Polygonizing volumetric terrains with sharp features*

V. Shakaev
Ltd. "K-Mobile"

In recent years, volume-based terrains have been steadily increasing in popularity due to their vastly superior capabilities compared to conventional elevation-based terrains. However, using a fully 3D representation entails a number of challenges, mainly reducing memory consumption, employing Level-Of-Detail techniques and achieving seamless polygonization for artifact-free rendering. We present a modified Dual Contouring algorithm which is designed for efficient out-of-core isosurface extraction. The algorithm can be interpreted as a bottom-up traversal of linear octrees. Then we describe a compact representation for storing levels of details of a volumetric terrain, particularly well-suited for the above algorithm. Finally, we show how our approach can easily support crack-free polygonization of a volumetric terrain with sharp features and different levels of detail at interactive frame rates.

Keywords: *volumetric terrain, isosurface extraction, dual contouring, out-of-core strategy*

1. Introduction

Interactive terrain rendering is an important component in many applications ranging from GIS visualization to videogames. Traditionally, terrains have been modeled using heightmaps: scalar elevation values sampled over a 2D regular grid. Due to their planar nature, heightmaps can represent only 2.5D surfaces: it's inherently impossible to have caves, arches or overhangs. These limitations can be overcome by using a full-blown 3D, volumetric representation. However, this incurs significant memory cost and increases rendering complexity.

Typically, for interactive rendering of a volumetric terrain a polygonal mesh approximating the surface of the terrain needs to be extracted using a method such as Marching Cubes (MC) [8]. The resulting mesh can then be efficiently rendered in real-time by the graphics hardware. A desirable feature is ability to capture sharp features which allows to represent artificial objects such as buildings on the terrain.

In order to allow for visibility culling, level of detail (LOD) selection and editing, large-scale terrains are usually broken up into separate pieces. Polygonizing each part independently from its neighbours leads to the well-known problem of cracks along the boundary between parts with different levels of detail. In real-time or interactive applications the tasks of loading the necessary levels of detail and seamless isosurface extraction need to be computed at high speed.

To solve these problems we developed a modified version of the adaptive dual contouring algorithm. Our approach is based on the linear octree representation, doesn't have any additional space overhead and is much simpler to implement than existing alternatives. Then we describe a compact representation for storing levels of details of volumetric terrains which is naturally suited for our algorithm. The methods are generic enough to be used for volume tiling and seamless out-of-core isosurface extraction in other application domains as well. Finally, we suggest further improvements and future research directions.

2. Related work

Although there has been separate work on feature-preserving isosurface extraction and on real-time rendering of volumetric terrains, there has been little previous work in considering these areas together.

Isosurface extraction is a well-studied (and still actively researched) problem. Marching Cubes (MC) [8] is the most popular algorithm for isosurface extraction, but it's not feature-preserving. Moreover, it was designed for uniform grids and cannot be applied for crack-free adaptive triangulation.

Dual Contouring (DC) [2] is a unified method to extract crack-free surfaces using both uniform and adaptive subdivision. DC generates one representative vertex inside each boundary cell (a boundary cell intersects the isosurface). When Hermite data (exact intersection points and normals of the surface with cell edges) is available, DC can reconstruct sharp features by positioning the vertex at the minimizer of the quadratic error function (QEF). On a uniform cubic grid, for each zero-crossing cell edge DC forms a quad by connecting the four vertices from the grid cells sharing the edge. The orientation of the quad is derived from the signs at the edge's endpoints (signs denote inside/outside statuses).

The task of polygonizing a voxel terrain can be viewed as a problem of out-of-core isosurface extraction. More recently, [5] presented an adaptive out-of-core isosurface extraction method. The method consists in dividing the volume into rectangular blocks which are then seamlessly combined together, using a modified Dual Marching Cubes (DMC)[9] algorithm adapted to the linear octree representation.

In contrast to traditional heightmap-based terrains, real-time rendering of volumetric terrains has received relatively little attention, with most of research devoted to smooth voxel terrains.

The Transvoxel Algorithm [3] was designed for seamless and high-performance triangulation of

Работа опубликована по гранту РФФИ №16-07-20482

a chunked multiresolution voxel terrain. Between neighbouring blocks with differing resolutions transition cells are introduced, which are triangulated using an extended MC table.
Material stacks [1, 10] is a compact and efficient representation for 3D terrains based on vertical columns with multiple material layers. To render stack-based terrains in real-time [6] developed techniques for LOD generation and seamless isosurface extraction using GPU-based dual contouring.
To generate high-quality, and, possibly, feature-preserving surface meshes at interactive frame rates [7] presented a GPU-based parallel processing pipeline using DMC [9], but didn't address visualization of large-scale terrains with LODs.
Finally, tetrahedral bisection [11, 12] allows for a GPU-friendly LOD algorithm with large volumetric datasets and doesn't need any preprocessing or crack patching.
In this article, we explicitly address the issue of generating surface meshes for interactive rendering of volumetric terrains with sharp features.

3. General approach

In this section we outline our approach to storage and real-time rendering of volumetric terrains. We present a modification to the dual contouring algorithm which accepts as input a signed linear octree and doesn't have any additional space overhead. Then we describe a memory-efficient data structure to represent different parts of the terrain at varying resolutions.

3.1 Surface Extraction

We use adaptive Dual Contouring [2] for generating triangle meshes, because it provides built-in solutions for LOD generation with seamless multiresolution contouring and offers a good compromise between simplicity, performance and mesh quality. Adaptive Dual Contouring [2] generates a feature-preserving and crack-free mesh from an unrestricted, signed octree with different cell sizes. To generate a closed contour, only minimal octree edges (those edges of leaf cells that do not properly contain an edge of a neighboring leaf) should be examined. In an octree, each minimal edge is shared by four or three leaf cells. Following [5], for more efficient out-of-core processing we adopt the linear octree representation.

3.1.1 Dual Contouring of Linear Octrees

A linear octree consists of only terminal, leaf nodes together with their locational codes for uniquely identifying each node within the hierarchy. We use Morton codes as keys for fast neighbor finding [4]. Assuming the key of the root is 1, a Morton code for the given node can be built by recursively descending the tree and concatenating the 3-bit octal number of each encountered node. Then the depth of the node is encoded by the most-significant bit in the node's code and the code of the parent can be obtained by removing the last 3 bits of the node's code.
The use of linear octrees with keys allows to completely abandon internal nodes with pointers and bypass hierarchical traversal. For adaptive isosurface extraction it's necessary to store only boundary, leaf cells. In our implementation, each cell contains the quantized position of the representative vertex, the signs at the corners of the cell and is associated with the corresponding Morton code.
The resulting dual-contouring algorithm can be described in three steps. First, all leaf nodes are sorted by their Morton codes in descending order so that the smallest leaves (the deepest in the hierarchy) are placed at the beginning. This step is required only when the octree is modified (which is always the case in our seamless out-of-core isosurface extraction strategy).
Second, for each leaf cell we create a mesh vertex. The vertex's position is de-quantized using the cell's bounding box, which in turn is computed from the cell's Morton code.
Third, we iterate over each leaf cell and create a quad for each intersecting edge of the cell. Specifically, we search for the other three adjacent leaf cells sharing the edge, and if they are all found, we emit a quad. We build the neighbour's Morton code and check it for overflow. If the neighbour lies deeper than the current cell or is located outside the octree bounds, the current edge is discarded. For neighbour finding we use binary search on the (sorted) array of leaf cells, starting with the next cell (to skip the current cell and previously visited ones). If no adjacent cell is found, the current edge is likewise discarded, and we continue with the next active edge. Finally, if the other three neighbouring cells are found (they are the same size or larger than the current cell), we create a quad by connecting the mesh vertices corresponding to the four leaf cells containing the edge. In transitional areas of the octree there are only three distinct cells sharing the edge, and the quad degenerates into a triangle, as can be seen in Figure 1.
In comparison to [5], the above algorithm doesn't rely on any auxiliary data structure, besides the sorted array of leaf nodes. No explicit tagging is needed for cells/edges that have already been visited. Each leaf cell is processed exactly once, in a sequential manner. Therefore, the running time of the algorithm is roughly linear in the number of leaf nodes.
Compared to the traditional top-down recursive approach [2] where all edges in the octree are enumerated, the bottom-up traversal strategy allows to visit only intersecting edges.
These design choices make the contouring algorithm very simple and easy to implement. Instead of

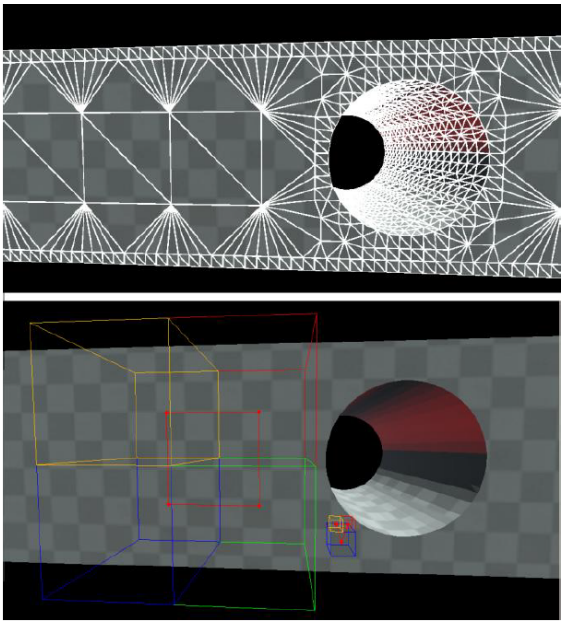

Рис. 1: Surface extracted from an implicitly-defined object using adaptive dual contouring. The lower image shows octree cells in the uniform and adaptive case.

implementing several mutually recursive procedures, the algorithm requires only a few dozen lines of C code.
As long as the leaf cells continuously cover the surface and their signs are consistent with each other, the algorithm will produce a closed, watertight surface. If 2-manifold meshes are required, the algorithm can be extended to adaptive DMC [9, 5], at the cost of increased complexity.

3.2 Terrain Generation

In our tests all data sets have been generated procedurally or created by scan-converting existing triangle meshes.

3.3 Terrain Representation

For enabling out-of-core processing the voxel terrain is broken up into same-sized rectangular blocks (or meta-cells using the terminology from [5]), with up to 4 levels of detail each. In our framework, the finest (lowest) LOD of each block can be stored using several types of volumetric representations, namely, Signed Distance Fields with distance gradients, binary volumes with Hermite data, Points with Implicit Connectivity (PIC) [14] and Layered Depth-Normal Images. Our framework currently handles only solid and empty space.
All coarser levels of detail are stored as signed linear octrees, so that they can be readily used by the adaptive dual contouring algorithm. This representation is similar to PIC, except that we don't store the object's interior (which is required for CSG operations).
In the PIC representation, a solid is adaptively sampled into an octree composed of black (inside), white (outside), and gray (boundary) leaf nodes. Each gray leaf cell contains one representative vertex and inside/outside classification values of the cell corners [14]. PIC provides better memory efficiency than other feature-preserving representations, because Hermite data is not explicitly stored. Additionally, since the positions of all representative vertices have been precomputed, the dual contouring phase reduces to polygon generation.
In our signed linear octree representation only boundary leaf cells are stored which leads to further memory savings. In practice, only a small subset of all cells are boundary, with more cells being allocated in regions with thin features or high curvatures. In our implementation, each cell contains a 32-bit Morton code, the position of the representative vertex, quantized to three 8-bit values based on the cell's bounding box, and signs at the cell corners. The resulting data structure can capture sharp features in a space efficient manner and still have sufficient precision due to hierarchical quantization.

3.4 Level Of Detail

Real-time rendering of large landscapes requires a LOD system to allocate limited computational resources for an optimal balance between performance and visual quality.

3.4.1 LOD generation

To generate levels of detail for each block of the terrain we follow the standard bottom-up approach [2]: we sample the source data at the maximum resolution, build a full-precision linear octree and then simplify the resulting linear octree in-place, until the maximum tree depth at the given LOD is reached.
This approach generates fairly good approximations for low-frequency, coarse terrain, but is not very suitable for models with fine geometrical features as it distorts the original topology and causes thin features to vanish at lower resolutions (which is a common drawback of all clustering-based methods).

3.4.2 Crack-free triangulation

Dual-contouring each terrain block individually, without considering its neighbours, leads to cracks and gaps between adjacent blocks (see Fig.2, left). Adaptive dual contouring offers the ability to generate crack-free multiresolution meshes. To seamlessly connect a terrain block to its neighbours, the octree cells from neighboring blocks, which are face-adjacent to the current block, must also be processed by the contouring algorithm.
However, when later a neighbouring block is triangulated, its triangles will overlap the existing ones

at the boundary between the two blocks, which will cause Z-fighting in the seam region.
To prevent creation of duplicate polygons at block boundaries it's necessary to impose a strict block traversal order and cull redundant edges [5] or modify the dual contouring algorithm so that seam polygons are created only if the corresponding edges belong to different blocks. In the second strategy a linear octree is built for a group of 2x2 blocks. Assuming that the current block is located in the 0th octant, we gather octree nodes from the seven adjacent blocks. Then we create a linear octree for the whole group by inserting a 3-bit octant index just below the depth bit into each cell's Morton code. During octree contouring we create quads only if the corresponding four cells do not belong to the same octant in the group (this can be checked using bitwise operations on their Morton codes).

Рис. 2: "Happy Buddha"divided into blocks with different levels of details and then seamlessly joined using our approach.

4. Implementation and results

The algorithms have been implemented in C++, using Direct3D 11 as a graphics API. All the experiments have been conducted on a machine running 64-bit Windows 7, equipped with a 2.8 GHz quad-core processor, 8 GiB RAM and AMD Radeon 6950.
The resolution of each terrain block is 32^3, although it can be any power of two up to 512 due to the use of the linear octree with 32-bit Morton codes (one octree level is required for stitching), at the cost of slower updates/surface extraction.

We don't maintain any caches for storing the whole terrain in memory. Instead, blocks are loaded as needed without any run-time allocation. We use Lightning Memory-Mapped Database (LMDB) [13] for saving and reading terrain data. As LMDB doesn't support nested transactions, to seamlessly connect a block to its neighbours the boundary cells from adjacent blocks must be kept in memory. On average, it takes approximately 3 ms to triangulate a single block in the PIC format at full resolution.
Currently, LOD loading and triangulation happen in the main thread which causes occasional stalls and freezes during movement. To achieve a constant frame rate, these tasks should be offloaded to a background thread. In addition, popping artifacts are observed during LOD changes.
Even when using adaptive dual contouring the resulting mesh often contains excessive number of triangles in large flat regions. To further reduce triangle count, surface mesh simplification could be applied.
Uniform partitioning scheme doesn't scale to large terrains. For example, the terrain in Fig.3 consists of 16x16x8 blocks which results in about 2 million triangles rendered at over 200 frames per second.
In all our experiments, the whole terrain could fit into the main memory. The terrain shown in Fig.3 occupies approximately 170 MiB on disk as an LMDB map file. All levels of detail are stored as signed linear octrees without any compression.

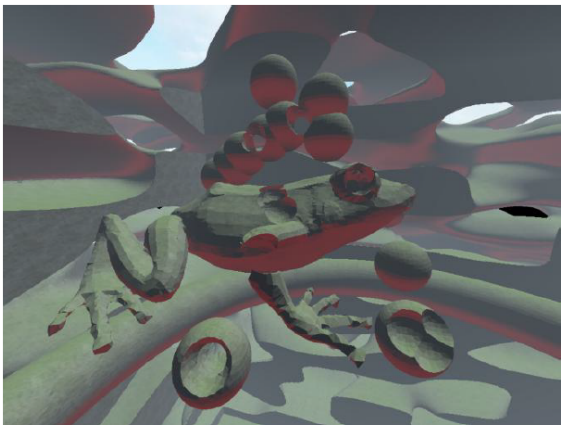

Рис. 3: Randomly generated 3D terrain with a custom mesh.

5. Conclusion and future work

We have described an approach for representing and real-time rendering of volumetric terrains. Our approach combines a compact data structure for storing levels of details and a modified adaptive dual contouring algorithm. Levels of details are stored as signed linear octrees which can be directly triangulated by the contouring algorithm.

The algorithm can be viewed as a simple bottom-up traversal of linear octrees and doesn't have any additional space overhead.

Our approach can be used for seamless out-of-core isosurface extraction in general. We believe our algorithm will perform well on GPU architectures.

Future avenues for research and improvement include: algorithms for generating 2-manifold meshes, such as adaptive DMC [5], topology-preserving simplification for generating better LODs, compression of the signed linear octrees using the properties of Morton codes, support for multiple materials and strategies for hiding LOD transitions.

Литература

[1] Benes B., Forsbach R. Layered data representation for visual simulation of terrain erosion // In Computer Graphics, Spring Conference, 2001, pp. 80–86.

[2] Ju T., Losasso F., Schaefer S., Warren J. Dual Contouring of Hermite Data // ACM Transactions on Graphics, 21(3), 2002, pp. 339–346.

[3] Lengyel, E. Voxel-Based Terrain for Real-Time Virtual Simulations // PhD diss., University of California at Davis, 2010.

[4] Lewiner, T., Mello, V., Peixoto, A., Pesco, S., Lopes, H. Fast generation of pointerless octree duals // Symposium on Geometry Processing 2010, Computer Graphics Forum, vol. 29, 2010, pp. 1661–1669.

[5] Lobello R.U., Dupont F., Denis F. Out-of-core adaptive iso-surface extraction from binary volume data // Graphical Models 76 (6), 2014, pp. 593–608.

[6] Löffler F., Müller A., Schumann H. Real-time rendering of stack-based terrains // In Proceedings of 16th international workshop on Vision , Modeling, and Visualization (VMV), 2011, pp. 161–168.

[7] Löffler F., Schumann H. Generating Smooth High-Quality Isosurfaces for Interactive Modeling and Visualization of Complex Terrains // In Proceedings of 17th international workshop on Vision, Modeling, and Visualization (VMV), 2012, pp. 79–86.

[8] Lorensen W., Cline H. Marching Cubes: a high resolution 3D surface construction algorithm // Computer Graphics (SIGGRAPH 87 Proceedings), 1987, pp. 163–169.

[9] Nielson G. Dual Marching Cubes // In Proceedings of the conference on Visualization, 2004, pp. 489–96.

[10] Peytavie A., Galin E., Merillou S., Grosjean J. Arches: a Framework for Modeling Complex Terrains // Computer Graphics Forum (Proceedings of Eurographics) 28 (2), 2009, pp. 457–467.

[11] Scholz M., Bender J., Dachsbacher C. Level of Detail for Real-Time Volumetric Terrain Rendering // In Proceedings of 18th international workshop on Vision, Modeling, and Visualization (VMV), Vision, Modeling, and Visualization, 2013, pp. 211–218.

[12] Scholz M., Bender J., Dachsbacher C. Real-Time Isosurface Extraction With View-Dependent Level of Detail and Applications // Computer Graphics Forum, vol. 34, iss. 1, 2014, pp. 103–115.

[13] Symas Lightning Memory-mapped Database // 2016. https://symas.com/products/lightning-memory-mapped-database/

[14] Zhang N., Qu H., Kaufman A. CSG operations on point models with implicit connectivity // In Proceedings of the Computer Graphics International, 2005, pp. 87–93.