caves. Typical applications for this format include: geological surveys, interpretation of satellite photography, computer games and flight simulators.

The most pressing concern with storage of terrain is the amount of storage space required. When a Digital Elevation Matrix is utilized the storage space required is linearly proportional to the amount of terrain to be stored and exponentially proportional to the sampling resolution of the terrain. This makes high resolution terrains the most difficult to store.

Shown below in Figure 1 is the relationship between memory usage, area stored and sample resolution. Note that the axes of the graphs have been annotated to show requirements for certain simulation, display and storage tasks.

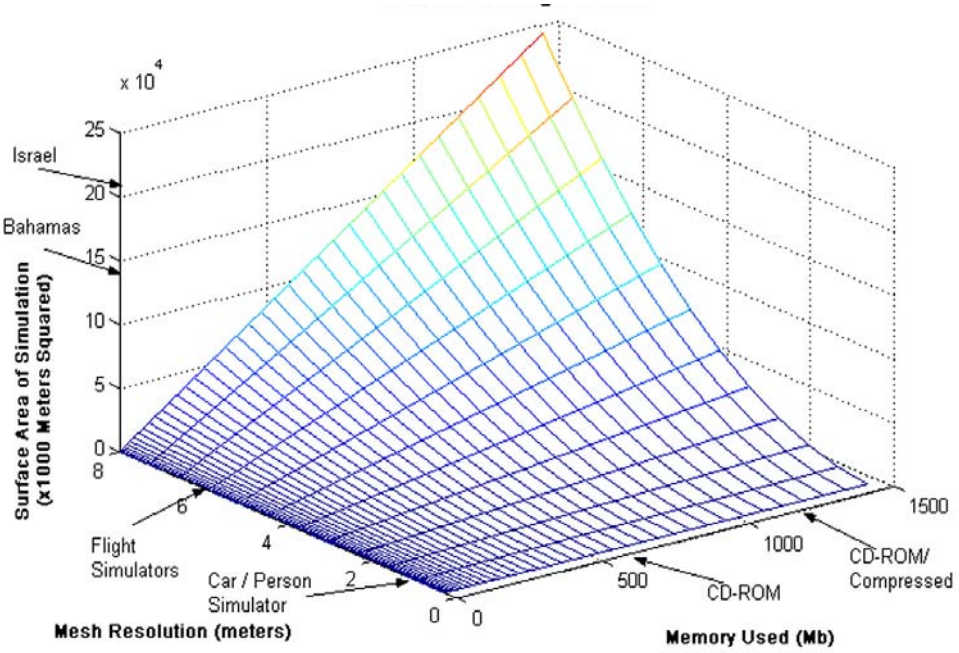


*Figure 1. Terrain Storage Costs*

## TERRAIN CREATION

A landscape can be considered a fractal surface with nearly infinite surface area. For a description of fractal surfaces see Mandelbrot [7].

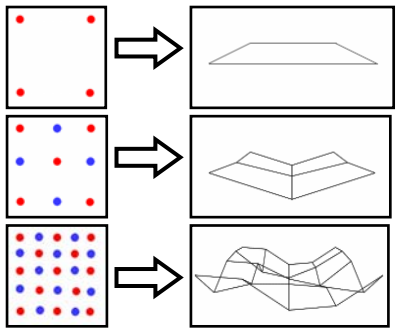

*Figure 2. Terrain Generation*

A form of fractional displacement known as Brownian motion (brown noise) is suitable for generating fractal landscapes by displacing a surface. To adapt the Brownian motion technique to three dimensions we follow the methods of Barnsley et al. [1]. This work can be applied directly to a height field grid. Brownian motion in three dimensions is achieved by a process called midpoint displacement. This process works by dividing four points of a square to produce smaller squares. Each point (corner) stores information on the height of the final grid. This process is shown in Figure 2, the dots on the left represent positions filled in a digital elevation matrix.

## PAGE MANAGEMENT SCHEME

The objective of page management is to divide a landscape into multiple blocks or pages which tile together to form a visible landscape. The midpoint displacement algorithm can be used to terraform (create terrain) a page of data in a page management scheme. By creating individual random landscapes for each page we have the basis for on demand

creation of landscapes. The use of pages also increases the speed of landscape creation by reducing the amount of overall recursion present in the creation of a terrain. These pages are created individually using corresponding boundary values and tiled together to form a terrain. The use of terrain pages prevents the midpoint displacement algorithm from having to generate the entire map at once, thus allowing for efficient generation of only the terrain that is needed.

A quasi white noise generator [6] is used to control the random seeding of terrain pages. This algorithm takes as an input a number representing the location of the terrain page and produces a unique seed for that page. Importantly the unique seeds appear to be randomly distributed, but the algorithm will alway produce the same output given the same inputs. This allows for terrain pages to be created identically each time they are revisited.

A major challenge to the terrain generation algorithm is the maintenance of terrain around the user's viewpoint. This algorithm is intended to only generate the terrain that is around the user's viewpoint and visible to the user. To achieve continuous generation a page management algorithm is constructed. The Offset Spherical Buffer is an approach devised by the authors especially for this work. For details for the Offset Spherical Buffer refer to Danaher and Creemers [3] and for information concerning existing page management algorithms see Eberly [5]. (Note that in the field of page management: a page is defined as a height field grid; a map is defined as a collection of tessellating pages; a submap is a collection of adjoining pages usually representing the user's visual vicinity.)

An entire map of pages (e.g. to cover a planet) is typically too large to be stored in memory. To work around this limitation this work maintains a collection of pages called a submap. This submap is defined with the user's point of view being in the centre of the submap. The submap is responsible for storing all terrain data visible to the user at the current point in time. The Continuous Level Of Detail (CLOD) algorithm employed to visualise the terrain needs to operate directly on this submap. The paging algorithm employed for creation and removal of pages in the submap must be highly efficient since it is concerned with frequently paging large amounts of memory.

In conventional page management the software must first calculate which page holds the necessary data and then it must take the costly step of finding out where in memory the page is stored. This is usually done by searching a page look up table for the location of the desired page. A key advantage the offset spherical algorithm possesses over existing algorithms is the ability to quickly access data directly from inside a page without first having to determine where the page is stored in memory.

**EXAMPLE**

Our method is fundamentally different to existing page management techniques. In existing methods a user would remain centred in the submap with the pages changing. Here the user moves across the submap. For example, a user travelling in a straight line will move through different pages in the submap. When a user encounters the edge of the submap they will reappear on the other side of the submap. An example of a user moving to the right is shown in Figure 3. As the user moves from square 5 to square 6 new pages are loaded into 1, 4 and 7. The user can always see one page ahead. As the user moves off square 6 he/she moves onto square 4 which is displayed in front, i.e. the view has wrapped around to the other side of the submap. It should be noted that the user does not experience any discontinuity in viewing as he/she can at all times see one page ahead.

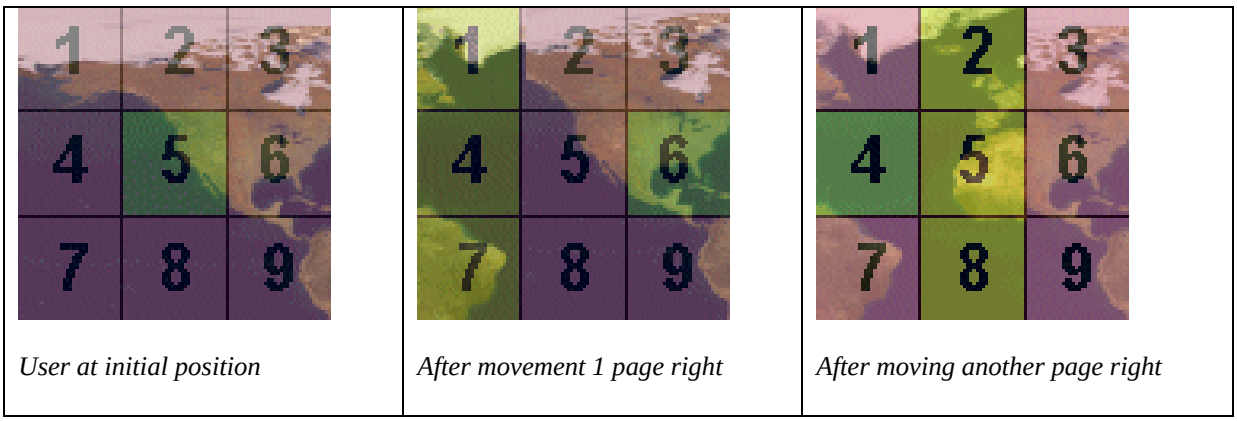


*Figure 3. : Example of page management with offset spherical algorithm*

The terrain pages shown in Figure 3 are typical of pages stored in an offset spherical buffer. The pages are a collection of tessellating terrain pages stored out of order i.e. their tessellating edges are not lined up.

**COMPARISON OF RESULTS**

As expected our simulator demonstrated execution speeds well below the execution speeds present in the other simulators examined. The Draken program, which used a lot of stored data, was able to execute twice as fast as our simulator. This kind of result however was expected and the objective of this work was only to reach a frames-per-second (FPS) suitable for real-time interaction. The 30 FPS obtained by this simulator is deemed to satisfy this goal. Figure 4 shows the results obtained.

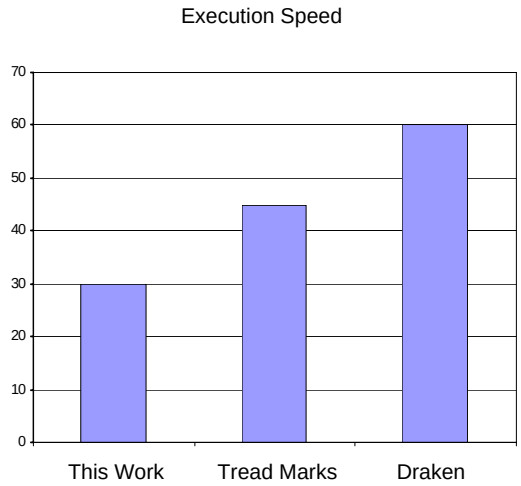


*Figure 4. Execution Speed Comparison*

Polygon density is an indication of visual quality of the final rendered images. This work was able to achieve a high polygon density without a map size trade-off. The program Tread Marks achieves a higher polygon density, but makes an enormous sacrifice in terms of size of defined landscape to do this. Figure 5 presents the comparison graphically.

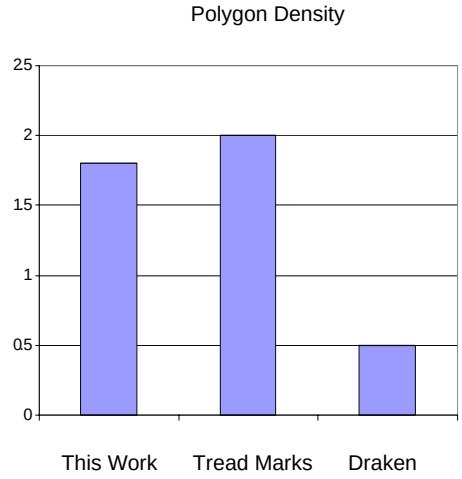


*Figure 5. Polygon Density Comparison*

This work gives a viewable distance (distance from user to vanishing point/horizon) par with industry standards. There is nothing inherent in the technology preventing an implementation from delivering a larger viewable distance. It should be noted however that any increase in viewable distance results in decreased execution speed. This is demonstrated by the fast execution speeds of Draken with a low viewable distance and Tread Marks with its large viewable distance having a more modest execution speed. Since our work incurs additional CPU overheads it is likely that implementations of this work will have less CPU resources available to use in other simulator tasks. This drain on CPU time makes the task of achieving extremely large viewable distances more difficult. Figure 6 shows the results obtained.

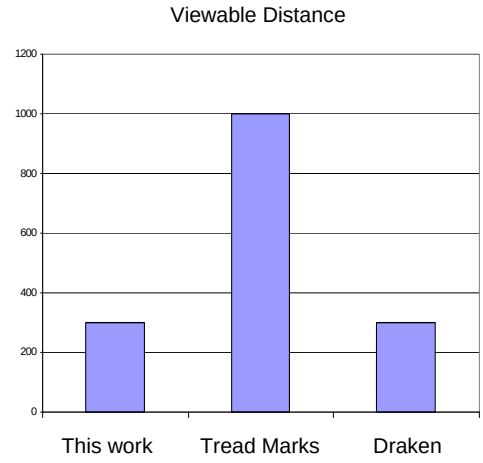


*Figure 6. Viewable Distance Comparison*

Having preformed modestly against other simulators so far in the comparisons we now examine the area in which this simulator was built to excel, map size. The simulator presented delivers a map size far beyond that presented by the other simulators investigated. Draken, already noted for its immensely large worlds, and large storage costs thereof, was able to produce under a millionth the landscape available in our system. Tread Marks, which obtains very high detail levels, had the smallest map present. The software compensated for this by having the map tessellate, resulting in the user being able to see the same hills repeated indefinitely. Figure 7 shows the results obtained. Note the graphs y-axis is scaled logarithmically to allow for easy interpretation.

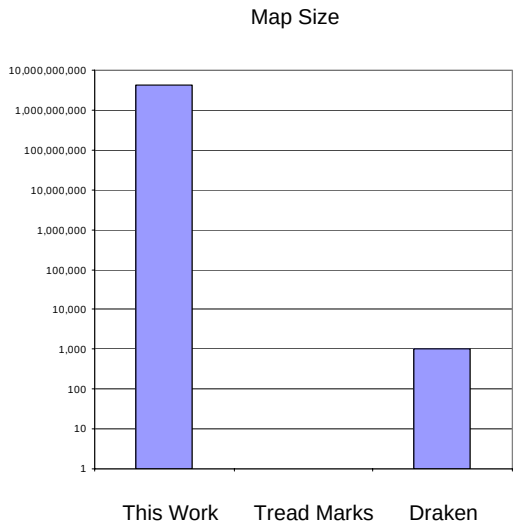


*Figure 7. Map Size Comparison*

## CONCLUSION

A method for generating terrain in real time has been presented. The techniques of procedurally generating terrain during a program execution may be slower than conventional techniques but offer an effective solution for providing larger and more sophisticated landscapes than is currently possible using non-generative techniques.

## REFERENCES

[1] Barnsley, M. F., Jacquin, A., Malassent, F., Reuter, L., & Sloan, A. D., Harnessing Chaos for Image Synthesis. Siggraph, 22(4), 131-140, 1988.

[2] Bitters, B. Terrian Data, 2000, Available: http://bbq.ncgia.ucsb.edu/education/curricula/cctp/units/unit06/06.html.

[3] Danaher, M. & Creemers, W. Real Time Terrain Simulation, Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems,(CIRAS), Singapore, 2003

[4] Denman, S., Patmore, A., & Ebling, T., Drakan, Order of the Flame (Version 1), Surreal Software, 1999

[5]. Eberly D.H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics, Morgan Kaufmann, 2001.

[6] Knuth, D. E., Semi-numerical Algorithms, 3rd ed., Vol. 2, Massachusetts: Addison-Wesley, 1997

[7] Mandelbrot, B. B. The Fractal Geometry of Nature, New York: W.H. Freeman and Company, 1977

[8] McNally, S., & McNally, J., Tread Marks, Version 1.0.1, Longbow Digital Arts., 2000.

[9] NIMA, National Imagery and Mapping Agency, 2000, Available: http://www.nima.mil/ .

## ABOUT THE AUTHORS

Warren Creemers, Department of Computer and Information Science, Edith Cowan University, Western Australia: w.creemers@ecu.edu.au.

Maurice Danaher, PhD, Department of Computer and Information Science, Edith Cowan University, Western Australia: m.danaher@ecu.edu.au.