Accelerated Artificial Landscape Visualisation

A. James and A.M. Day

School of Information Systems, University of East Anglia,
University Plain, Norwich, Norfolk, NR4 7TJ. England.
aj@sys.uea.ac.uk, amd@sys.uea.ac.uk

Abstract

We describe the design and implementation of four stages in an artificial landscape visualisation program. We begin by outlining a very simple but quick method of generating structurally realistic landscapes. We describe how the terrain can be stored in a multiresolution Binary Space Partitioning (MRBSP) tree such that each level holds a complete representation of the terrain at increasing resolution. A level of detail (LOD) technique is then illustrated which preserves the visually apparent size of triangles (after projection) so that the number rendered is reduced while preserving the aesthetic quality. Finally, we show how visibility calculations can be preprocessed and used to achieve output sensitivity by limiting the observer to ground based motion.

1. Landscape Generation

In this paper, we avoid 'mathematical' terrain generation [1, 2]. Instead, we use a very simple and *controllable* technique to generate quickly, realistic artificial landscapes with a regular triangular mesh by allowing a designer to 'draw' a desired landscape.

The difficulty associated with 3D interactive terrain creation is avoided since we allow the designer to draw the landscape using any standard *two-dimensional* art program. The image is then interpreted by preserving the x and y coordinates (representing longitude and latitude) and converting the intensities of the image pixels into z values (representing height) where black is sub-marine and white is a mountainous peak.

Hills and inclines can be generated by a background of subtle grey-scale shades such as those found with cloud-like images. The user may then add rivers, lakes and seas etc. by masking areas with black. Mountain peaks and ridges can be created by the addition of white lines or by lightening regions of the background to raise them from the land.

Further terrain manipulation can be achieved by tolerance inputs from the designer. Sea level adjustment allows the user to 'sink' or 'float' the land relative to the sea and height adjustment to determine the highest mountain peak. A seabed factor is used to increase or decrease the size of the beaches. Smoothing, to form *rolling* hills, is also controlled by the user but mountain ranges are exempt from increased smoothing so that jagged, natural peaks are preserved.

2. Multiresolution Binary Space Partitioning

Wiley *et al.* [3] describe their own modified BSP tree for multi-resolution models. We now present a similar technique which results in a two times increase in polygons at each level (instead of four by Wiley). This factor of two is required to preserve the visually apparent size of triangles after projection (described in section 3).

Each (non-root) node of our MRBSP tree contains a triangle in 3-space (right-angled in the xy-plane), and a vertical partitioning plane. The triangle represents a part of the surface which when combined with all others at the node's height in the tree forms an approximation of the terrain in equally sized triangles (in the xy-plane). The vertical partitioning plane divides the triangle into two halves by dividing it along the line from the centre of the hypotenuse to the opposite vertex and the sub-triangles are stored as the children.

The MRBSP tree creation follows the usual form

Published by the Eurographics Association, ISSN 1017-4656.

of a recursive function. A subset of the height points (obtained from the pixel intensities) is passed to the function and used to calculate the height of the corners of the triangle. We begin by passing all height points to the function and create a root node which comprises a *square* bounding the scene in x and y coordinates and which is coplanar with the sea level in the z (height) coordinate. The divider is aligned as described above.

Each child is passed one triangle after subdivision by the divider, and the subset of height points which is found on the relevant side of the divider. The coordinates of the triangle belonging to the node are known in the x and y coordinates since it is a regular subdivision of the parent, however their z (height) values are crucial to the surface definition. The height of each corner of the triangle is calculated by finding the three closest height points to the corner's x, y coordinate and then by interpolation to find the approximated vertex height.

3. Level of detail calculations

We have adopted a level of detail technique for tree traversal which preserves the *visually apparent* size of triangles, such that those in the distance *appear* the same size as those in the foreground.

Upon MRBSP tree traversal, we examine the triangle at each node. We find the distance, D, of the triangle's furthest vertex from the observer and then calculate $(M - log_2 \frac{D}{S})$; where M is the (maximum) depth of the tree, and S is the (short) side length of the smallest triangle (i.e. the size of the triangle at $depth M$). The resulting value (the *LOD threshold*) is the depth in the tree at which the triangle should be displayed to obtain its correct size.

An extra step is required to stop 'holes' appearing in the terrain. While continuity was explored in [4], our approach allows us to treat triangles *independently*. This is very advantageous since the BSP tree representation loses spatial neighbour relationships!

The problem is apparent in Figure 1 where some vertices are marked with their LOD thresholds. It has occurred due to a change in level of detail between two adjacent triangles – the culprit here is the triangle T. The problem occurs with triangles whose vertices fall within different LOD thresholds (between 12 and 13 in the example). With such triangles, the 'hole' scenario is only problematic if an edge is joined by a neighbouring vertex (of a higher resolution triangle) as described in [4] and illustrated in Figure 1 – this is identified when just one vertex has a threshold lower than the other two and that vertex is the right angle. (This is because we only subdivide triangles from a right angle vertex to the centre of the hypotenuse). When a potential hole exists, we render the triangle at the next level of detail to 'split' the triangle into two parts – this split connects the problematic 'kink' to its neighbour thus sealing the hole.

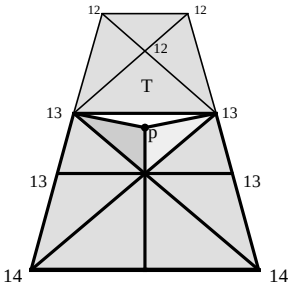

Figure 1: *A perspective projection of an LOD terrain with a continuity problem (vertices are numbered by LOD threshold)*

4. Visibility Calculations

Various techniques have been used to calculate terrain visibility at run-time [5, 6, 7, 8, 9], but by exploiting the progressive nature of the MRBSP tree, we can perform such calculations at the *preprocessing* stage. We achieve this by dividing the height-surface (1.8 metres above the landscape surface) into a set of rectangular cells and perform area-to-area visibility calculations for each cell. Thus at run-time, our current grid cell location can be used to provide us with an overestimated set of the (partially) visible triangles.

Using vertex-to-vertex sampling, visual anomalies are rarely apparent due to the coherence of a terrain surface (e.g. terrains do not exhibit characteristics such as portals). The size of each grid cell affects the preprocessing times and run-times – the smaller the grid cell, the longer the preprocessing time due visibility being calculated for more grid cells. However, the run-time is faster since the (overestimated) set is a closer approximation to the visible set. Preprocessing time has been accelerated since, for each grid cell we use surface triangles of a coarseness several times stronger than the finest terrain to act as 'blockers' and 'blockees'. Storage requirements are small and run-time overheads are insignificant since we only store cell-to-cell *changes* in visibility for each grid cell to it neighbours. At run-time, we make visibility changes by toggling the visibility flag of each node.

Figure 2 shows an example of visibility change in a *two-dimensional* scene (which can be seen as a cross section of a 3D landscape in the xz-plane). The landscape is approximated by line segments shown in black separated with ticks and numbered hierarchically according to their position in the tree, a coarse landscape

representation shown in light grey is used for visibility and is labelled likewise. Grid cells which permit visibility changes are indicated in dark grey and are labelled alphabetically.

Table 1 shows (part of) the file that would be associated with such a landscape (assuming that the observer is at height zero above the landscape). The first column represents the grid cell change and the second column the change in coarse landscape visibility. The third column (which would not be stored in the file) is the change in visibility of the fine segments and is calculated from the coarse representation at run-time. Thus, for example, as we progress from d to e, the segments 00 and 01 (and their subsegments) disappear and segments 10 and 11 (and their subsegments) appear.

Figure 2: *A 2D terrain illustrating visibility calculations*

Chng	Coarse chng	Fine chng
a-b	11	1100, 1101, 1110, 1111
d-e	00, 01, 10, 11	0000, 0001, 0010, 0011,...
g-h	00	0000, 0001, 0010, 0011

Table 1: *Cell-to-cell changes in visibility*

5. Results

Testing is performed with a 65,536 triangle landscape. While visibility (VIS) calculations are performed in a preprocessing stage, LOD calculations of 1/320s per frame can be achieved. Table 2 illustrates run-time performance for the methods used, the number of triangles rendered, overall rendering time (PPC 117Mhz, 16bit colours at 800x600), and the percentage of time compared to the brute force method.

Method	Tris	Time(s)	Time(%)
Brute	7,238	0.68	100.0
LOD	3,008	0.35	51.5
VIS	2,755	0.28	41.2
Both	1,732	0.18	26.5

Table 2: *Results using speed-up combinations*

While *both* LOD and visibility calculations are beneficial, two factors need to be highlighted. Firstly, as the landscape grows outwards, the LOD calculations become more beneficial since as distance increases, the coarser the representation can become. Secondly, as the land becomes rougher and more hills and mountains appear, the more beneficial visibility calculations become because the level of obscuration increases. Thus the efficiency of the techniques presented in this paper increase as the landscape becomes more complex.

References

1. A. R. Dixon and G. H. Kirby. A data structure for artificial terrain generation. *Computer Graphics Forum*, 13(1):37–48, 1994.
2. C. A. Pickover. Generating extraterrestrial terrain. *IEEE Computer Graphics and Applications*, 15(2):18–21, 1995.
3. C. Wiley, A. T. Campbell, S. Szygenda, D. S. Fussell, and F. W. Hudson. Multiresolution bsp trees applied to terrain, transparency, and general objects. *Graphics Interface*, pages 88–96, May 1997.
4. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time, continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH*, pages 109–118, August 1996.
5. C. Sansoni. Visual analysis: a new probabilistic technique to determine landscape visibility. *Computer-Aided Design*, 28(4):289–299, 1996.
6. L. De Floriani and P. Magillo. Horizon computation on a hierarchical triangulated terrain model. *The Visual Computer*, 11(3):134–149, February 1995.
7. D. Cohen-Or and A. Shaked. Visibility and deadzones in digital terrain maps. *Computer Graphics Forum*, 14(3):171–180, 1995.
8. L. C. C. Guedes, M. Gattass, and P. C. P. Carcalho. Real-time rendering of photo-textured terrain height fields. *Proceedings of SIBGRAPI*, pages 18–25, October 1997.
9. G. Nagy. Terrain visibility. *Computers & Graphics*, 18(6):763–773, January 1994.