REPRESENTING RANDOM TERRAIN ON RESOURCE LIMITED DEVICES

DAMIEN MARSHALL[ϕ], DECLAN DELANEY[ϕ], SEAMUS MCLOONE*, TOMAS WARD*

[ϕ]Department of Computer Science,
NUI Maynooth
Maynooth, Co. Kildare
E-mail: damienm@cs.may.ie

* Department of Electronic Engineering
NUI Maynooth,
Maynooth, Co. Kildare
E-mail: tomas.ward@eeng.may.ie

KEYWORDS
Random Terrain, Perlin Noise, Mobile Devices, Game Boy Advance

ABSTRACT

Random terrain generation is the procedural creation of a set of data that represents closely a believable landscape. Common techniques of achieving this include the use of fractals and noise. Such techniques usually require a large volume of memory, as the geometry of the terrain needs to be calculated and stored at run time. Given the limited memory available on mobile devices, such as mobile telephones, the storage of the data required to represent massive terrains can be difficult. In this paper, we propose a novel method of storing terrain data on devices with limited memory. This method involves placing pre-computed blocks of terrain, known as terrain tiles, together in a psuedo-random manner as governed by a noise function known as Perlin noise. This allows large amounts of terrain data to be represented while still giving the appearance of a randomly generated terrain. Traditionally, Perlin noise is used in the procedural generation of textures and the modeling of naturally occurring phenomena. Using Perlin noise, only a subset of the overall data generated by the function needs to be stored at any one time. The approach outlined in this paper associates a tile of terrain with each value generated by the Perlin Noise function, meaning that only a subsection of the total terrain is stored in memory at any one time. We show how this process can be executed in real time on a resource limited device known as the Game Boy Advance, and also illustrate a significant reduction in the memory requirements of terrain storage when compared with traditional methods.

INTRODUCTION

A random terrain is a group of procedurally calculated values that represents closely a believable landscape. They are utilised in all areas of computer science, especially computer graphics and games (Pickover 1995), where they allow for the creation of game content with minimal time overhead. Also, as terrain maps can be quite large, techniques of generating random terrain, such as fractals (Dudgeon and Gopalakrishnan 1996), and models of naturally occurring phenomena (Kelley et al. 1988), help to minimise fixed storage requirements. However, this means that such methods usually require a voluminous amount of Random Access Memory (RAM).

With recent advances in processing power, computer games are becoming more popular on handheld devices (Rittern et al. 2003). However, there are still limitations to the amount of available RAM for such devices, so that the creation of randomly generated terrain for computer games on such devices can be problematic.

In this paper, a novel method based on the use of terrain tiles and a pseudo random noise known as Perlin Noise is described (Rabinovich and Gotsman 1997; Perlin 2002) Traditionally, Perlin noise is used in the generation of procedural textures and the modeling of naturally occurring phenomena such as clouds and smoke (Ye and Lewis 1999; Holtkämper 2003) – see Figure 7. Here, we propose an approach that associates a tile of terrain with each Perlin noise value. A terrain tile describes a square block of terrain data. Using Perlin noise, only a subset of the overall data generated by the function needs to be stored at any one time. Therefore, large terrain maps can be represented in real time with a small memory footprint, as only the terrain tiles of interest to the user are stored at any given time.

The proposed method is implemented on a limited device known as a Game Boy Advance (www.nintendo.com), and results are presented for the implementation We show that significant saving can be made in terms of memory space required when compared to traditional methods of storing and representing terrain. As this method makes no use of floating point arithmetic it is very efficient, making it suitable for use on a wide variety of devices where no dedicated floating-point unit is available. It would be particularly suitable for mobile telephones, as it allows for the generation of massive terrain maps with a minimal over the air download. This is of significant importance as large downloads have been identified as a limiting factor in the next generation of three-dimensional games on mobile telephones (EDGE Magazine 2004).

The rest of the paper is laid out as follows. The next section introduces Perlin Noise and outlines the suitability of the Game Boy Advance as a testbed. The Implementation section details the realisation of the proposed method, followed by a selection of performance values in the Results section. Finally, the paper concludes with suggestions for future work.

BACKGROUND

Perlin Noise

Perlin noise is one of the more important noise functions (Perlin 2002). Created by Ken Perlin, this noise has the special property of appearing random to the perceiver, yet

remaining entirely controllable. It works by defining a number of key points at set distances apart, and defining the values of the intermediate points procedurally using interpolation. Any subsection of the data created by the Perlin noise function can be reconstructed, without the need to store and generate the entire data set. In this sense, Perlin noise can be considered to be quasi-random noise. Further detail regarding an implementation of Perlin Noise is given in the Implementation section.

Game Boy Advance

Released in 2001, the Game Boy Advance (GBA) is the successor to the multi-million selling Game Boy and Game Boy Color. The most important specifications of the Game Boy Advance are detailed in Table 1.

Processor	16Mhz
Memory	384 kilobytes
Screen Resolution	240 * 160 pixels
Available Colours	32,768

Table 1 Important specifications of the GBA

From Table 1, it can be seen that the GBA is very limited in terms of processing power and available Random Access Memory (RAM). Coupled with the ease of programming for the system, this makes the GBA a perfect platform on which to test our implementation.

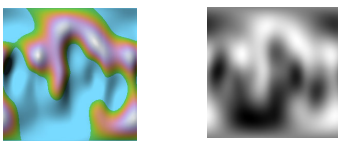
In the next section, the implementation details of the proposed method are described, and example screenshots of the application in action on the GBA are given.

IMPLEMENTATION

In this section, the method by which massive terrains can be constructed with a minimal memory footprint is discussed. The means by which the terrain data is rendered to screen at a stable frame rate on the GBA is also introduced.

Perlin Noise Map of Terrain Tiles

The basis of the technique described here is what is known as a terrain tile. A tile describes a square block of terrain. Terrain tiles are used extensively in many popular game engines such as Torque (Marshall et al. 2004). Each tile is typically defined by two groups of data – a height map, which details the geometry of the terrain, and a texture map, which details the colours and shading of the terrain (Rabinovich and Gotsman 1997). Figure 1 (a) and (b) details examples of both category of maps. Both images in Figure 1 were created using the Wilbur random terrain generation application (Slayton 2001).



(a) (b)

Figure 1 (a) Texture map (b) Grayscale Height Map.

By placing these tiles together in a random order, a large area of terrain can be defined – see Figure 2(a). However, as the tile size decreases, the storage cost of the order of the tiles can become expensive, and can eventually begin to approach that of the traditional method of storing each height of terrain in an array – see Figure 2 (b) and (c). This can make such an approach unsuitable for a limited memory device.

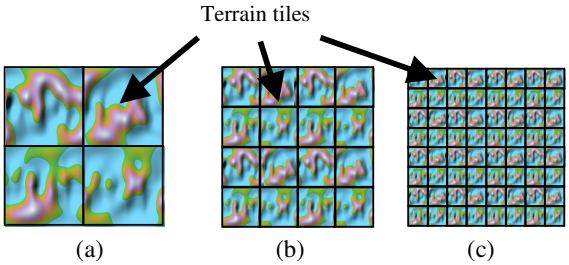


Figure 2 (a) Four 128*128 tiles (b) Eight 64 *64 tiles. (c) Sixty four 32 * 32 tiles. A decrease in tile size increases detail, but leads to an increase in memory requirements.

This problem can be alleviated using Perlin noise to govern the placement of tiles. As mentioned in the previous section, any subsection of the data generated by a Perlin noise function can be recreated given the same input values, without the need to store the complete data set generated by the function. Therefore, only the section of data of interest to the viewer needs to be calculated and stored at any one time, and it is guaranteed that other subsections of the data will be generated consistently with the same values.

The main tenet of the approach outlined in this paper is that by defining a number of tiles with small dimensions, and associating each value generated by the Perlin noise function with a terrain tile, then only the small portion of a massive terrain map needs to generated and stored at any one time, thus providing dramatic saving in terms of computation time and memory - see Figure 3. In Figure 3, each tile is represented by a different shade of gray. As the user moves about the large terrain map, the subset of terrain tile data is repopulated to represent the users immediate environment.

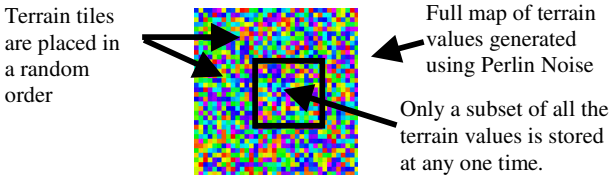


Figure 3 Only a small subset of all the tile values is stored at any one time using a Perlin noise function. This is repopulated as the user moves about the terrain.

Perlin Noise Implementation

The first aspect of the approach explained in this paper is the manner by which the Perlin noise map of terrain values is generated. The primary step of this process is the definition of a grid that defines the overall layout of the map. Each value in this grid is known as a terrain structure value. These terrain structure values are simply random numbers, and are not related to any particular terrain tile. Rather, they are used in conjunction with adjacent terrain structure values in order to define intermediate terrain tile values procedurally. Each

of these terrain tile values is a reference to a particular terrain tile. Although this grid of terrain structure values will have tiny dimensions, it can be thought of as being virtually large, as each element represents a point along a larger map – see Figure 4. So, for example, a 4 * 4 array, with a virtual distance of 1024 pixels between elements of the array, represents a 4096 * 4096 pixel terrain map.

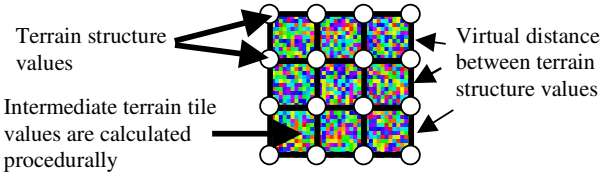


Figure 4 A grid of terrain structure values defines the overall layout of the terrain. Intermediate values are calculated procedurally.

Next, the terrain structure values are used to procedurally define the intermediate terrain tile references that describe the terrain immediately surrounding the user.

Firstly, the area within the grid of terrain structure values that currently houses the user is located, and the points that define the boundaries of that area are recorded. So, taking the example from above, if the player was at point (1056,3042) on a 4*4 map with a virtual distance of 1024 pixels between each element, then the player would be located between the terrain structure elements (1,2), (1,3), (2,2) and (2,3), as detailed in Figure 5 below.

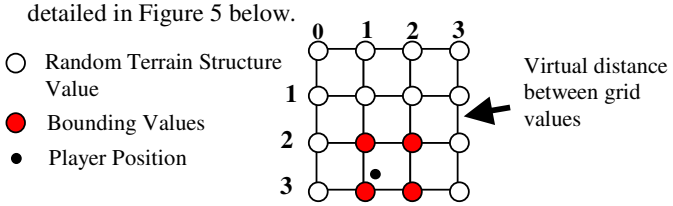


Figure 5 Location of a player's position within the terrain structure grid.

By linearly interpolating between the two top and the two bottom bounding values using the player's horizontal position, and interpolating between the results of these operations using the player's vertical position, a value which represents the tile in which the user resides can be defined – see Figure 6.

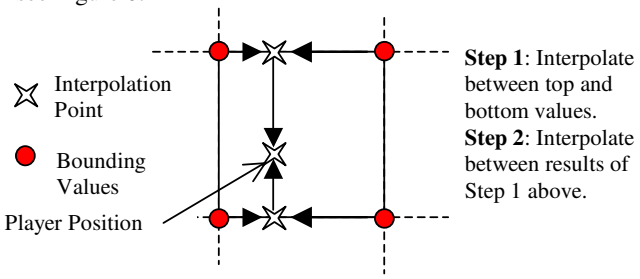


Figure 6 Resolving the terrain tile value based on the players position and the calculated bounding values.

Map of Terrain Tiles

Traditionally, the value generated by this process would be used in conjunction with other generated values in order to represent a texture or naturally occurring phenomena procedurally, as in Figure 7.

Figure 7 An example of a traditional output from the Perlin noise function

However, for this implementation, each value is actually a reference to a particular terrain tile. As there will be only a limited number of terrain tiles, and the generated value may exceed the total number of terrain tiles, the result may have to be scaled. This can be achieved using a simple modulus operation.

This process is repeated using the coordinates surrounding the player. The number of repetitions required will vary, depending on the size of each individual tile and the overall map size required. This results in a temporary array of data containing references to the terrain tiles that immediately surround the user. An example of such an array is given in Figure 8. For clarity, each separate map is represented by a different shade of gray. Each colour represents a map similar to that in Figure 1(a).

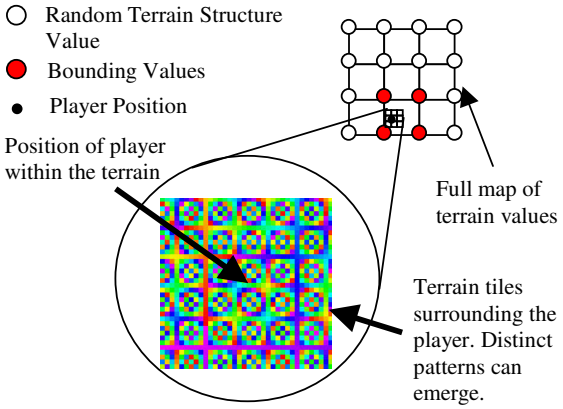


Figure 8 Only the terrain tiles that immediately surround the player are calculated. The player is located at the centre of these tiles.

However, as can be seen in Figure 8, through the use of this method alone, a series of distinct patterns can be observed in the placement of terrain tiles. Such patterns are exactly what Perlin noise is celebrated for. However, in the technique under discussion, more randomness is desired to generate truly random landscapes. This is achieved via the use of another array of random values, which is populated along with the terrain structure grid. The value acquired by the interpolation process is used to index into this random array, allowing for another layer of randomness, with little computation overhead. An example of the temporary array of terrain tile values produced using this method is given in Figure 9. Again, for clarity, each separate map is represented by a different shade of gray.

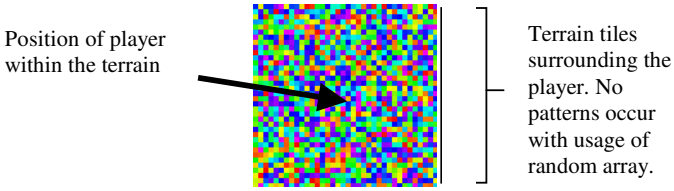


Figure 9 No discernable patterns appear when a random number look-up table is utilised

This temporary array represents only a small subsection of the overall terrain data. However, this fact is not noticeable to the user, as the temporary array of terrain tile references is repopulated when the player reaches a certain distance from the perimeter of the map. This array is then used as input to the rendering process.

Rendering Process

The rendering process uses a method known as ray-casting and voxels in order to draw the relevant terrain tiles to the screen in a timely fashion (Kreeger et al. 1998; Steinbach et al. 2000). Although this method has its limitations, such as only two planes of movement, it allows for a pseudo three-dimensional effect on resource-limited hardware such the GBA, at a stable frame-rate. Figure 10 details screenshots of the rendering application in action on the GBA. Superimposed on the images is a grid defining the boundaries of each terrain tile. Figure 10 also reinforces the relationship between terrain tiles, as seen in Figure 1 (a) and (b), and the temporary array of tile references as seen in Figure 8 and 9.

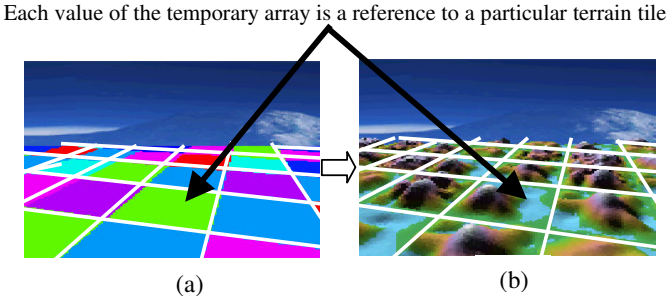


Figure 10 (a) The temporary array of tile values rendered to the screen. Each tile is represented by a separate colour. (b) The associated height and colour maps rendered to the screen.

RESULTS

Computation Time

As a preliminary exercise, the computation time required to calculate the temporary array of terrain tiles surrounding the player was recorded. This is achieved by using an in-built timer on the GBA that is incremented on every clock tick. A running average of the amount of time in milliseconds taken to populate the tile map is noted. In each case, the tile map required was 1024 * 1024 pixels. The tester navigated the environment for 60 seconds, and at the end of this time, the average computation time was recorded. The results of this experiment are presented in Table 2.

As can be seen from Table 2, the computation time required is minimal, and can be calculated in real time on the GBA. As no floating-point arithmetic is used in the calculation of the terrain tile values, this process is suitable for a wide range of devices where no hardware support for floating point values is available. To further demonstrate the efficiency of this approach, the computation time required for the creation of a completely new random map was also investigated. In each case, random maps were generated for 60 seconds, and a running average of the computation time was recorded using the in-built GBA timers. As this process simply involves the regeneration of the terrain structure grid, the computation time required is minimal, as detailed in Table 3.

Tile Size	Tiles required	Time
128 * 128 pixels	8 * 8 tiles	<1 ms
64 * 64 pixels	16 * 16 tiles	10 ms
32 * 32 pixels	32 * 32 tiles	67 ms
16 * 16 pixels	64 * 64 tiles	165 ms

Table 2 The computation time required for the generation of a 1024 * 1024 pixel map using terrain tiles of varying size.

Terrain Structure grid dimension	Time
4 * 4 grid elements	1. 25µs
8 * 8 grid elements	3.5µs
12 * 12 grid elements	6.625 µs
16 * 16 grid elements	16.5µs

Table 3 The computation time required for the generation of a new random map is minimal as the terrain structure grid size is insignificant.

Memory Usage

The most critical aspect of implementing the proposed technique in mobile devices is how it exploits the available memory. We therefore consider memory usage to be the most important metric by which to assess our proposed approach. This will be done by performing a comparative analysis with the traditional technique of storing a single large terrain geometry map and texture map.

In Table 4 the memory usage of a traditional terrain geometry map and texture map is analysed. Every element of each map is assumed to be a 16 bit unsigned integer.

Individual Map Size	RAM
256 * 256 pixels	256 kb
512 * 512 pixels	1024 kb
1024 * 1024 pixels	4096 kb
2048 * 2048 pixels	16384 kb

Table 4 Values representing cost of storing full maps in memory.

As can be seen from Table 4, the storage of a single randomly generated map utilises a large volume of memory resources. The generation of a small 256 * 256 pixel map requires approximately 66% of the entire 384 KB of RAM on a GBA. As the map dimensions double, the memory requirements quadruple.

Next, the storage cost of the approach detailed in this paper was analysed – see Table 5. Results are given for the cost of storing one hundred terrain tiles both in fixed storage and in RAM. Each tile consists of a height and texture map, and every element of a map is an unsigned 16-bit integer. The RAM columns display the cost of storing the temporary array of terrain tile references that is procedurally generated. The dimensions of this array vary depending on the individual terrain tile size. If, for example, a temporary map of dimensions 1024 * 1024 pixels is required using terrain tiles of dimensions 8 * 8 pixels, then a 128 * 128 temporary terrain tile reference array is required.

Terrain Tile Dimensions	Cost to store 100 Terrain Tiles	RAM 256 * 256 map	RAM 512 * 512 map	RAM 1024 * 1024 map	RAM 2048 * 2048 map
8 * 8 pixels	25 kb	2 kb	8 kb	32 kb	128 kb
16 * 16 pixels	100 kb	0.5 kb	2 kb	8 kb	32 kb
32 * 32 pixels	400 kb	0.125 kb	0.5 kb	2 kb	8 kb
64 * 64 pixels	1600 kb	0.03125 kb	0.125 kb	0.5 kb	2 kb
128 * 128 pixels	6400 kb	0.0078125 kb	0.03125 kb	0.125 kb	0.5 kb

Table 5 Values representing memory cost of storing the terrain tiles and the cost of storing temporary tile value array in Random Access Memory (RAM).

Table 5 shows that the amount of RAM employed by the proposed approach is minimal. If the individual map dimensions are increased in size by 100%, the amount of RAM required reduces by 75% whereas the fixed storage requirements increase by 400%. However, the cost of storing the terrain tiles is nominal at smaller sizes, thus making them suitable for download over low bandwidth connections, such as on a mobile phone network.

Regardless of the dimensions of the overall terrain map required, the figures presented in the RAM columns of Table 5 remain constant. This is due to the fact that, as discussed in the Implementation section, only a subset of the overall terrain map is ever stored, regardless of variation in the dimensions of the overall terrain map. Therefore, any size map could be potentially represented on a resource-limited device such as the Game Boy Advance, with no implications for the overall memory requirements.

CONCLUSIONS

In this paper we have described a method of representing the large volumes of data required to store a random terrain on a device with a limited amount of Random Access Memory known as the Game Boy Advance. By using terrain tiles and Perlin Noise, we have shown how this technique is particularly suitable to devices with limited processing power, as it makes no use of floating point arithmetic. Therefore, the terrain tile positions can be calculated easily at run time. In addition, a totally new random terrain can be generated rapidly, as only the small terrain structure grid needs to be repopulated.

It has also been shown that a massive terrain can be represented using a minimal amount of both fixed and dynamic memory. The cost required to store the necessary terrain tiles can be small, thus making this approach suitable for devices with limited storage capacity, and slow download speeds such as mobile telephones. As only a portion of the overall terrain is stored at any one time, the volume of memory used is fixed, regardless of variation in the dimensions of the overall terrain.

Future work will involve investigating methods of seamless blending between adjacent terrain tiles, so as to provide a more uniform visual experience to the user.

ACKNOWLEDGEMENT

This work was funded by Enterprise Ireland Basic Research Grant SC/2002/129/.

REFERENCES

Dudgeon, J. E. and R. Gopalakrishnan (1996). "Fractal-based modeling of 3D terrain surfaces".In *Bringing Together Education, Science and Technology*, (Tampa, Florida, USA, 11-14 April 1996), IEEE, 246 - 252.

EDGE Magazine (2004). "Mobiles prepare for 3D revolution."In *EDGE Magazine Issue 135, April 2004*: 7-9.

Holtkämper, T. (2003). "Real-time gaseous phenomena: a phenomenological approach to interactive smoke and steam".In *Computer graphics, virtual reality, visualisation and interaction in Africa*, (Cape Town, South Africa, February 03 - 05), ACM Press, New York, NY, USA, 25 - 30.

Kelley, A. D., M. A. Casey and G. M. Nielson (1988). "Terrain simulation using a model of stream erosion".In *15th Annual Conference on Computer graphics and interactive techniques*1988), ACM Press New York, NY, USA, 263 - 268.

Kreeger, K., I. Bitter, F. Dachille, B. Chen and A. Kaufman (1998). "Adaptive perspective ray casting".In *1998 IEEE symposium on Volume visualization*, (Research Triangle Park, North Carolina, United States, ACM Press, New York, NY, USA, 55 - 62.

Marshall, D., A. McCoy, D. Delaney, S. McLoone and T. Ward (2004). "A Realistic Distributed Interactive Application Testbed for Static and Dynamic Entity State Data Acquisition".In *Irish Systems and Signals Conference*, (Belfast, Ireland, 30 June - 2 July), IEE, 83-88.

Perlin, K. (2002). "Improving Noise".In *29th annual conference on Computer graphics and interactive techniques*, (San Antonio, Texas, July 23 - 26), ACM Press New York, NY, USA, 681 - 682.

Pickover, C. A. (1995). "Generating extraterrestrial terrain."In *Computer Graphics and Applications, IEEE Issue 2, March 1995*: 18 - 21.

Rabinovich, B. and C. Gotsman (1997). "Visualization of large terrains in resource-limited computing environments".In *8th conference on Visualization '97*, (Phoenix, Arizona, United States, 19-24 October), IEEE Computer Society Press, Los Alamitos, CA, USA, 95 - 102.

Rittern, H., T. Voigt, M. Tian and J. Schiller (2003). "Experiences using a dual wireless technology infrastructure to support ad-hoc multiplayer games".In *Proceedings of the 2nd workshop on Network and system support for games*, (Redwood City, California, May 22 - 23), ACM Press, New York, NY, USA, 101 - 105.

Slayton, J. (2001). Wilbur Terrain Generator http://www.ridgecrest.ca.us/~jslayton/index.html.

Steinbach, E., B. Girod, P. Eisert and A. Betz (2000). "3-D reconstruction of real-world objects using extended voxels".In *2000 International Conference on Image Processing*, (Vancouver, BC Canada, 10-13 September 2000), 569 - 572.

Ye, A. G. and D. M. Lewis (1999). "Procedural texture mapping on FPGAs".In *1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, (Monterey, California, United States, February 21 - 23), ACM Press, New York, NY, USA, 112 - 120.