Visualizing Large Procedural Volumetric Terrains Using Nested Clip-Boxes

Sven Forstmann † and Jun Ohya †

Abstract

This paper proposes a novel terrain-rendering method based on nested Clip-Boxes to visualize massive procedural volumetric terrains. In our method, the terrain is defined as a three dimensional function, provided by the user, which generates appealing looking, unique volumetric terrain shapes. For the visualization, we sample the function in a user defined quadratic view-region around the viewpoint and store the result as voxel volume data in a first step. In the second step, the volume data is converted into a triangle mesh for the hardware accelerated visualization. To provide high quality rendering as well as high computational performance, we employ level of detail by introducing a novel nested Clip Box strategy. Our results show, that using our strategy, 25 frames per second can be achieved on average for a highly detailed landscape. Different from existing methods, ours is the first to allow the immediate visualization of arbitrary sized volumetric terrains in real-time, as it does not depend on any pre-computation.

1 Introduction

1.1 Background

Video games have been popular since their very inception, and their popularity has continued to grow since then. In 2008, video games yielded a market larger in revenue than the movie and music industries [23]. In the video game market, especially Japan plays a significant role as it's share is, with a revenue of over $7 Billion as of 2008 [26], the world's second largest. Despite the increase in revenue, however, development costs have also dramatically increased. For many productions, development costs are approaching or exceeding the revenue of the respective video game [25]. Thus cutting costs is very important in this industry.

In general, a large component of the development cost is the cost of labor for content generation [25]. Since video game users constantly demand new, larger, and more detailed virtual-worlds, the current system of labor-intensive content generation is not sustainable. Therefore, automated content generation is a viable way to greatly reduce development costs.

As users tend to get bored if the same contents are presented each play, the demand for different contents in each play in one video game increases.

In walk-through type of games, the size of a level is normally limited. This is mainly due to hardware constraints, such as the capacity of the disk space and data transfer rates as content is pre-generated and just displayed at game-time. If not very carefully exercised, this often results in a limited walk-through range, or a mere repetition of the plays of the same level already experienced. Hence, the users will become bored. In order to mitigate such effects, the walk-through range should not be experienced as limited, and there should not be any repetition within a given level.

However, repetition is not the only issue. Content detail and flexibility is a problem as well. Conventionally, height-map based methods were used for rendering terrain [1,2]. While height-map based approaches are largely sufficient for video games featuring isometric perspective such as real-time tactics, first person games demand more interesting landscapes, including concavities and overhangs. Therefore, recently height-maps were step-by-step replaced by volumetric terrains [14,20,24] so that more variable terrain landscapes, including concavities and overhangs, can be generated.

The creation of these complicated, and thus interesting, volumetric terrains necessary for long-range walk-through environments can either be achieved by manual operations [8], or procedural methods [20,24]. Manual creation by content creators is expensive in terms of time and financial cost and thus should be reduced as much as possible. Procedural methods save time for creators, however, they produce huge amounts of data, which needs to be stored and loaded again on run-time. To solve this issue, procedural methods need to be integrated into the video game for generating contents at run-time.

1.2 Related Work

Related works can be found in various areas: Academia, video games, and general applications. Previous related work focused either on generating procedural terrains in an offline process, or on the visualization of large and detailed three-dimensional objects in real-time.

† Global Information and Telecommunication Institute, Waseda University

We therefore review two separate types of algorithms. First, algorithms used to generate procedural terrains, and second, algorithms used to visualize large and detailed three-dimensional objects.

1.2.1 Procedural terrains

In games, procedural terrain generation has already been used. An example of this is the successful video game "The Elder Scrolls II: Daggerfall", by Bethesda Software. A massive sized terrain (a flat map, no height information) was one of the main elements of this game.

In academia, procedural terrains can be found as well. P. Prusinkiewicz has developed a method to create fractal height-map based terrains [16]. More advanced is the method of A. Peytavie et. Al. [24]. He has proposed an algorithm to automatically generate large volumetric terrains including caves and overhangs. As in our method, also his method uses volume data for the creation of the terrain.

In other areas, non-game and non-academic, procedural terrain generation has been developed as well. Terragen [15] allows the generation of arbitrary, height-map based terrains. In Pandromeda [14], height-map based terrains and also volumetric terrains can be generated. In both, [14] and [15], the user can freely choose a terrain function. A method that generates a volumetric terrain for the visualization in real-time, is the NVidia Cascades Demo [20]. There, the terrain function is fixed to Perlin Noise [28].

However, all related works create the terrain as an offline process, even though they support to visualize it in real-time as in [20]. There is no algorithm available yet that supports the dynamic generation of procedural volume data on the fly in parallel to the visualization process.

1.2.2 Visualization of large 3D Objects

Since our algorithm visualizes the terrain volume data as polygonal mesh, we also review methods that visualize large and detailed objects, which consist of either polygonal mesh data or opaque volume data.

One published algorithm is [17], where the terrain is represented by a 512x512x64 voxel grid, and visualized by using multi-resolution raycasting.

As for the interactive visualization of large iso-surfaces from volume data, Gregorski et al. [4] present a method that recursively subdivides the scene into diamonds based on pre-calculated error-values. The method is basically a three-dimensional extension of the height-map based terrain rendering method that is known as ROAM [1], and converts the input data into a special format in a pre-processing step.

For visualizing large meshes, several methods have been invented. Most of them, such as [6,12], cluster the input mesh in multi-resolution shapes, such as cuboids or tetraeders. They have to be created in a pre-computation step for the dynamic assembly at runtime. The approach presented by Lindstrom [7] is similar. In his method, vertices are clustered in a hierarchical fashion to achieve the view-dependent LOD.

Other related approaches propose the usage of point sprites, also known as splats, for representing the scene [10,11]. In [10], a combination of splats and polygons is used, where polygons solve the geometry near the viewpoint and splats are used for distant geometry.

A method that utilizes a LOD structure, which is similar to ours, is called GoLD [9]. Here, the mesh resolution is continuously reduced according to distance by switching among several pre-computed detail levels of the initial mesh. The LODs are computed by vertex removal in order to enable a smooth transition by geo-morphing.

However, none of the methods [6,7,9,10,11,12] is suitable for visualizing large on the fly generated volumetric terrain data. All of the aforementioned approaches require intensive preprocessing of the full data set—prior to visualization, and they have to store the complete terrain data to be visualized. Besides the large amount of resources necessary during preprocessing of polygon or volume data as in [10], to create the run-time structure, it is easily visible that the amount of data generated obviates the application of large walk-through ranges.

1.3 Proposed Method

To solve the problems above, we propose a method that can efficiently visualize 3D terrain data that is generated on the fly by a function based procedural approach. Similar to our approach, also [14,15] and [20] use functions to generate the terrain.

The method at hand only requires the terrain functions for generating the underlying volumetric data and their parameters. Storing the entire volumetric data generated from these functions is not necessary. Since any arbitrarily expressive function can be chosen for data generation, the walk-through range and the number of levels are limited only by the parametric range of the function. Due to the possible large range of variations, there is a rich number of distinct concavities, overhangs, and other interesting structures that can be generated in run-time. Note, that as procedural creation of volumetric terrains is already addressed by various methods such as [14,15,20,24], the main focus of this paper is on generating the visualized terrain immediately on the fly, without relying on any pre-processed data. Different from the proposed method, [14,15,20,24] are not able to create and update the terrain data in parallel to the real time visualization.

Our method provides the following benefits.

- Visualize arbitrary massive terrains, including interesting structures such as concaves and overhangs at interactive frame-rates.
- The amount of manual labor necessary for content creation is reduced.
- The walk-through range of a level is potentially unlimited.
- The number of terrains that can be generated is only limited by the number of outputs of the procedural generation function.
- The terrain data is generated on the fly, in parallel to the visualization.

The paper is organized as follows: Section 2 overviews the paper; section 3 explains the clip-box algorithm; section 4 discusses the experimental results; and section 5 concludes the paper.

2 Overview

Our landscape visualization method merges terrain synthetization and visualization in one system. The terrain itself is defined as three dimensional function created by the user. For the visualization, the function is sampled in a cubic region around the view point, and stored as volume data. For the hardware accelerated visualization on the GPU, the volume data is converted into triangle data. The conversion from volume data to triangles is very similar to visualizing iso-surfaces and can be solved by using one of the conventional algorithms such as marching cubes [3]. However, as the amount of triangles arising from direct volume data to polygon conversion is immense, we have to employ an efficient level-of-detail (LOD) approach to our system. This is necessary to keep the polygon-count reasonable for today's graphics hardware.

Nested geometry clip-maps, which derive from clip-maps [13], provide all of our desired features for the two-dimensional height-map based case–however, they cannot solve the three-dimensional volume-data based case.

We hence extend the clip-map based terrain visualization approach of Lossaso and Hoppe [2] on geometry clip-maps to the third dimension by introducing nested clip-boxes, shown in Fig.1. They have very similar properties to clip-maps, but are unlike more complex. Fig. 2 shows an example of a single Clip-Box (CB). In contrast to clip-maps, where nested regular grids suffice to represent the geometry (Fig. 1), CBs carry complex, rapidly changing mesh-topologies. While each geometry clip-map is represented as a rectangular portion of the landscape's height-map, each clip-box represents the iso-surface of a cubic portion of the terrain volume data.

Our algorithm visualizes the terrain using a two-threaded approach that is shown as a diagram in Fig.3. The

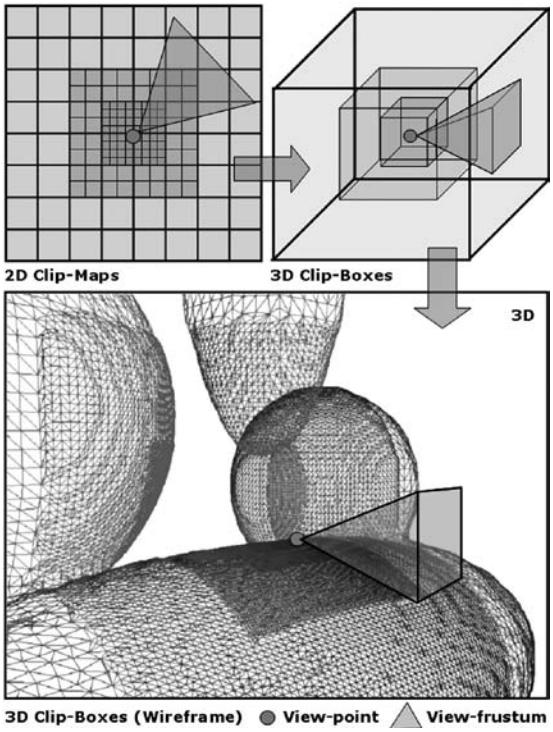

Fig. 1 The evolution from Clip-Map to Clip-Box: Nested geometry clip-maps [6] are shown top left; our Clip-Box based approach as sketch is top right and the final result as a wire-frame below.

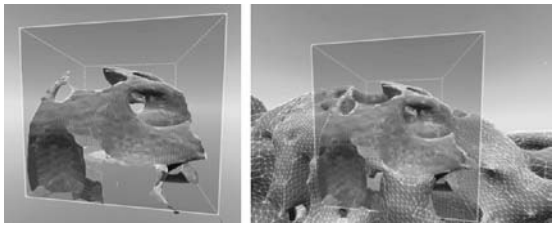

Fig. 2 The Clip-Box: The left image shows the pure Clip-Box geometry, the right shows it embedded into the landscape.

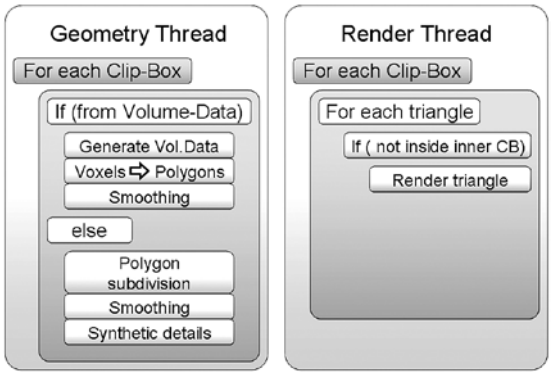

Fig. 3 Algorithm Overview: Using two threads helps to optimally distribute the rendering and voxel to polygon conversion tasks on modern multi-core-CPUs.

first thread with a low update rate creates the procedural volume data and converts it into polygons. The second thread with a high update rate continuously displays the polygons on the screen.

For the procedural terrain generation method, which computes the landscape volume-data to be used by the nested-clip-box algorithm, we use a relatively simple function that produces landscapes complex enough to prove the efficiency of our method. Since the formula for the terrain generation can be defined by the user, we do not focus on inventing a novel formula. However, we refer to three interesting works on offline rendered volumetric terrains that show the large variety of possible landscapes that have been created based on mathematical computations rather than artistic modeling: Pandromeda Mojoworld [14], Terragen [15] and Arches [24]. The references show that volumetric terrains can be much more interesting than height map based terrains – even though they might not be always realistic. Especially in the area of entertainment, realism often is not the main purpose. One of the most successful movies ever, "Avatar" [27], might be the best example. There, a fantasy world called Pandora, with large floating rocks has been one of the main elements in the movie.

3 Clip-Box Algorithm

Our nested Clip-Box algorithm utilizes a simple and efficient structure to represent the terrain mesh. Similar to [2], where the terrain geometry is cached in a set of nested regular grids, our algorithm caches the geometry in a set of nested Clip-Boxes (Fig.1). Once the viewpoint changes, all Clip-Box positions are updated incrementally to preserve the concentric LOD structure.

3.1 Clip-Box

We define a Clip-Box (CB) as the polygonal conversion of a cubic portion of the entire terrain's volume data. This can be seen clearly in Fig. 2, where a pure CB is shown in the left image. The right image shows it embedded into the surrounding landscape. Unlike clip-maps [2], which remain simple regular grids with near constant complexity over time, CBs strongly vary in their complexity as they are shifted through the volume data.

3.2 Data Structure

For each CB, we store the 8-bit volume data where each voxel is either set (opaque) or unset (transparent). The polygon data that is created from conversion consists of triangle strips where each vertex inside the strip carries x- y- and z-coordinates as well as a normal vector.

In addition to these two structures, we further store adjacency information for each voxel to speed up the voxel to polygon conversion process. The links (32-Bit pointers) that we have introduced can be seen in Fig.4. They are utilized as follows:

- Voxel to vertex. Required for inserting a new vertex. It is used to check whether a vertex has already been created for the specific voxel.
- Vertex to vertex. Required for quick smoothing. Each vertex has a list of references to maximal 6 connected points.
- Surface to surface. Required for seeking triangle-strips. Each surface refers to all neighboring surfaces.
- Surface to vertex. Required to access vertices for rendering each surface.
- Vertex to surface. Required for connecting new surfaces. The reference also helps to add the surface-to-surface connections instantly.

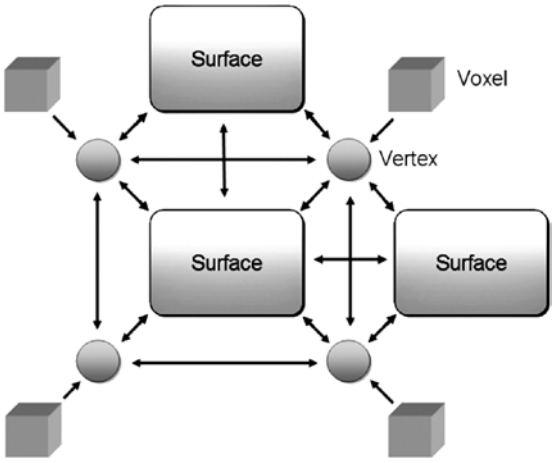

Fig. 4 Adjacency information: Introducing adjacency information helps to speed up polygon extraction and allows for efficient triangle-stripification.

3.3 Procedural Volume-Data Creation

To verify our algorithms' feasibility, we employ a basic procedural volumetric method to generate terrains that are complex enough for testing our algorithm. We therefore apply constructive solid geometry (CSG) operations to the volume data as in Fig.5. We procedurally add and subtract thousands of spheres from the empty voxel-volume using Boolean operations to create complex landscapes for testing purposes. The required parameters, size and position of each sphere, are random values of a user-defined range.

Our approach is similar to [20], only that our method computes the visible terrain portions on the fly, rather than pre-computing the entire terrain.

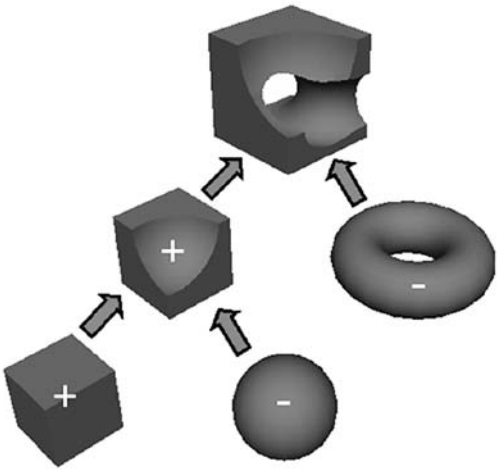

Fig. 5 Landscape synthetization: Complex shapes can easily be created using simple CSG operations.

3.4 Volume-Data to Polygon Conversion

As for the required basic conversion from volume data to polygons, numerous algorithms are available such as [3], [18] or [22]. However, since we also have to consider LOD, the three basic algorithms are not directly applicable. We further need to take care of the following two issues: First, how to close breaks in the geometry at LOD boundaries efficiently (Fig. 6) and second how to achieve a fast conversion. Marching cubes [3] and marching tetraheda [18] achieve a fast and appealing looking conversion from volume data to polygons. However, they complicate welding of two LOD boundaries and also generating adjacency information between vertices for our desired post-processing gets more difficult.

We therefore simplify the conversion process and regard each voxel as a cube with six quadrilateral surfaces. This allows us to efficiently weld bounding LOD levels together seamlessly by further enabling the fast creation of adjacency information. The drawback of this approach is obviously a block-like-looking initial polygonal conversion. We solve this by geometry smoothing in a post-processing step. To weld two LOD levels together, the conversion algorithm considers all voxels in the bounding area of two nested CBs.

For the conversion, our algorithm visits each voxel

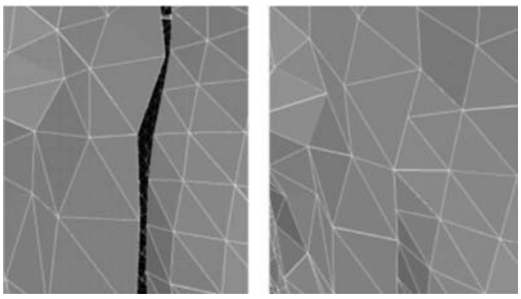

Fig. 6 Clip-Box connectivity: The simple method (left) shows an erroneous gap, while the improved version (right) solves this problem.

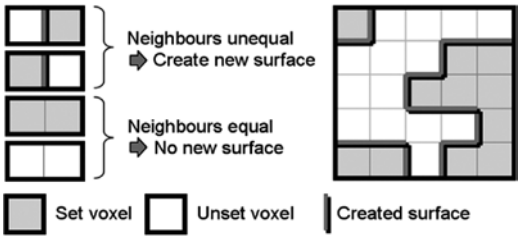

Fig. 7 Voxel to polygon conversion: Surfaces are created by analyzing each voxels bounding neighbors in +x, +y and +z direction.

of the volume-data that is enclosed by one Clip-Box and creates surfaces - if required - by taking direct bounding neighbor voxels in x-, y- and z-direction into account. The used volume-data is binary; each voxel is either set or unset. We included a simple sketch in Fig. 7 to demonstrate the voxel to polygon conversion for the 2D case.

3.5 Nesting

Nesting is required by our algorithm to achieve LOD, which helps to reduce the number of triangles. The LOD is already shown in Fig. 1. The scale factor for the Clip-Boxes increases exponentially by the power of two, while the number of voxels contained by each Clip-Box remains constant. For example, the size of CB one is 100x100x100, the size of CB two is 200x200x200 and so on – however, the number of voxels contained by each CB is constantly 1003. This means for CB one, the voxel size is one, for CB two the voxel size is two, for CB three it is four and so forth. In Fig. 1 we can see that for each CB, all geometry that would interfere with the next inner CB has to be omitted from rendering.

It is also important that all CBs are connected seamlessly without exhibiting gaps at the border- geometry. Gaps occur if the boundaries of two nested CBs are not well connected, as demonstrated in Fig.6. Therefore, once the creation of a CB is finished, bordering vertices are connected properly to the next outer CB to avoid gaps. This can be achieved efficiently by exploiting pointers in the data-structure.

3.6 Moving the View-Point

To fully understand the entire algorithm, it is further necessary to know what happens in case the view-point is moved. In the event that the viewpoint is moved, Fig. 8, it is important to verify all Clip-Box positions in order to preserve our concentric LOD structure. In an ideal case, all Clip-Boxes are permanently centered about the viewpoint, even if the observer starts moving. However, it is impossible to update all Clip-Boxes fast enough. We therefore update inner Clip-Boxes often and outer ones only rarely, as done in Losasso [2]. This approach becomes self-evident when the four steps in Fig. 8 are reviewed. For example

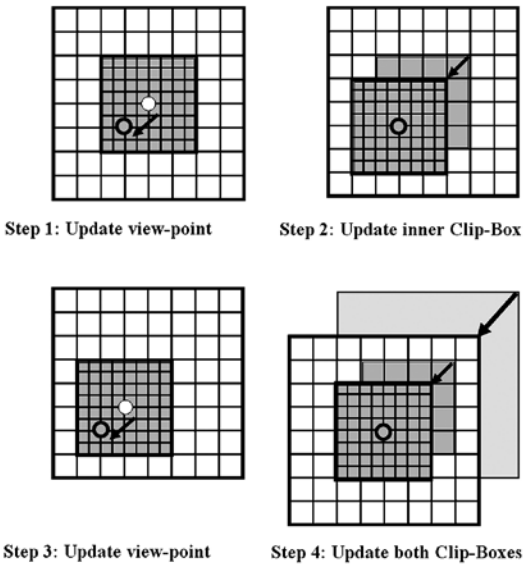

Fig. 8 Moving the viewpoint: In the event that the viewpoint is only moved slightly, it is sufficient to update the inner Clip-Box and let the outer remain at the same position.

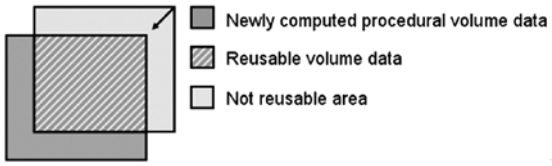

Fig. 9 Caching volume data: After the Clip-Box (CB) is moved, most of the volume data can be reused and only few portions need to be newly computed by the procedural terrain generation algorithm.

the viewpoint change from step 1 to 2 only requires the inner CB to be updated. The outer CB remains at its position, as the viewpoint change is not significant enough. Moving only the inner CB is possible, as the outer CB accommodates all geometry that is enclosed by its volume and can hence cover up for the gap arising from the move of the inner CB. A further advantage of this approach is that we can dynamically adjust the number of triangles on the screen by simply skipping the innermost Clip-Boxes.

To minimize the amount of newly computed procedural volume data in the event of a Clip-Box-update, we cache the previously computed data and only perform differential updates (Fig. 9). The updated portions are referred to as newly computed data inside the Figure.

3.7 Geometry Post-Processing

After the surfaces are obtained, we apply smoothing by Laplacian filtering [19]. This significantly improves the visual quality, since the mesh is very block-like after the initial conversion. In Fig. 10, the difference between image one and two is clearly visible, as image one shows the immediate result after conversion, while image two shows the smoothed geometry.

In the event that a high update-rate for small CB's near the viewpoint is desired, our algorithm enables fast creation of Clip-Box geometry from surface subdivision, rather than using the more complex extraction from volume-data. Fig.11 shows the result of creating the innermost one to four CBs from surface subdivision. Subdivision is done according to Fig. 12.

Even though we do not propose a novel terrain generation method, we added a post processing effect that helps make the generated terrain look more interesting. Our method therefore supports synthetic details by random midpoint displacement [16]. The effect can be seen in Fig. 10, images three and four.

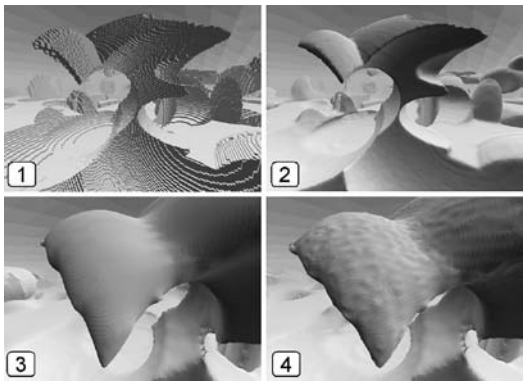

Fig. 10 Geometry-processing: The four images show the proposed steps to process the initial mesh: (1) direct conversion from volume data (2) smoothed (3) surface subdivision (4) synthetic details.

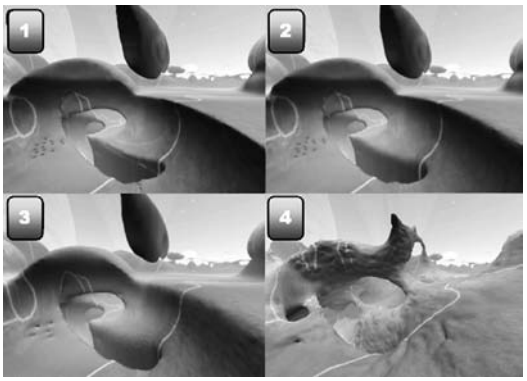

Fig. 11 Fractal details: Increasing the number of CBs generated from surface subdivision plus random midpoint displacement often leads to more natural and appealing terrains (image 4) than the initial result (image 1).

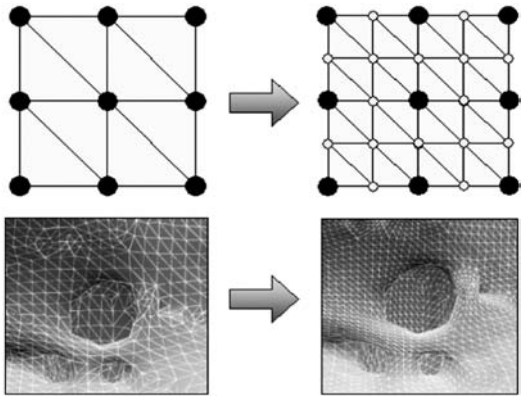

Fig. 12 Triangle subdivision: For each vertex, three additional vertices are inserted as above to preserve the near regular grid structure of our Clip-Box mesh.

3.8 Implementation Details

Our algorithm has been implemented by using C++. For the graphics API, OpenGL has been employed. We use a two-thread approach to separate geometry processing from rendering (Fig. 3). This approach maps well to the current generation of multi-core processors, as each thread is able to occupy one core. Each thread uses the corresponding CPU core to 100 percent continuously. Load balancing has not been implemented. The task distribution of the two threads is as follows:

The geometry thread is in charge of computing the CB's mesh. This involves polygon extraction from voxel data, triangle subdivision, mesh smoothing and random midpoint displacement (Synthetic details). To further improve the performance, we added a module to group all surfaces into triangle strips, allowing cache-optimal rendering. This is done by a depth-first search, utilizing the surface-to-surface connectivity information.

Thread two, the rendering thread, is in charge of rendering all CB meshes correctly by sparing the triangles of the next smaller CB inside. As it runs parallel to the first thread, we have to be aware of concurrent use of the mesh data. We solved this by implementing a double-buffer system, where each mesh buffer is assigned to one thread. Then, once a CB update is completed, the buffers are swapped synchronously.

In case of low voxel resolutions with many subdivision levels, problems near certain voxel patterns often occur that strongly affect the smoothed result. In Fig. 13, those critical regions are emphasized with a white circle. We therefore employed a simple filter (lower left border in the Figure 13) that detects and reduces these patterns by search and replace. The result (right) indicates that most of the problematic patterns from the left image can be eliminated successfully.

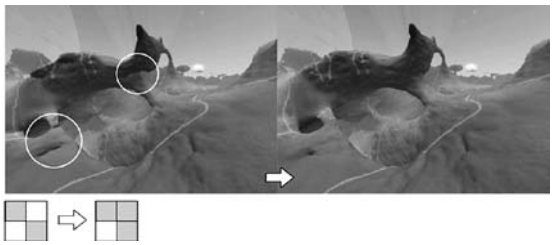

Fig. 13 Smoothing errors: Applying a simple filter operation on the volume data can avoid most problems (Marked by white circles). The filter seeks the left pattern shown below and replaces it by the right one.

3.9 Limitations

Since our method is based on volume data, the average memory consumption is higher than conventional height-map based methods such as geometry clip-maps.

Regarding the geometry update of a clip box in case that the view-point is moved, this might be slightly visible in case of low clip-box resolutions.

4 Experimental Results

Results from our method can be seen in Fig. 14, where numerous landscapes demonstrate the variety of terrains that might be visualized. The upper image shows a terrain that is additionally enhanced by shaders for the grass and handcrafted items to demonstrate the applicability for computer games. The following images below have been included to give further impressions of what is possible with volumetric terrains in general.

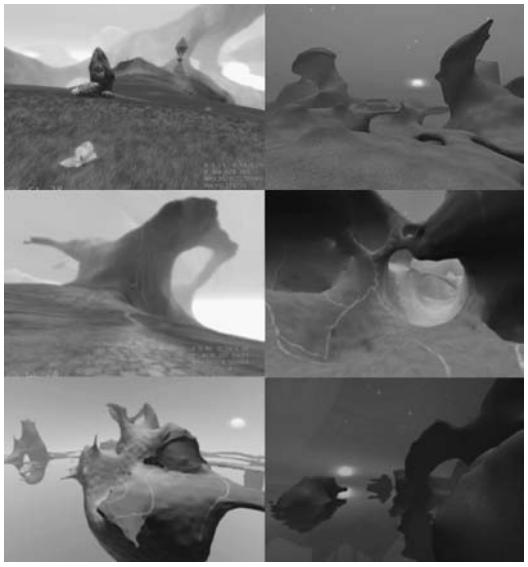

Fig. 14 Complex topologies: The presented method is able to visualize arbitrary landscape topologies, which cannot be visualized using conventional height-map based methods.

In Fig. 15, we demonstrate that our method can be adapted to conventional height-maps as well, where the height-map serves as source for the CB volume data. The height-map and the color-texture are public available on the USGS servers [21]. The major difference by rendering height-maps in volume based methods to conventional height-map based methods is the vertical resolution. While the vertical resolution of our volume based method is reduced with each level of detail, height map based methods, such as geometry clip-maps, have a constant vertical resolution such as 16 bit integer per height-map pixel.

To evaluate our method's performance, we generated an example terrain consisting of about 50000 CSG operations, which can be seen in Fig. 16. The hardware for testing has been a dual core Pentium D 3.0 Ghz, equipped with 1GB RAM and an NVIDIA GeForce 8600 GTS graphics card.

To analyze the speed performance, we prepared two benchmarks. First, a detailed timing of the algorithm pipeline in Table 1, and second an evaluation of the continuous timing behavior of a flight lasting 222 seconds through a landscape, shown in Fig. 17.

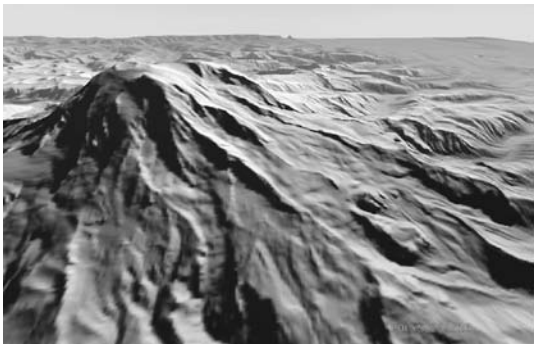

Fig. 15 Real data: Our method may also handle conventional height-map data. Here the Puget Sound region in WA, USA.

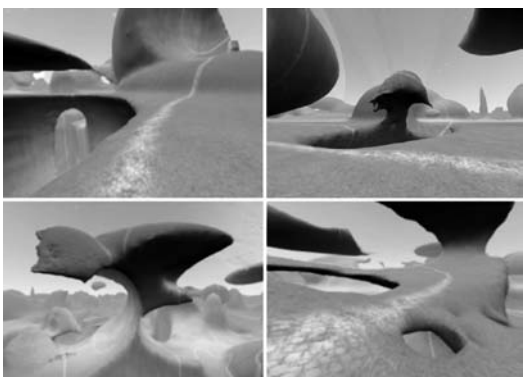

Fig. 16 Benchmark scenario: Various screenshots of the terrain used in our performance measurements.

In the first benchmark of Table 1, we tested the timings for one CB resolution (128) in detail and further compared the results among different CB resolutions.

In the test, 5 out of the 7 CBs are created from

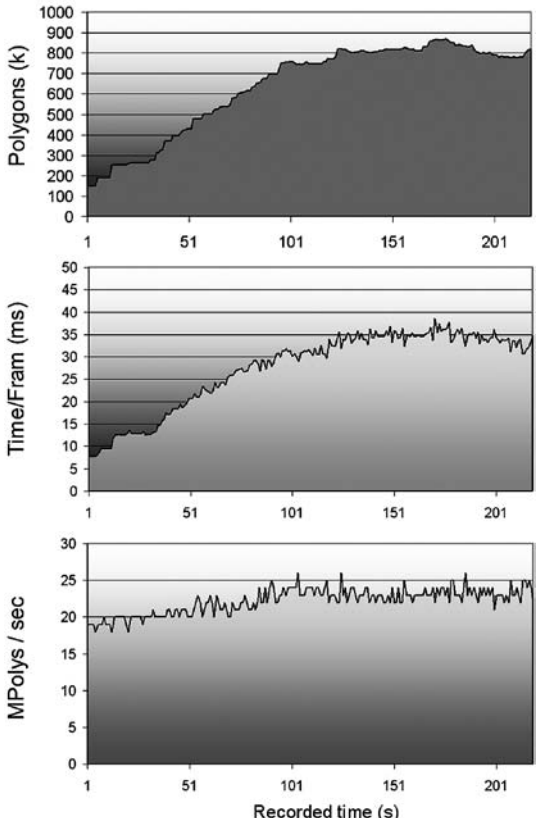

Fig. 17 Continuous performance: We recorded a flight lasting 222 seconds into the example landscape of Fig.16 and captured the amount of polygons, the elapsed time per frame and the render performance in million polygons per second. The CB resolution has been set to 128^3 for this test.

Table 1 Performance analysis: In the upper row, update and render times for one CB resolution (128) are analyzed in detail, while the lower row compares the performance of different CB resolutions.

ClipBox 128	1	2	3	4	5	6	7
Procedural generation	0	0	16	23	114	116	42
Voxels to polygons	0	0	658	674	727	885	745
Subdivision	29	41	0	0	0	0	0
Smoothing	0	2	96	137	152	174	178
Fractal details	4	7	0	0	0	0	0
Surface normals	13	16	28	39	43	41	52
Triangle-strips	4	7	18	61	52	49	76
Total (ms)	51	73	816	933	1088	1265	1093

Render(ms)	1	1	3	6	6	5	6
Polygons (k)	39.6	40.2	90.2	121	127	113	161

ClipBox Resolution	192	160	128	96	64	
CB update avg. (ms)	2671	1315	989	476	162	Thread 1
Render Total (ms)	68	47	29	17	8	Thread 2
Polys Total (k)	1657	1115	692	352	134	
Memory usage (MB)	418	410	265	151	108	

volume-data (CB no. 3 to 7), whereas the two smallest (no.1 and 2) are created from subdivision and enhanced with fractal details (random mid-point displacement). The equivalent of the visualized data volume has been 2048^3 voxels.

As for the timing evaluation, we can see that most time is spent for the surface extraction process (Voxels to polygons). As for the procedural volume data generation, it requires relatively less time, which is a result of the employed caching scheme. If caching is switched on, about 80% of a CB's volume data can be reused during a CB update, which reduces the average time for the procedural computation from 100ms to about 20ms.

In the lower half of Table 1, different CB resolutions are compared. To make the use of multi-threading more clear, we refer to the Geometry Thread as *Thread 1* and to the Render Thread as *Thread 2*. In the table, we can see that the average time to update one CB (CB update avg.) is roughly proportional to the number of processed voxels. More generally speaking, the update frequency for a CB resolution of 128 is sufficient for an interactive exploration at high quality, but it is not suited well for a fast flythrough. In this case, either lower resolutions such as 96 or 64 are suited well, or an increased number of subdivision levels can also be helpful, as well as the earlier mentioned opportunity to skip the innermost CBs. In many cases, an increased number of subdivision-splits combined with random midpoint displacement might even be desirable. Doing so, most CBs are not only updated faster, the terrain also receives a completely different style, which is often more appealing and natural than the initial terrain without using subdivision. In Fig.11 this behavior is shown in four steps, where each step is equivalent to generating one more CB from subdivision.

In the second performance test, we analyzed the frame-rate continuity of our method. Often, visualization algorithms using LOD have difficulties to provide a continuous frame rate since geometry updates are causing short stalls in rendering for many methods, which can be observed as hic-ups in the frame rate. To confirm that the proposed method does not have this problem, we recorded benchmark data over a longer period of time while flying through the artificial terrain of Fig. 16. The resulting diagram can be seen in Fig. 17. However, even at polygon-counts around 800k, the triangle throughput remains continuous at about 20 million triangles per second and does not reveal major peaks. If we further regard the time to render one frame (time/frame), we notice that it changes smoothly in proportion to the scene' s complexity (Polygons). Our algorithm does therefore not reveal any problems that might occur due to the LOD. The frame-rate ranged from 25 to 130 frames per second, which is sufficient for interactive applications such as video games.

In order to measure the render quality of the visualized landscapes, we analyzed the landscape of Fig. 16 at different Clip-Box resolutions by disabling subdivision and texturing. As a reference, we chose the highest possible resolution that our hardware was able to handle, a landscape with 7 Clip-Boxes at a resolution of 192. This is equivalent to visualizing a total data volume of 12288^3 voxels, which would require roughly 210 GB of memory, assuming each voxel is represented by a single bit. To measure the screen-space-error, we compared the renderings of lower Clip-Box resolutions to the reference resolution, as can be seen in Fig.18. To evaluate the error-map, we gray-scaled all images and marked each pixel as erroneous that differed more than 20 in a range of 0 to 255 from the reference image and hence have been noticeable.

The qualitative results show that we can achieve good

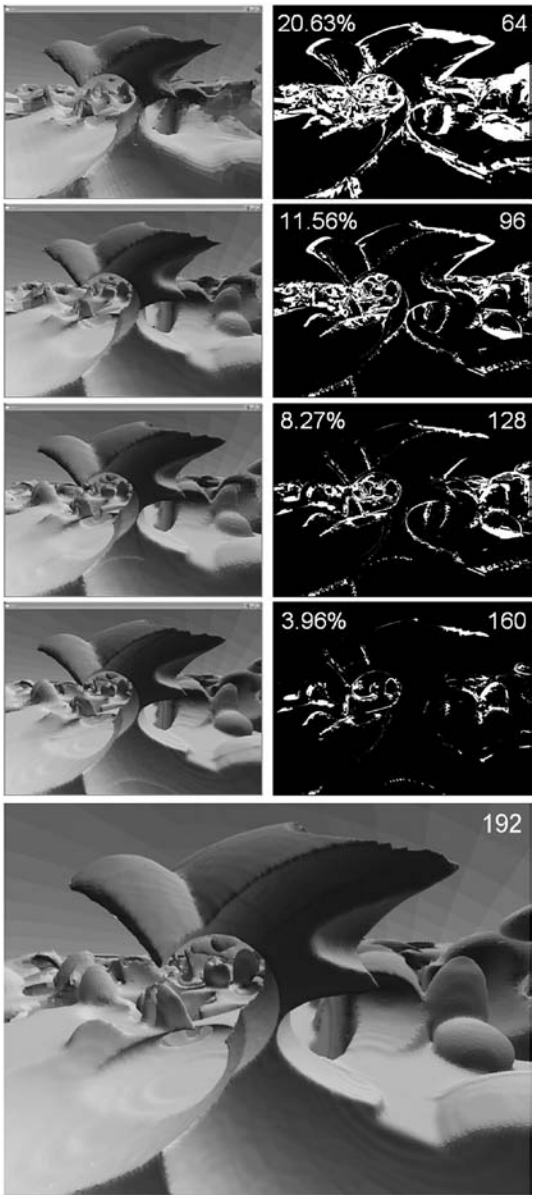

Fig. 18 Screen-space error: The highest Clip-Box resolution (192) was compared to the lower Clip-Box resolutions 64, 96, 128 and 160. (Subdivision has been disabled for this test)

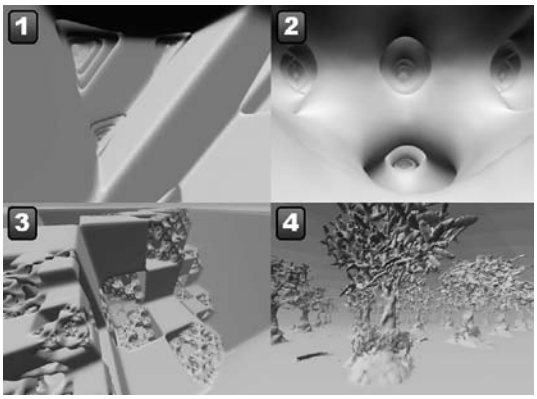

Fig. 19 Function plotting and real data: Our method can be applied to visualizing mathematical problems in an interactive walk-through, images one to three, as well as for conventional iso-surfaces in image four.

quality renderings if the Clip-Box resolution is at least 128. For lower resolutions, the screen-space error increases rapidly and leads to more inaccuracies especially at high distant geometry. As for the quality in general, we observe an asymptotic error behavior, where the error is about halved for each increase in the resolution.

To show further application areas of our method, which exceed the world of gaming, we show that our method can also serve as a 3D function grapher to visualize general math problems. Our method is able to visualize any function f_{Math} that is defined as follows:

$$f_{Math}: \mathbb{Z}^3 \rightarrow [0,1] \quad (1)$$

The function input is defined as a three dimensional integer coordinate vector (Euclidian space), while the output is defined as zero (represented as air in the visualizer) or one (represented as solid terrain). We have prepared results of three generic functions in Fig. 19, image one to three, to show this ability. There, we visualized exclusive-or (1), saw-tooth (2) and sine curve (3). As the evaluation and visualization are done immediate, it is further possible to alter the function parameters on run-time.

To demonstrate further the applicability to conventional rendering of iso-surfaces, we included image (4), which shows a forest generated from the well-known bonsai tree data set. We can clearly see the different levels of smoothing, which have been used from near to far in order to limit the loss of geometric details. The tree that has been used was rescaled to a resolution of 256^3 and placed in the landscape 25 times. The tree scene as well as the function plot scene has been rendered with a CB resolution of 192 at about 10-15 fps.

5 Conclusion

We have presented a novel approach that is able to visualize large procedural volumetric terrains at high quality based on nested Clip-Boxes. We even achieved visualizing a 12288^3 voxel sized cubic window of the complete landscape's volume data at interactive frame-rates. We therefore believe that our method can efficiently be used to visualize interesting looking terrains with so far unseen size for video-games that may change each time the player starts the game by consuming only a negligible amount of memory on the mass-storage device and only posing minimal effort for the artist.

REFERENCES

[1] M.Duchaineau, M.Wolinsky, D.E.Sigeti, M.C.Miller, C.Aldrich, M.B.Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes", VIS1997, 81-88, 1997
[2] F.Losasso, H.Hoppe, "Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids", Siggraph 2004, 769-776, 2004
[3] W.E.Lorensen, H.E.Cline, "Marching cubes: A high resolution 3D surface construction algorithm", SIGGRAPH '87, 163-169, 1987
[4] B.Gregorski, "Interactive View-Dependent Rendering of Large IsoSurfaces", Visualization 2002, 475-484, 2002
[5] A.Knoll, "A Short Survey of Octree Volume Rendering Techniques", GI Lecture Notes in Informatics, Proceedings of 1st IRTG Workshop, 2006
[6] C.Erikson, D.Manocha, W.Baxter, "HLODs for faster display of large static and dynamic environments",SI3D '01, 111-120, 2001
[7] P.Lindstrom, "Out-of-Core Construction and Visualization of Multiresolution Surfaces", SI3D 2003, 93-102, 2003
[8] Coat 3D V3 Voxel sculpting: (visited August 2011) http://www.3d-coat.com/v3_voxel_sculpting.html
[9] L.Borgeat, G.Godin, F.Blais, P.Massicotte, C.Lahanier, "GoLD: Interactive Display of Huge Colored and Textured Models", Siggraph 2005, 869-877, 2005
[10] E.Gobbetti, F.Marton, "Far Voxels: A Multi-resolution Frame-work for Huge Complex 3D Models", Siggraph 2005, 878-885, 2005
[11] S.Rusinkiewicz, M.Levoy, "QSplat: A Multi-resolution Point Rendering System for Large Meshes" Siggraph 2000, 343-352, 2000
[12] P.Cignoni, F.Ganovelli, E.Gobbetti, F.Marton, F.Ponchio, R.Scopigno, "Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models", Siggraph 2004, 796-803, 2004
[13] C.C.Tanner, C.J.Migdal, M.T.Jones, "The clipmap: A virtual mipmap", ACM SIGGRAPH 1998, 151-158, 1998
[14] Pandromeda: http://www.pandromeda.com, visited August 2011
[15] Terragen: http://www.terradreams.de, visited August 2011
[16] P.Prusinkiewicz, M.Hammel, "A Fractal Model of Mountains with Rivers", Graphics Interface '93, 174-180, 1993
[17] M.Wan, N.Zhang, H.Qu, A.Kaufman. "Interactive Stereoscopic Rendering of Voxel-based Terrain", IEEE Virtual Reality, pages 197-206, 2000
[18] G.M.Treece, R.W.Prager, A.H.Gee, "Regularised marching tetrahedra: improved iso-surface extraction", Computers and Graphics 1998, 23(4): 583-598, 1998
[19] L.R.Hermann, "Laplacian-Isoparametric Grid Generation Scheme", J. of the Engineering Mechanics Division of the American Society of Civil Engineers, 102: 749-756, 1976
[20] Ryan Geiss, Michael Thompson, "NVIDIA Demo Team Secrets-Cascades", technical presentation at the Game Developers Conference 2007, 2007
[21] United States Geological Survey (USGS) http://www.usgs.gov, visited August 2011

[22] Tao Ju, Frank Losasso, Scott Schaefer, Joe Warren, "Dual contouring of hermite data", SIGGRAPH '02, 339-346, 2002
[23] Times: "Computer games to out-sell music and video", http://business.timesonline.co.uk/tol/business/industry_sectors/technology/article5085685.ece (visited August 2011)
[24] A.Peytavie, E.Galin, S.Merillou, J.Grosjean: "Arches: a Framework for Modeling Complex Terrains", Eurographics 2009, Volume 28, pp.457-467, 2009
[25] MSNBC: Top video games may soon cost more, http://www.msnbc.msn.com/id/3078404/, visited August 2011
[26] Analysis: Trends in the Japanese Game Market: http://www.gamasutra.com/php-bin/news_index.php?story=20461, visited August 2011
[27] Avatar the movie (visited August 2011): http://en.wikipedia.org/wiki/Avatar_%282009_film%29
[28] K.Perlin: Perlin Noise (visited August 2011) http://en.wikipedia.org/wiki/Perlin_noise