Space Filling: A new algorithm for procedural creation of game assets

Paul Bourke
iVEC@UWA, The University of Western Australia, 35 Stirling Hwy, Crawley, Perth, West Australia 6009.
Email: paul.bourke@uwa.edu.au

John Shier
Normandale Community College, Bloomington, Minnesota 55431, USA.

***Abstract* --** Here we present a new algorithm that could be employed to procedurally generate a range of gaming assets including 2 dimensional textures, 2.5 dimensional texture, and 3 dimensional geometry. The approach involves placing shapes randomly, without overlap and with a monotonically decreasing area, within a region on a plane (2 dimensional texture) or within a volume (3 dimensional). If the process is continued to infinity then the result is space filling thus providing a variable and infinite degree of visual detail. It will be proposed/illustrated that the process is independent of the actual shape being used and as such can find a wide range of potential applications.

Keywords -- procedural, texture, tiling, packing, space filling.

I. INTRODUCTION

The algorithmic (procedural) generation of assets, texture and geometry, within virtual worlds and gaming environments is well established. Well known examples include various noise based techniques [1], terrain generation [2], clouds [3], and plants [4]. These examples and others often involve fractal processes, not only because many natural phenomena can be represented by fractals, but fractal processes allow the game engine to create geometric detail on demand and when or where required, for example, creating texture only in front of the player in a first person view. This ability to create variable levels of detail ensures distant objects or objects out of view can represented at a lower geometric detail. The detail generated can be increased as the viewer gets closer and can appreciate the higher resolution.

As an introduction to the algorithm we will present how it may be used to create textures, both image based and 2.5 dimensional surface roughness. There are some desirable properties such a procedural generator of textures should possess; the main ones are listed here

- The ability to create the texture at variable levels of detail. For example, distant objects do not need as resolved textures as close objects.
- The texture detail can increase smoothly as it is inspected more closely. Absence of this results in so called "popping" in algorithms where the additional detail is introduced at discrete distances and thus appears as an abrupt visual effect.
- Textures need to be able to tile the plane seamlessly. This allows a small texture unit to be used to cover a large region.

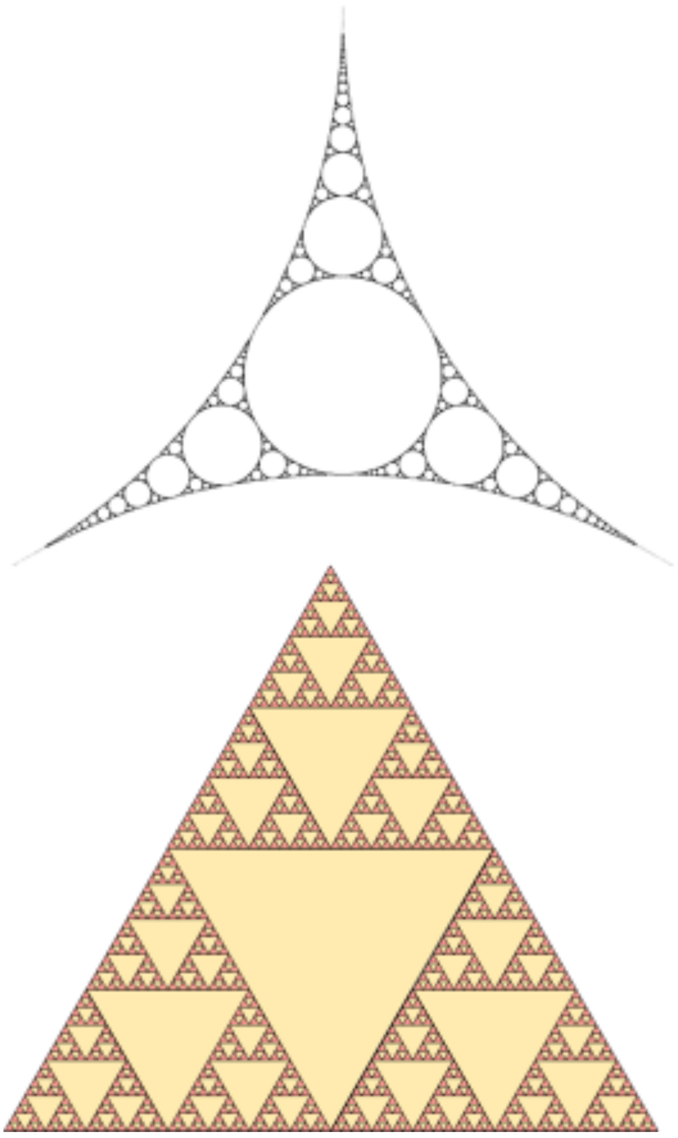

Figure 1. a) Appolonian packing. b) Sierpinski gasket.

The process outlined here provides a new answer to the question of how one might fill a bounded region with an infinite number of identical (this will be relaxed later) shapes so as to eventually fill the region, that is, fill in all the gaps [5]. Existing solutions include Apollonian [6,7] packing where shapes are iteratively introduced at random locations, they grow and perhaps move until they touch one or more existing shapes. Other solutions are recursive fractals such as the Sierpinski gasket [8] in which triangles fill the plane.

The solution outlined here sees shapes with a particular size added iteratively at a random location. If the shape does not overlap with any existing shape then it is placed permanently at that position otherwise a new random position is tried. The shape does not grow to fill the available gaps, and thus in general the shapes to not touch, rather the algorithm determines the size of each shape in advance. The question then is "*what is the monotonically decreasing function that determines the size of the added shape at the current iteration*". Clearly, if the size decreases too quickly then space filling will not be achieved. If the size decreases too slowly then the process will run out of space, there will be no gap large enough to add the next shape such that it doesn't overlap with the existing shapes.

II. ALGORITHM

If A_0 is the area of the first shape and $g(i)$ is the area scaling of the shape on each iteration i then the series of areas is simply

$$A_0, A_0\, g(1), A_0\, g(2), \ldots, A_0\, g(n), \ldots \tag{1}$$

The total area A is given by the sum of the above terms, namely

$$A = A_0 \sum_{i=1}^{\infty} g(i) \tag{2}$$

Therefore one is seeking a series $g(i)$ that is monotonically decreasing and sums to a constant, namely the area to be filled. If $g(i)$ decreases too fast then space filling is not achieved, if it doesn't decrease fast enough the procedure fails due to insufficient space for the next shape, see figure 2. While the discussion here concentrates on space filling, low values of $g(i)$ can be used as models for the distribution of water lilies on a pond or soap bubbles, see middle image in figure 2.

A series that satisfies the requirement and the one used here is as follows

$$g(i) = \frac{1}{i^c} \tag{3}$$

where c is some constant. This series is recognised as the Riemann Zeta function [9] which is known to converge for $c > 1$. For space filling one can choose a value of c, the sum of the series is used to determine the value of A_0 given the area A to be filled. Alternatively A_0 can be chosen which in turn determines the value of c.

The algorithm is as follows. Decide upon the bounded region to be filled and calculate the area A. Choose a value of c and based upon that and the sum of $g(i)$ calculate the area of the first shape A_0. On each iteration choose a random position from a uniform distribution for the current shape of area $A_0\, g(i)$. If the shape positioned at this candidate position does not result in any overlap with current shapes then place the shape at this position, otherwise keep trying other randomly selected positions until successful placement can be made.

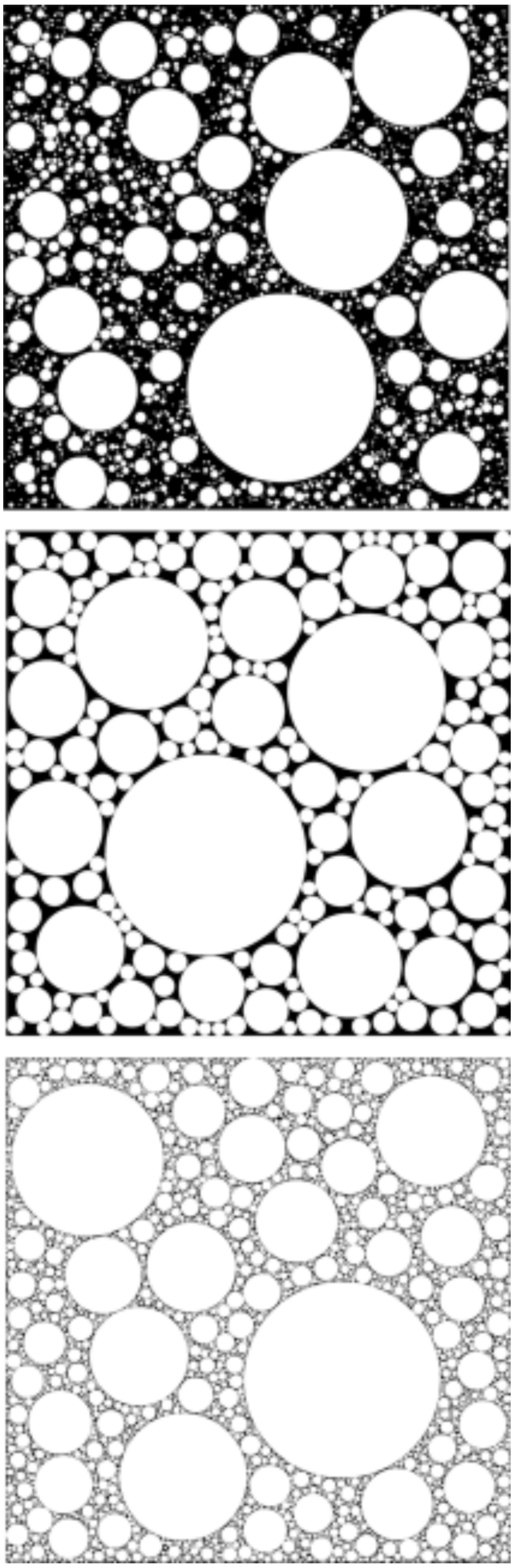
Figure 2. a) $g(i)$ decreases too fast [2000 circles], b) $g(i)$ decreases too slowly [no space after 200 circles], c) $g(i)$ decreases as presented to achieve space filling [5000 circles].

There are four requirements for an implementation that creates these space fillings:

1. Function that calculates the area of the shape given the parameters that define the shape.
2. Function that calculates the parameters of the shape given the area. This is essentially the inverse of the function in requirement (1).
3. Function that performs an intersection test between a shape and the boundary of the region being filled. This is used to determine whether the shape lies within the boundary and is also used to decide on issues of tiling.
4. Function that performs an intersection test between two shapes. This is performed on each proposed placement to ensure the shape to be added does intersect any existing shape, this is the computationally critical comparison since it is applied between the shape to be added at the current iteration and every existing shape.

III. PROPERTIES

The value of c controls the fractal dimension. For an embedding in dimension D (1,2,3) the fractal dimension d is given by

$$d = D / c \qquad (4)$$

Not any value of c may be chosen, in each dimension there is an upper limit and the limit depends on the shape being packed. For example in two dimensions (D=2) and for spheres the maximum value of c is 1.5.

If c and A_o are chosen as proposed then it is the authors claim that the process does not halt, that is, it does indeed allow an infinite space filling packing.

When used to create textures that tile seamlessly within a rectangular bounded region of width w and height h, the algorithm is modified such that when a placement is checked and made it is also performed at +-w and +-h. Figure 3 illustrates examples of tileable textures, bounded by a square but with toroidal boundary conditions. By comparison the examples in figure 2 were bounded within a square.

The derivation says nothing about the shape itself, only the area. As such the algorithm will work for any shape and indeed even for mixtures of shapes so long as the decreasing function of area on each iteration, g(i), is honoured. While this property is difficult to prove, it feels intuitively correct and the authors have not identified any shapes that cause the algorithm to fail . This includes highly convex shapes (upper example in figure 3) and shapes with holes (lower example in figure 3).

The placement of shapes is random in nature, as such the same random number seed will ensure the same sequence of shapes thus preserving level of detail continuity.

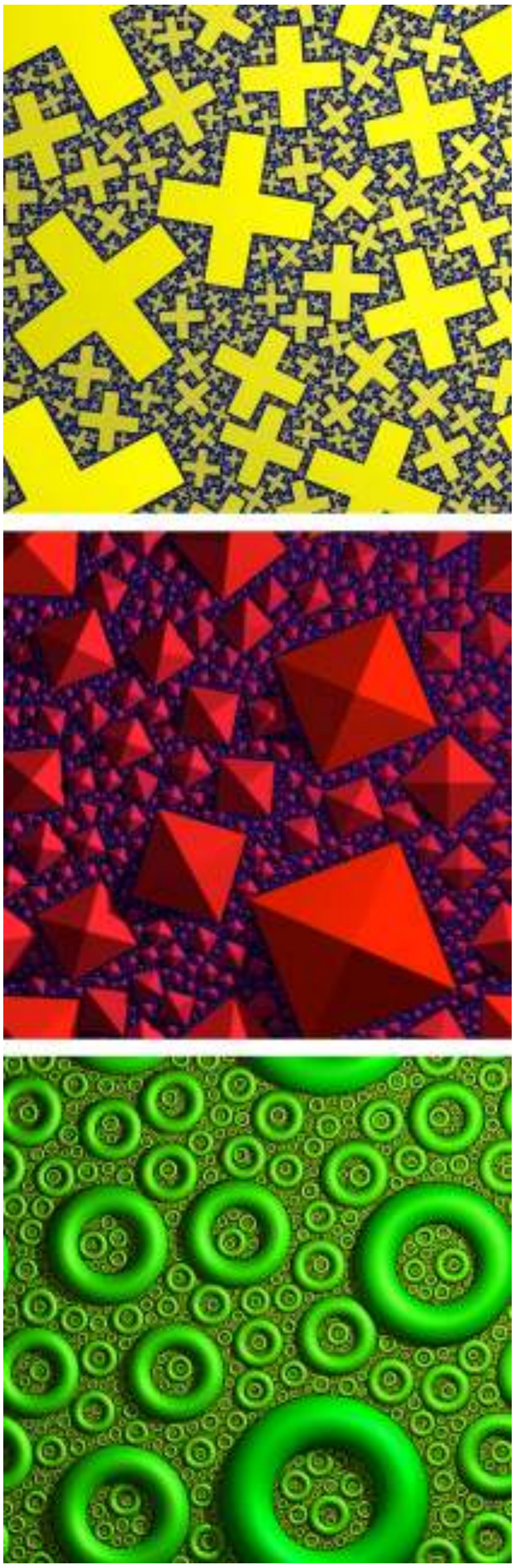
Figure 3. Examples of different shapes and toroidal boundary conditions Upper) Convex shapes. Middle) Randomly orientated pyramids. Lower) Shapes with holes.

Any irregularly bounded region can be textured, figure 5 shows an example of a toroidal region textured with smaller spheres. This capability only requires an intersection test between the shape being tiled and the boundary shape.

A minor modification to control the size of the first shape (and all subsequent shapes) is to modify the function of g(i) to

$$g(i) = \frac{1}{(i + N)^c} \tag{5}$$

Where N can be any positive real number but is usually an integer.

IV. OTHER DIMENSIONS

In the middle and lower example in figure 3 while there is a three dimensional nature to the shapes it is only the cross sectional area through the image plane that is used for the packing. This can be employed in the generation of random cityscapes [11] for example but the three dimensional nature is simply an extension (extrusion) into three dimensions and not a true three dimensional packing.

The method is however readily extended into other dimensions. In one dimension it transforms into packing line segments into a length L, see figure 4 for the case of a straight line but it could equally be some other curve. The function g(i) now defines how the length of each line segment is reduced on each iteration, so similar to equation 2.

$$L = L_0 \sum_{i=1}^{\infty} g(i) \tag{6}$$

In three dimensions g(i) governs how the volume of each shape reduces on each iteration. Figure 5 presents an example of tori filling a sphere but the algorithm can find application in the creation of three dimensional textures within a rectangular bounded volume. As discussed previously in two dimensions one may choose to fill a bounded volume or employ periodic tiling.

V. Conclusion

We have introduced a new algorithm with many of the features that are desirable when creating procedural multi-resolution two dimensional textures as well as three dimensional geometries. The procedural and iterative properties of this algorithm make it ideal for creating textures and geometry at sufficient detail to meet frame rate dictated performance constraints within a gaming engine dependent on the players distance and viewing direction. The algorithm allows for texture generation over a range of fractal dimensions and readily supports the ability to create tileable textures within rectangular regions.

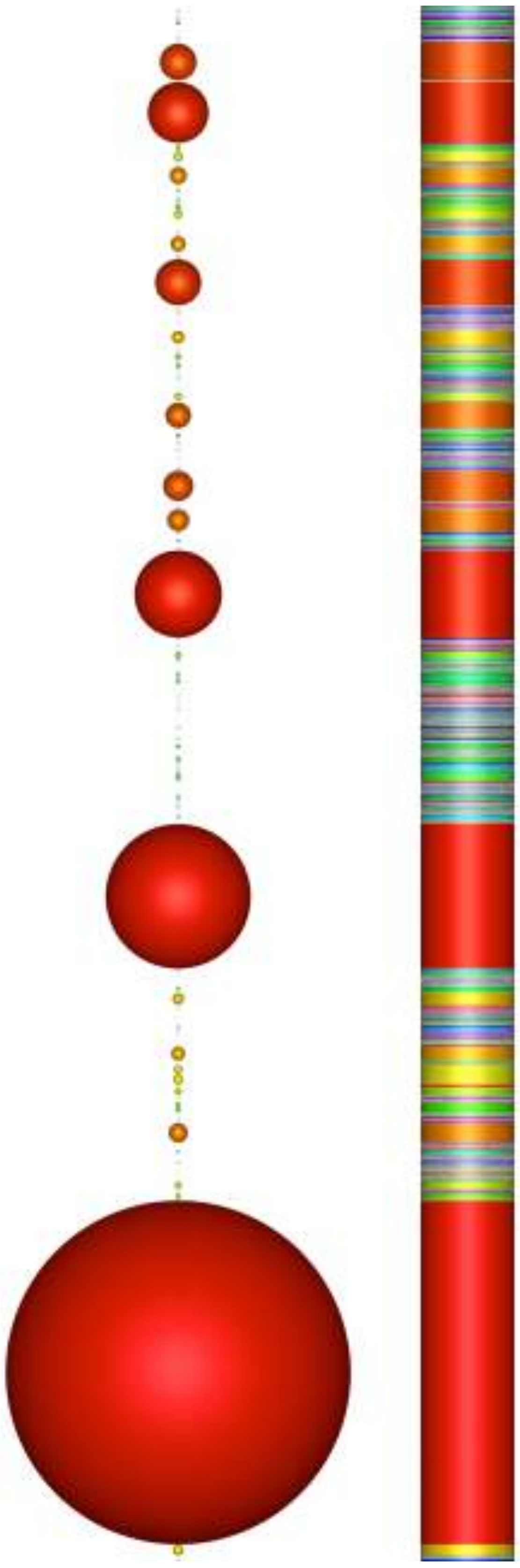

Figure 4. Example in one dimension. Presented as two representations, a sphere of diameter equal to the length of the line segment (left) and tubes (right).

ACKNOWLEDGEMENT

The algorithm presented has been inspired by John Shier [5] who conducted the pioneering research under the title of “statistical geometry”. The work was supported by iVEC through the use of advanced computing resources located at the University of Western Australia.

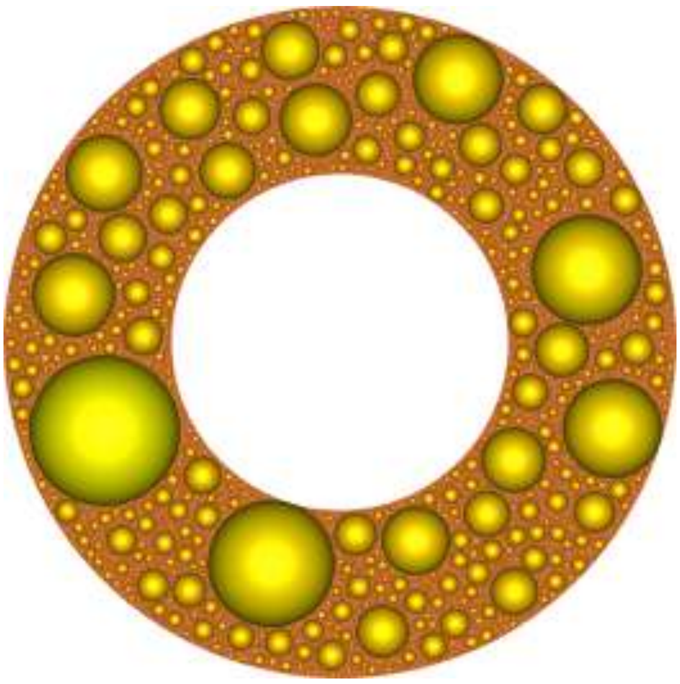
Figure 5. Example of non-rectangular bounded regions.

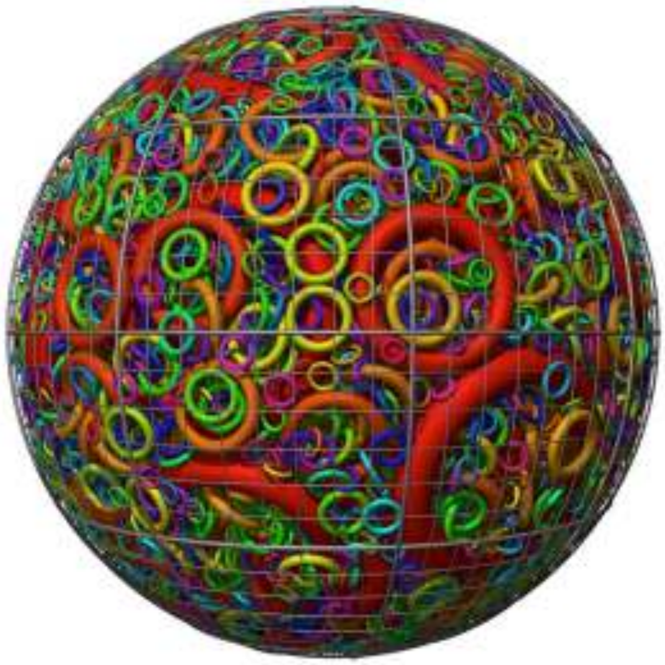
Figure 6. Example in 3 dimensions, tori filling a spherical region.

REFERENCES

[1] S. Green. nVidia Corporation. "Implementing Improved Perlin Noise". GPU Gems 2, Chapter 26,

[2] J.P. Lewis. "Generalized Stochastic Subdivision", Vol 6, 3 (1987)

[3] G.Y. Gardner. "Visual Simulation of Clouds". SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques. pp 297-304.

[4] P. Prusinkiewicz, A. Lindenmayer. "The Algorithmic Beauty of Plants". Springer-Verlag. (1990) pp 101–107. ISBN 978-0-387-97297-8.

[5] J Shier. "Filling Space with Random Fractal Non-Overlapping Simple Shapes". Hyperseeing summer 2011 issue, pp. 131-140, published by ISAMA (International Society of the Arts, Mathematics, and Architecture).

[6] P.D. Bourke. "Appolony fractal". Computers and Graphics, Vol 30, Issue 1, January 2006, pages 134-136.

[7] C.A. Pickover. "Cleopatra's Necklace and the Aesthetics of Oscillatory Growth". The Visual Computer, Vol 9, No 3 pp 166-169.

[8] I. Stewart. "Four Encounters with Sierpinski's Gasket". The Mathematical Intelligencer, 17, No. 1 (1995), 52-64.

[9] T.M. Apostol. "Zeta and Related Functions". NIST Handbook of Mathematical Functions, Cambridge University Press, ISBN 978-0521192255

[10] P.D. Bourke, J. Shier. The authors' web sites
http://paulbourke.net/randomtile (PB)
http://john-art.com (JS)

[11] S. Greuter, J. Parker, N. Stewart, G. Leach. "Real-time procedural generation of `pseudo infinite' cities". GRAPHITE '03 Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia.