

# A Voxel-based Octree Construction Approach for Procedural Cave Generation

Juncheng Cui<sup>†</sup>, Yang-Wai Chow<sup>†</sup> and Minjie Zhang<sup>††</sup>,

School of Computer Science and Software Engineering, University of Wollongong, Australia

Advanced Multimedia Research Lab<sup>†</sup>

Intelligent Systems Research Lab<sup>††</sup>

## Summary

Procedural content generation is becoming an increasingly popular research area as a means of algorithmically generating scene content for virtual environments. The automated generation of such content avoids the manual labour typically associated with creating scene content, and is extremely useful in application areas such as computer graphics, movie production and video games. While virtual 3D caves are commonly featured in these virtual environment applications, procedural cave generation is not an area that has received much attention among researchers to date. This paper presents a procedural approach to generating 3D cave structures. Other than the development of a method to effectively automate the generation of visually believable 3D cave models, this paper also investigates how to efficiently construct and store this spatial information using a voxel-based octree data structure. In addition, the proposed approach demonstrates that caves with different characteristics can be generated by adjusting certain parameters in order to facilitate the creation of diverse cave structures.

## Key words:

*Procedural Content Generation, Caves, Voxel, Octree.*

## 1. Introduction

Procedural content generation, or procedural modelling, has received much attention especially in areas like computer graphics, virtual environments, movie production and video games, as a method of algorithmically generating scene content for applications in these areas. The growth, in terms of size, detail and visual realism, of the synthetic 3D virtual environments represented in these application areas, means a constant demand for increasing the amount of content to fill up these virtual worlds [1]. This increase in visual complexity and detail makes manual content creation extremely laborious and time consuming, thus, potentially increasing the production time and costs required to create these virtual worlds.

This makes content creation via procedural techniques particularly attractive, as it allows for the automated

generation of scene content by a computer. The widespread development of procedural techniques over the last decade has been due in part to the increase in computational power, which has provided researchers with the opportunity to explore methods previously infeasible [2]. To date, researchers and practitioners have developed procedural content generation techniques in a variety of areas, for example, in terrain and landscape generation [3, 4], plant and tree models [5], buildings [6], cities [7], roads [8], etc. In addition, procedural techniques are particularly suitable for creating the randomness present in naturally occurring phenomena like fire [9], water and waves [10], and smoke [11], which have additional animation characteristics.

Virtual 3D caves often appear in movies and video games [12]. However, despite being commonly featured in movie and video game environments, the procedural creation of caves has remained relatively unexplored [13, 14]. This paper investigates the use of procedural techniques to synthetically generate 3D cave structures. The usefulness of procedurally generating 3D caves is not solely limited to the entertainment industry, since virtual 3D cave environments can also be used in the development of training simulations for cave explorers, and possibly for training search-and-rescue personnel [13].

Terrain generation is a particularly successful area in procedural content generation, and is very much related to cave generation. The success of procedural terrain generation has led to the development of a variety of physically-based generation techniques for the synthesis of virtual terrains which display advanced visual features like hydraulic erosion, etc. [3, 15]. The purpose of the approach present in this paper is not to simulate physically-based cave formations, but rather to generate visually believable 3D cave structures.

The task of procedurally generating 3D cave models presents a number of challenges. In particular, this involves developing a method of effectively automating

the generation of the 3D structure, as well as efficient storage and rendering. Terrain generation techniques typically use the 2D height map approach for storing height field information, due to its simplicity and efficient memory storage requirements. However, height maps have the limitation of only being able to represent a single height value for each position on the horizontal plane, and as such cannot be used to represent features like overhangs, arches or caves [3, 4].

The method adopted in this paper is to use a voxel-based approach, as voxels (volumetric elements) have the capability of representing 3D spatial data. One of the drawbacks of using voxels is its large memory storage requirements. Therefore, an octree data structure will be used to store the voxel data. A 3D noise function will be used to procedurally construct the voxel-based 3D cave model. However, using a noise function by itself gives rise to the problem of 'floating islands' and 'air pockets' in the resulting structure. Thus, this paper also presents a method to process and remove these problem areas, along with the approach used for constructing the data structure. Furthermore, this work shows that the appearance of the generated cave structure can be varied by simply adjusting certain parameters of the noise function.

The rest of the paper is organised as follows. Section 2 describes related work in 3D cave models and octree construction techniques. A detailed description of our approach is given in section 3. Experimental results are demonstrated and discussed in section 4. Finally, the paper is concluded and future work is outlined in section 5.

## 2. Related Work

### 2.1 3D Cave Models

Existing efforts in the construction and visualisation of 3D cave structures include the use of scanning hardware to obtain accurate spatial data about actual cave structures [16]. The scanned spatial data can then be used to reconstruct a virtual representation of the real cave that can be visualised on a computer display. However, the 3D mapping of caves using this physical approach is an extremely painstaking and time consuming process. Schuchardt and Bowman [17] investigated whether the visualisation of complex 3D cave structures using immersive virtual reality provided a higher level of spatial understanding of such structures, which cannot be mentally visualised using traditional means such as 2D cave maps. The cave model used for their system was constructed from cave survey and measurement data

obtained from an actual cave, and converting this information into a 3D cave model.

The procedural creation of synthetic 3D cave models has previously been investigated by Boggus and Crawfis [13, 14, 18]. Their work focuses on the procedural generation of solution caves, which are caves formed by rock being dissolved by acidic water. In their research, they apply knowledge about the formation of solution caves in order to create cave models for virtual environments. Their proposed method involved approximating water transport to create a coarse level of detail model for a cave passage. They also demonstrated methods of generating 3D cave models using cave patterns, and proposed that surface detail could be added using techniques like bump mapping and displacement mapping.

Johnson, et al. [19] examined an approach of using a cellular automata-based algorithm for the real-time generation of 2D infinite cave maps, for the purposes of representing cave levels in video games. However, the generation of 3D caves maps using this approach was left for future work. Peytavie, et al. [4] presented a framework for representing complex terrains, which includes caves, using a volumetric discrete data-structure. In addition, they proposed a procedural rock generation technique to automatically generate complex rocky scenes with piles of rocks. Their aim was to generate and display physically plausible scenes without the computational demand of physically-based simulations. Their approach mainly focused on using their unique data-structure for efficient interactive sculpting, editing and reconstruction using high level terrain authoring tools, as opposed to a purely procedurally driven approach.

### 2.2 Octrees

Voxels have traditionally been used to store volumetric data for applications like MRI scans [22]. In general, the use of voxels consumes huge amounts of memory when used for truly volumetric 3D data. However, Laine and Karras [22] have shown that memory usage drops significantly when used to encode surfaces in sparse voxel octrees. In their work, they analysed the efficient use of sparse voxel octrees in conjunction with GPUs for ray casting.

Octrees, like quadrees, are hierarchical data structures based on the decomposition of space [23]. To date, much research has been done on efficient construction and traversal of octrees. Octrees have been used for a variety of purposes including for accelerating rendering, for collision detection and for operations such as locating

neighbours. The typical approach to building an octree is to take a top-down approach, where one starts with a volume that encompasses the entire scene and recursively subdivides space into octants until a certain maximum depth or termination criterion is reached. Some researchers have also proposed methods for bottom-up octree construction. However, most methods of building octrees rely on knowledge of the scene to be known *a priori*, before subdivision can take place. This paper presents an approach to building an octree from procedural data that is not known *a priori*, and nodes are created and inserted from bottom-up as required.

### 3. Our Approach

#### 3.1 Generating the Cave Structure

The fundamental idea behind this approach to generating the cave structure is to be able to separate air from stone in a given 3D spatial area. The 3D area in question will form the foundation of the cave structure as it will provide the overall cave shape. For simplicity, basic volumetric 3D shapes like spheres, ellipses, cuboids, cylinders, cones, etc. can be used as the cave foundation. These basic shapes can easily be connected and composited to form entire cave systems, with passages, crevices, caverns, etc. Voxels are used to store the 3D spatial data, with each voxel representing an area of 3D space. Thus, for each voxel position in the given 3D area, a binary operation needs to be defined that will return the value of 1 for air and 0 for stone.

To produce the randomness which is a key feature of cave structures, a 3D noise function will be used to create the overall distribution of air and stone. A 3D Perlin noise function was employed for this work. Note that other 3D noise functions can also be used which will result in different distributions and may potentially give rise to different cave structures. Nonetheless, this was left as a topic for future work. Perlin noise is a type of gradient noise that gives a pseudo-random appearance and has the property of generating a smooth continuous noise distribution [20]. In addition, a bias function proposed by Perlin and Heffort [21], was used in conjunction with 3D Perlin noise to control and facilitate the smooth separation between air and stone at the walls of the cave. Thus, different cave structures result from adjusting the bias parameter,  $b$ . The bias was defined as the following power function [21]:

$$f^{\frac{\ln(b)}{\ln(0.5)}} \quad (1)$$

Figure 1 gives a 2D cross-section depiction of the overall process described above, note that the actual approach is in 3D. Figure 1a shows an example of a basic shape (a sphere in this case) used as the cave foundation, with black representing stone and white representing air. A depiction of Perlin noise is given in Figure 1b. Figure 1c shows the result of applying the bias function (with  $b = 0.05$ ), to the cave foundation (Figure 1a) in order to smoothen the separation between stone and air. Finally, Figure 1d presents the cave structure obtained after the combination of Perlin noise (Figure 1b) with the biased cave foundation (Figure 1c). This was done by comparing the Perlin noise value with the biased cave foundation value for each voxel. If the Perlin noise value was less than the biased cave foundation value for a particular voxel position, that voxel would be assigned to contain stone, otherwise it would contain air. One can see small ‘floating islands’ in the resulting cave structure in Figure 1d, which will have to be removed. Additionally, there are also small ‘air pockets’ in the stone which are redundant to the final cave’s display and should therefore be removed.

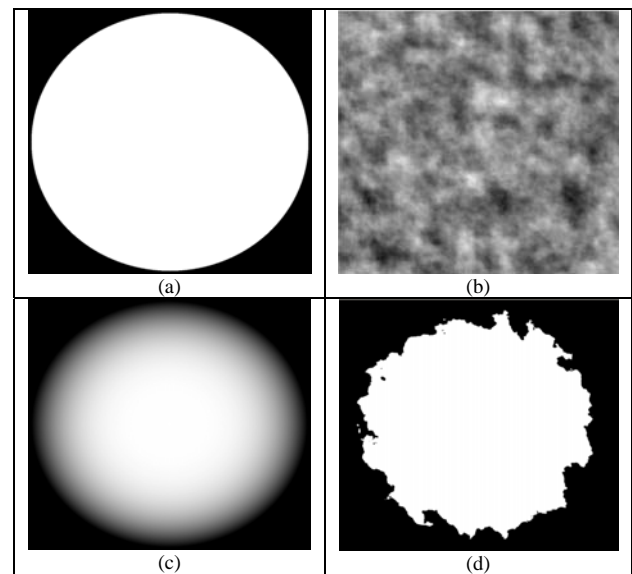


Fig. 1 2D cross-section depiction of the process used to form the cave structure. (a) Cave foundation; (b) Perlin noise; (c) Cave foundation after bias function, with  $b = 0.05$ ; (d) Cave structure resulting from the combination of (b) and (c).

#### 3.2 Building the Voxel-based Octree

While it is possible to first generate the cave structure and store this in a 3D voxel lattice, then utilise this data to build an octree, this would mean that the voxels would initially have to be stored in memory. A better approach would be to bypass storing the 3D voxel lattice and to

construct the voxel octree by inserting nodes into the octree on the fly as required. Therefore, a specialised bottom-up octree construction approach was designed for this purpose.

A quadtree will be used to describe the approach here as it will simplify the explanation; nonetheless, extending this to an octree is trivial. All nodes are assigned a unique numerical index based on the node's depth and its child number. The number of digits in a node's index is equal to its depth, and each digit represents the child node's number. Since a parent node in a quadtree can have a maximum of 4 children, the child node's number ranges from 0 to 3, whereas for an octree this would be from 0 to 7. Figure 2 (a) to (c) illustrate an example of the node indices at different depths for a quadtree. From the figure, it can be seen that the child nodes are number in a 'Z' order (i.e. top left = 0, top right = 1, bottom left = 2 and bottom right = 3).

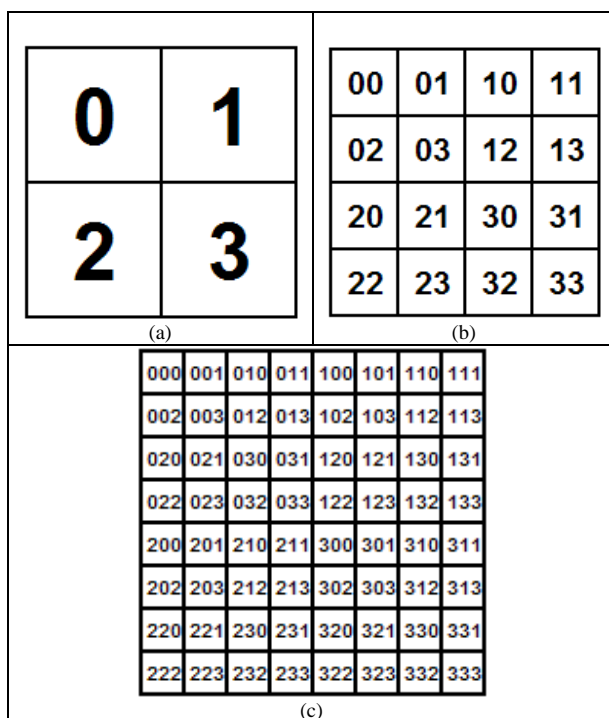


Fig. 2 Unique node indices for quadtree. (a) Depth of 1; (b) Depth of 2; (c) Depth of 3.

The tree construction approach is described as a bottom-up approach as the leaves of the tree data structure are inserted one after another. As such, the maximum tree depth has to be defined from the start. Voxels form the leaves of the tree and encompass areas of either stone or air. Parent nodes for the child nodes are created whenever

necessary. To check what material a voxel contains, the procedural cave structure generation method previously described in section 3.1 is used. If the voxel contains stone and is not already in the tree, it is created and inserted into the tree. Once the tree is fully built, all remaining leaf nodes are created and assigned as voxels containing air.

The node insertion algorithm follows a scan-line fill algorithm. The scan-line fill algorithm is commonly used in computer graphics for filling connected pixels in a polygon with a colour along scan-lines. In this case, the algorithm is used to insert connected voxels containing stone material into the tree. After a node is inserted, a test is performed to check whether the sibling nodes all contain the same material (i.e. stone). If so, a merge operation is performed where the parent node is assigned that material and the child nodes are removed from the tree. This reduces memory storage requirements. Figure 3 gives an example which illustrates the overall process.

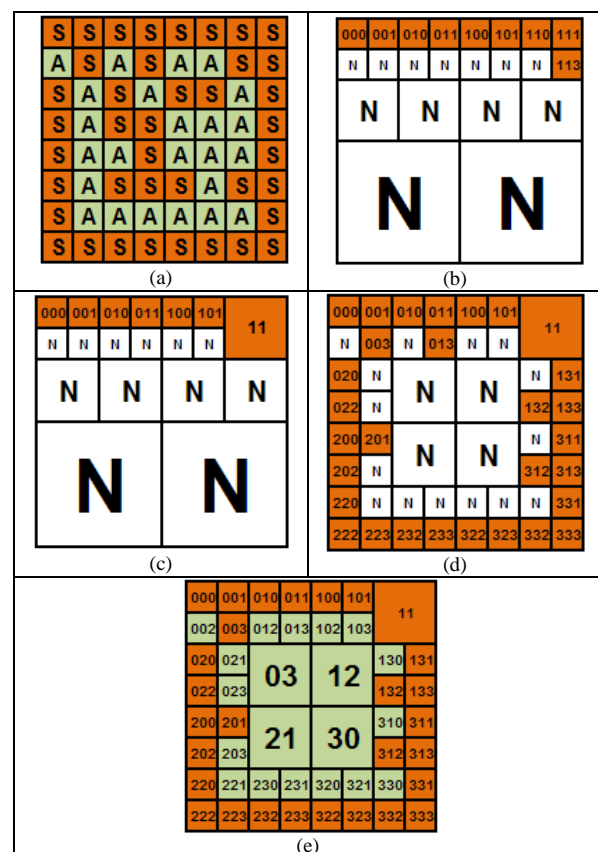


Fig. 3 Example of building the quadtree. (a) Actual composition of air and stone for the cave structure; (b) Insertion along scan-lines; (c) Example of a merge operation; (d) All connected stone voxels inserted; (e) Material of remaining leaves assigned as air.

Figure 3a shows the composition of stone and air that would be obtained from the procedural cave structure generation algorithm, 'S' represents stone and 'A' represents air. Figure 3b depicts quadtree state after the first scan-line and moving to the second scan-line, 'N' represents null (i.e. nodes not yet created). In Figure 3c, a merge operation was just performed and the children for that node were deleted. Figure 3d shows the state of the quadtree after insertion of all connected stone voxels. In Figure 3e, the material for all the remaining leaf nodes are assigned as air. In comparing Figure 3a with 3e, one can see that the stones in the middle that are not connected to the walls of the cave have been removed. This quadtree example can be seen as a slice or cross-section of an octree. To build the octree with the cave structure, simply repeat the process for all the slices. It can be seen that all unconnected 'floating islands' are ignored when building the tree.

### 3.3 Determining the Cave Boundary Surfaces

This next stage deals with the removal of small 'air pockets' from the stone (previously highlighted in Figure 1d, and represented in Figure 3e as node 002). At the same time, the boundary surfaces that separate stone from air (i.e. the internal walls of the 3D cave) are determined. The approach adopted is based on the flood-fill algorithm (also known as seed-fill), traditionally used in computer graphics to fill connected neighbouring pixels with the same colour.

The important property of the flood-fill algorithm exploited in this approach is that it can be used to find neighbouring voxels in 3D. However, when used in the context of voxel-based octrees, a complication arises when neighbouring voxels are of different sizes and at different depths in the octree. In view of the fact that all nodes were assigned a unique index based on their depth and child number (refer to Figure 2), the problem can be overcome by referring to the neighbouring voxels' indices. Figure 4 depicts how this was done in a quadtree scenario. Figure 4a shows the initial seed voxel, 'S'. In Figure 4b, neighbouring nodes 'N1' to 'N4', of the same depth are found. If a neighbouring node has children, the child nodes that are adjacent to the seed node should be considered instead. This is depicted in Figure 4c, where the children of 'N3' and 'N4' are considered, and the child nodes adjacent to the seed node are selected. Figure 4d shows all the connected neighbours that were found.

The initial seed voxel must be a voxel containing air that is located somewhere inside the cave. The algorithm will recursively search for neighbouring air voxels until it hits

a stone voxel. That means that on completion of the flood-fill process, any 'air pockets' in the stone will be removed as only the boundary voxels that separate air from stone will remain in the octree. Figure 5a illustrates the quadtree after the flood-fill process. From this, the cave boundary surfaces can be determined by selecting the smaller of adjacent stone-air voxels (i.e. nodes with larger depth), as depicted in Figure 5b. Then the polygonal surfaces of the cave wall can be computed, as shown in Figure 5c.

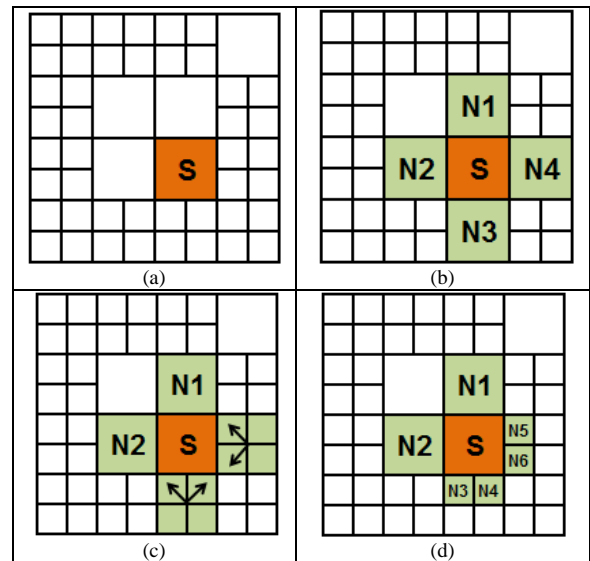


Fig. 4 Finding neighbouring voxels at different depths. (a) Initial seed voxel; (b) Neighbours of the same depth; (c) Child nodes found where necessary; (d) All connected neighbours.

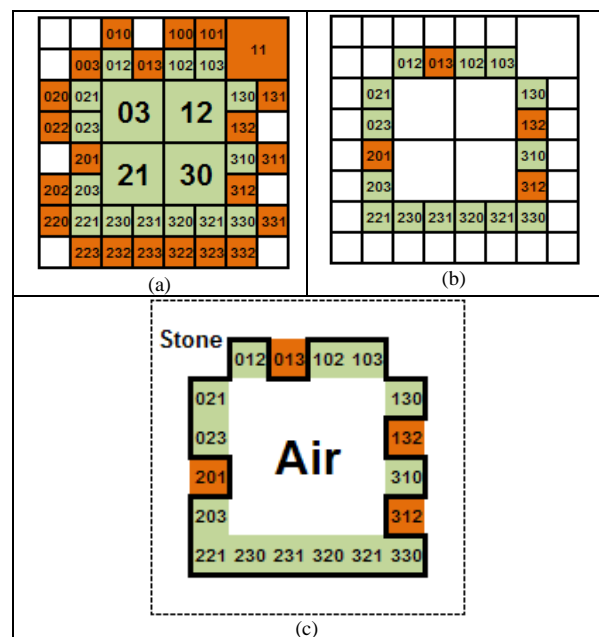


Fig. 5 (a) Boundary voxels; (b) Selecting the smaller of adjacent stone-air voxels; (c) Polygonal surfaces of the cave walls.

#### 4. Results and Discussion

A number of experiments were conducted to evaluate the procedural cave generation and voxel-based octree building method proposed in this research. The proposed techniques were implemented in an application program that was written in C# on an Intel dual-core 3.33GHz PC, and the XNA framework was used for displaying the resulting 3D cave model.

Table 1: Comparison between the number of voxels, vertices and triangles for different octree depths.

| Octree Depth | Boundary Voxels |       |        | Vertices | Triangles |
|--------------|-----------------|-------|--------|----------|-----------|
|              | Stone           | Air   | Total  |          |           |
| 5            | 250             | 172   | 422    | 674      | 1296      |
| 6            | 1012            | 606   | 1618   | 2693     | 5196      |
| 7            | 4235            | 2284  | 6519   | 11135    | 21376     |
| 8            | 17858           | 9501  | 27359  | 47756    | 90760     |
| 9            | 73760           | 43766 | 117526 | 207836   | 388084    |

Table 1 shows the complexity of a typical octree resulting from the 3D cave model that was built using the proposed approach. The cave model was constructed using a spherical foundation with a bias value of 0.05. The table gives a comparison between the total number of boundary voxels in the octree, along with the breakdown of boundary voxels containing stone and air. Without using an octree, the total number of voxels that would have to be stored in memory would equal  $2^{\text{depth}} \times 2^{\text{depth}} \times 2^{\text{depth}}$ . It can be seen that the total number of octree voxels significantly reduced memory requirements. In addition, the total number of vertices and triangles that form the surface of the cave walls are also provided in Table 1.

Experiments were also conducted to compare two common octree construction approaches with the specialised approach that was developed in this research. The first approach that was used to build the octree followed a top-down sequence. In this approach, voxels in a node are sampled and if the node encompasses voxels of more than one type of material (both air and stone), the node is subdivided into eight child nodes. Then the scanline-fill process was performed to remove 'floating islands' from the octree. This was followed by the flood-fill process for 'air pocket' removal. The second approach used was a bottom-up octree construction approach with each node visited in the traditional Morton order [24]. The 'floating island' and 'air pocket' removal process was similar to the first approach described above. The third approach was the method detailed in section 3. To provide

a basis for comparisons, all three approaches were built using the same procedural 3D cave model.

For comparison, the three different approaches were evaluated based on the total number of required material checks, i.e. whether a given voxel's material was air or stone. Table 2 shows the total number of material checks that were done when building the octrees. Table 3 in turn shows the total number of material checks performed on the already constructed octree in order to obtain the resulting cave boundary voxels. It can be seen that in both comparisons, the third approach required less material checks as opposed to the other two approaches. This means that the specialised approach developed in this paper successfully reduced the overall computational requirements.

Table 2: Total number of voxel material checks required in the construction of the octree.

| Octree Depth | Approach 1 | Approach 2 | Approach 3 |
|--------------|------------|------------|------------|
| 5            | 6211       | 4096       | 3992       |
| 6            | 106212     | 32768      | 29642      |
| 7            | 414810     | 262144     | 225313     |
| 8            | 3329425    | 2097152    | 1799992    |
| 9            | 26616879   | 16777216   | 14187327   |

Table 3: Total number of octree node material checks required to obtain the cave boundary voxels.

| Octree Depth | Approach 1 | Approach 2 | Approach 3 |
|--------------|------------|------------|------------|
| 5            | 7706       | 7706       | 3357       |
| 6            | 39154      | 39154      | 17109      |
| 7            | 236444     | 236444     | 90318      |
| 8            | 1588910    | 1588910    | 678910     |
| 9            | 10026594   | 10026594   | 4860953    |

Table 4 and 5 demonstrate how diverse cave models can be generated by adjusting certain parameters to give rise to different cave characteristics. These tables show 2D cross-sections of the cave structure produced by varying the bias value and the Perlin noise frequency. The cave models shown in Table 4 were generated using a spherical cave foundation, whereas the cave models in Table 5 were created from a cubic foundation. It can be seen that larger bias values give rise to caves with rough walls, while smaller bias values result in cave models that closely follow the cave foundation shape. In addition, Perlin noise frequencies with smaller values leads to cave models with smoother surfaces as opposed to when larger values are used.



Table 4: 2D cave structure cross-sections resulting from a spherical foundation with different bias values and Perlin noise frequency.

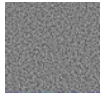
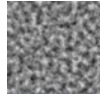


















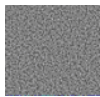
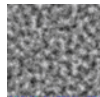


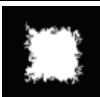















|                |        | <i>Perlin noise frequency</i>  |  |  |  |
|----------------|--------|--|--|--|--|
|                |        | 64   | 16   | 4  | 1  |
|                |        |   |   |   |   |
| <i>Bias, b</i> | 0.5    |   |   |   |   |
|                | 0.05   |   |   |   |   |
|                | 0.005  |   |   |   |   |
|                | 0.0005 |  |  |  |  |

Table 5: 2D cave structure cross-sections resulting from a cubic foundation with different bias values and Perlin noise frequency.

|                |        | <i>Perlin noise frequency</i>   |   |   |   |
|----------------|--------|---|---|---|---|
|                |        | 64  | 16  | 4   | 1   |
|                |        |  |  |  |  |
| <i>Bias, b</i> | 0.5    |  |  |  |  |
|                | 0.05   |  |  |  |  |
|                | 0.005  |  |  |  |  |
|                | 0.0005 |  |  |  |  |

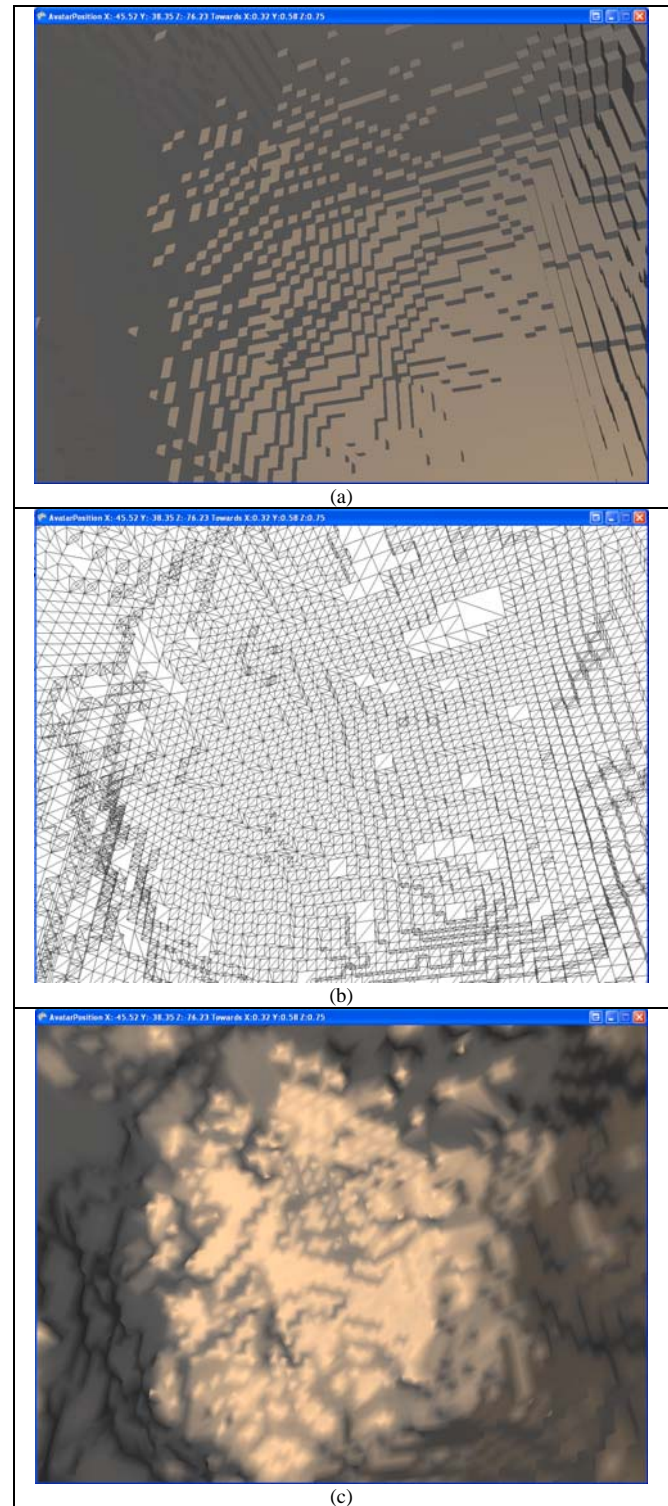


Fig. 6 Screenshots of a 3D cave model. (a) Voxels; (b) Wireframe depiction of the polygonal surfaces; (c) After smoothening.

There are a number of methods for displaying voxel octrees, including ray casting, splatting and iso-surface construction. For real-time rendering, the approach used in this research was to render the polygonal surfaces of the cave walls. Screenshots showing components of an example procedurally generated cave model are shown in Figure 6. In Figure 6a, individual voxels were rendered to highlight the voxel-based cave model. Figure 6b shows the triangles that were formed from these voxels. From the wireframe rendering, one can see that some voxels are larger than others. Figure 6c in turn shows the results of the 3D cave model after a smoothening function was applied to the polygons. The cave model was rendered with directional lighting only, without any texturing.

## 5. Conclusion and Future Work

The automated generation of synthetic 3D cave environments is an important but largely unexplored area of research. This paper presents an approach to procedurally generating diverse cave structures using 3D noise functions by adjusting parameters such as the bias value, overall cave foundation shape and sampling. An efficient method of storing the resulting 3D spatial cave data using voxel-based octrees was also presented along with a specialised bottom-up octree construction approach. Furthermore, techniques to remove 'floating islands' and 'air pockets' from the resulting octree were detailed along with how to determine the boundary surfaces of the cave.

In order to increase the realism of the 3D cave model, real caves have speleothems such as stalactites and stalagmites. Future work will investigate the generation of cave features like stalactites and stalagmites, as well as how to incorporate these into the resulting cave structure. The use of different 3D noise functions to create different cave models will also be examined, along with the parameters required to control the look and feel of the cave structure. In addition, the procedural generation of realistic cave textures is also an important topic for future work.

## References

- [1] Smelik, R.M., Tutenel, T., de Kraker, K.J. and Bidarra, R., "A Declarative Approach to Procedural Modeling of Virtual Worlds," *Computer and Graphics*, Vol. 35, pp. 352-363, 2011.
- [2] Fletcher, D, Yue, Y. and Al Kadar, M., "Challenges and Perspectives of Procedural Modelling and Effects," in *Proceedings of the 14<sup>th</sup> International Conference on Information Visualisation*, pp. 543- 550, 2010.
- [3] Zhou, H., Sun, J., Turk, G. and Rehg, J.M., "Terrain Synthesis from Digital Elevation Models," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 13, No. 4, pp. 834-848, 2007.
- [4] Peytavie, A., Galin, E., Grosjean, J. and Merillou, S., "Arches: a Framework for Modeling Complex Terrains," *Computer Graphics Forum, Proceedings of EUROGRAPHICS*, Vol. 28, No. 2, pp. 457-467, 2009.
- [5] Lluch, J., Camahort, E. and Vivó, R., "Procedural Multiresolution for Plant and Tree Rendering," in *Proceedings of the 2<sup>nd</sup> International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH '03)*, pp. 31-37, 2003.
- [6] Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L., "Procedural Modeling of Buildings," *ACM Transactions on Graphics (TOG)*, Vol. 25, No. 3, pp. 614-623, 2006.
- [7] Parish, Y.I.H. and Müller, P., "Procedural Modeling of Cities," in *Proceedings of the 28<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, pp. 301-308, 2001.
- [8] Sun, J., Yu, X., Baciú, G. and Green, M., "Template-based Generation of Road Networks for Virtual City Modeling," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '02)*, pp. 33-40, 2002.
- [9] Fuller, A.R., Krishnan, H., Mahrous, K., Hamann, B. and Joy, K.I., "Real-time Procedural Volumetric Fire," in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*, pp. 175-180, 2007.
- [10] Hinsinger, D., Neyret, F. and Cani, M.P., "Interactive Animation of Ocean Waves," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*, pp. 161-166, 2002.
- [11] Shi, L. and Yu, Y., "Controllable Smoke Animation with Guiding Objects," *ACM Transactions on Graphics (TOG)*, Vol. 24, Issue 1, pp. 140-164, 2005.
- [12] Flores, L. and Horsley, D., "Underground Cave Sequence for Land of the Lost," in *Proceedings of SIGGRAPH: Talks*, 2009.
- [13] Boggus, M. and Crawfis, R., "Procedural Creation of 3D Solution Cave Models," in *Proceedings of the 20<sup>th</sup> IASTED International Conference on Modelling and Simulation*, pp. 180-186, 2009.
- [14] Boggus, M. and Crawfis, R., "Explicit Generation of 3D Models of Solution Caves for Virtual Environments," in *Proceedings of the 2009 International Conference on Computer Graphics and Virtual Reality*, pp. 85-90, 2009.
- [15] Štáva, O., Beneš, B., Brisbin, M. and Křivánek, J., "Interactive Terrain Modeling using Hydraulic Erosion," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08)*, pp. 201-210, 2008.
- [16] am Ende, B.A., "3D Mapping of Underwater Caves," *IEEE Computer Graphics Applications*, Vol. 21, No. 2, pp. 14-20, 2001.
- [17] Schuchardt, P. and Bowman, D.A., "The Benefits of Immersion for Spatial Understanding of Complex



- Underground Cave Systems,” in Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology (VRST '07), pp. 121-124, 2007.
- [18] Boggus, M. and Crawfis, R., “Prismfields: A Framework for Interactive Modeling of Three Dimensional Caves,” in Proceedings of the 6<sup>th</sup> International Conference on Advances in Visual Computing (ISVC '10), Volume Part II, pp. 213-221, 2010.
- [19] Johnson, L., Yannakakis, G.N. and Togelius, J., “Cellular Automata for Real-time Generation of Infinite Cave Levels,” in Proceedings of the 2010 Workshop on Procedural Content Generation in Games (PCGames '10), pp. 1-4, 2010.
- [20] Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K. and Worley, S., Texturing and Modeling: A Procedural Approach, 3<sup>rd</sup> Edition, Morgan Kaufmann, 2003.
- [21] Perlin, K. and Heffort, E.M., “Hypertexture,” Computer Graphics, Proceedings of SIGGRAPH '89, Vol. 23, No. 3, 1989.
- [22] Laine, S. and Karras, T., “Efficient Sparse Voxels Octrees,” in Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 55-63, 2010.
- [23] Wilhelms, J. and Van Gelder, A., “Octrees for Faster Isosurface Generation,” ACM Transactions on Graphics, Vol. 11, No. 3, pp. 201-227, 1992.
- [24] Morton, G.M., “A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing,” IBM Ltd. Technical Report, Ottawa, Canada, 1966.

Computer Science from the University of New England, Australia in 1996. Her research interests include distributed artificial intelligence, multi-agent systems, agent-based simulation and modeling in complex domains, grid computing, and knowledge discovery and data mining.



**Juncheng Cui** received the B.S. degrees in Software Engineering from Tongji University, China, in 2007. He is currently working toward to the M.S. degree under the supervision of Dr. Yang-Wai Chow and A/Prof. Minjie Zhang. His research interests include computer graphics, procedural content generation and artificial intelligence.



**Yang-Wai Chow** received his BSc., B.Eng. (Hons.) and Ph.D. from Monash University, Australia, in 2003 and 2007. He is currently a Lecturer in the School of Computer Science and Software Engineering, at the University of Wollongong, Australia. His research interests include computer graphics, virtual reality, interactive real-time interfaces, human visual perception and human computer interaction.



**Minjie Zhang** is an Associate Professor in the School of Computer Science and Software Engineering and the Director of Intelligent System Research Group in the Faculty of Informatics, at University of Wollongong, Australia. She received her BSc. degree from Fudan University, China in 1982 and the PhD degree in