THE VISUALIZATION OF UNSTRUCTURED POLYHEDRAL GRIDS

BY

BRAD RATHKE

BS, Binghamton University, 2009

THESIS

Submitted in partial fulfillment of the requirements for
the degree of Master of Science in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2015

Accepted in partial fulfillment of the requirements for
the degree of Master of Science in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2015

August 4, 2015

Kenneth Chiu, Chair and Faculty Advisor
Department of Computer Science, Binghamton University

Lijun Yin, Member
Department of Computer Science, Binghamton University

Abstract

Efficient visualization of volumetric data sets is an ever present need in the scientific visualization community. While interactive visualization of regularly structured grids is a largely solved problem unstructured grids are a persistent problem. Most attempts at improving the speed at which unstructured grids may be rendered have focused on utilization of GPU hardware which is in general only available in a minority of systems in any given data center. This thesis presents a method of visualizing unstructured grid data sets without the use of GPU hardware using a software ray tracer built as a plug-in module for the Intel OSPRay ray tracing framework. The method is capable of implicit isosurface rendering and direct volume ray casting of homogeneous unstructured grids and multilevel data sets with interactive frame rates and scales with variable SIMD widths and thread counts.

Acknowledgments

Firstly, I wold like to express my gratitude to my adviser Professor Kenneth Chiu for his continuous support for my graduate studies, for his patience, motivation, and immense knowledge of the field. His guidance helped me during the time of my research and in the writing of this thesis. I could not have asked for a better adviser and mentor for my graduate studies.

Additionally I would like to give special thanks to my other committee member Lijun Yin for contributing his expertise in the field of computer graphics to this process.

My sincere thanks goes also to Jim Jeffers, Ingo Wald, Gregory P. Johnson, Gregory S. Johnson, Bruce Cherniak, and Tim Rowley who allowed me the opportunity to join their team at Intel as an intern. Without their knowledge, support, and continual encouragement this research would not have been possible.

Lastly I would like to thank my family and friends, all of whom continue to support me throughout my studies and inspire me to aim higher.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

- AMR - Automatic Mesh Refinement
- AO - Ambient Occlusion
- BST - Binary Search Tree
- BVH - Bounding Volume Hierarchy
- CT - Computerized Tomography
- DVR - Direct Volume Ray Casting
- GPU - Graphics Processing Unit
- ISPC - Intel SPMD Program Compiler
- MMBVH - Min-Max Bounding Volume Hierarchy. AKA Implicit BVH.
- MMKDTree - Min-Max KDTree. AKA Implicit KD Tree.
- MRI - Magnetic Resonance Imaging
- RGBA - Red Green Blue Alpha color representation
- SAH - Spatial Area Heuristic
- SIMD - Single Instruction Multiple Data
- SPMD - Single Program Multiple Data
- SPP - Sample Position Packet
- TSFSL - Two Sided Face Sequence List

Chapter 1

Introduction

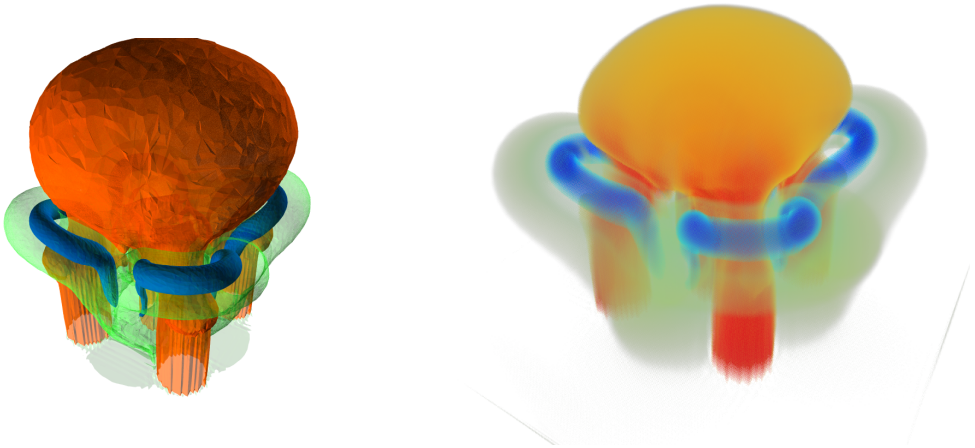
As the computation power has increased over the years domain scientists have increased the sophistication of their simulations. This increase in simulation complexity has allowed for continual leaps in both the scale of the simulations themselves as well as the volume and accuracy of the generated data. Modern simulations generate data on such a scale that the time-steps stored are moving ever further apart to account for both the final size on disk as well as the time taken to write the data. Because storage of a given data set is so space intensive it is not desirable to keep multiple copies in various formats.

Techniques to optimize the rendering of massive simulation data sets have often focused on regular grids or polygonal data sets. However, some simulations generate unstructured or multilevel grids and these techniques are not effective for such data sets. Since resampling the unstructured or multilevel data into a structured grid may not be acceptable it is desirable to have the ability to render unstructured grids directly.

In this work we will describe our approach to the visualization of unstructured polyhedral data sets as well as its integration into an existing scientific rendering framework. It is demonstrated that the rendering technique allows for the rendering of homogeneous unstructured polyhedral grids in interactive time as either an implicit isosurface or through direct volume ray casting. Our technique also avoids any resampling or reformatting of the grids themselves by allowing for directly working on the source data.

We briefly summarize previous techniques for volume rendering in Chapter 2. In Chapter 3 we analyze the aspects of unstructured grid data that causes structured grid rendering techniques to be ineffective as well as why resampling an unstructured grid into a structured

grid may be unacceptable. Chapter 4 presents our technique for implicit isosurface rendering and direct volume ray casting of unstructured polyhedral grids which was first presented in [14]. Further exploration of the results from the previous paper will be presented in Chapter 5 and finally our conclusions will be presented in Chapter 6.



(a) (b)

Figure 1.1: The "Jets" data set rendered via multiple isosurfaces each of which have their own material properties (a) and via direct volume rendering using a hand tuned transfer function such that the resulting image has a similar form to the multi-isosurface presentation (b). Images from [14]

Chapter 2

Volume Rendering

2.1 What is Volume Rendering

Volume data sets are in simplest terms a 3D scalar field. The scalar values in a field may represent any physical value of the data set. For example a volume data set generated through the use of an MRI may use the scalar field to represent variations in density of the scanned object.

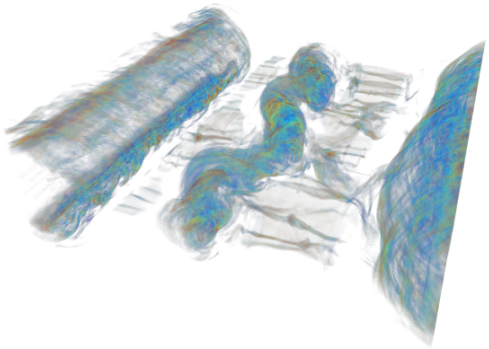


Figure 2.1: The "Magnetic Reconnection" [5] structured volume data set rendered via direct volume ray casting using the OSPRay Volume Renderer [1].

There are various types of 3D scalar field such as regular grids, polyhedral grids, or curvilinear grids. All types of 3D scalar field can be categorized into one of two groups; structured or unstructured grids. A structured grid has a scalar field for which a given topological pattern is repeated in all directions of space. A regular grid is a canonical example of a structured grid. The second group of 3D scalar fields has no topological pattern and is known as an unstructured grid. A collection of connected tetrahedra of varying dimensions is an example of an unstructured grid.

Given the above description of volume data sets, volume rendering can be defined as

a set of techniques used to visualize a 3D scalar field. In general volumes are visualized by either generating an isosurface representation of the volume or by direct volume ray casting (DVR). The basic overview of isosurface generation is to inspect each voxel of a grid and extract a surface for the isovalue of interest for every voxel. DVR in general follows a process of casting a ray through a volume and marching along it taking samples assigning an RGBA value to the sample through a transfer function. Volume rendering methods will be covered in more detail in section 2.4

It can be difficult to come up with a useful transfer function for any given data set and each data set will likely need one or more unique transfer functions to highlight the useful areas of the scalar field. In general transfer functions are created through trial and error, or with *a priori* knowledge of the data set. Changes in transfer function can drastically change the resulting image.

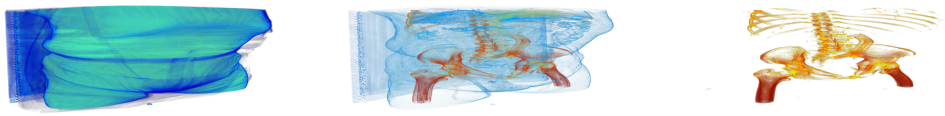


Figure 2.2: Three DVR renderings of a CT scan of the abdomen and pelvis. Each rendering is done with a different transfer function to highlight different scalar field value ranges.

2.2 Applications of Volume Rendering

Volume visualization is a useful tool for any domain scientist to have at their disposal for data analysis. 3D visualization of scalar fields allows for visual data exploration and analysis which can be used to confirm a given hypothesis about the data. Given its usefulness volume visualization has found a home in many scientific fields.

Volume visualization is used often in medical fields for visualization of machine-generated

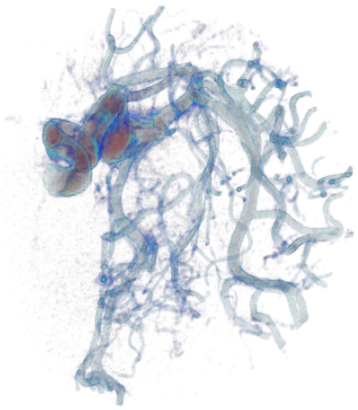


Figure 2.3: A DVR visualization of a publicly available grid generated from a rotational c-arm x-ray scan of the arteries of the right half of a human head. An aneurysm is present.

imagery of a patient (e.g. MRI, CT, PET, or ultrasound scans)[18][16][19]. The patient scans can be visualized to assist in diagnosis by allowing medical professionals to accurately see inside of a patient without surgery. For instance a visualization of an MRI could be used to assist a radiologist in planning radiation therapy for a cancer patient. Medical schools may also keep a library of scans which students could use for training purposes.

Volume visualization is also often used in scientific simulation. For instance a fluid dynamics simulation of turbulence around a virtual automobile can be used by engineers when fine tuning its aerodynamics before physical production. Meteorological simulations can also be visualized to assist in climate modeling.

Figure 2.4: Isosurface representation of shock waves emitted from the epicenter of an earthquake.

Seismology is yet another field in which volume visualization may be applied[8]. Volume visualization has seen wide use in the oil and gas sectors to assist with exploration of oil well sites. Seismologists can also use visualizations generated from real earthquake data and compare with visualizations of simulated earthquake data to assist in the modeling of future seismologic events

As we can see volume visualization can be applied in a wide range of fields. In fact it may be applied in any field for which a 3d scalar field can be generated.

2.3 Structured and Unstructured Volumes

As was mentioned previously there are two overall groups of volumes. In this section we will detail the key differences between the two groups.

Although there are various types, structured volumes follow a well defined set of rules for their construction. All structured volumes have the points in their scalar field aligned such that a pattern is followed. For instance, a sphere may be divided into a curvilinear grid by segmenting it out from the radius. Although such a grid will have voxels that vary in size, it is still a structured grid because a regular pattern is followed.

As a rule of thumb a 3D scalar field is structured if the voxel containing a 3D cartesian coordinate can be determined implicitly. Because of the voxels can be located implicitly sampling and traversal of the volume is relatively cheap when compared to unstructured grids.

An unstructured grid has no real rules regarding its topology. Such grids can be formed of one or more types of polyhedra and cell sizes may or may not vary wildly. Because of these properties we cannot implicitly determine which voxel to use for a given sample position nor can we tell which voxels a ray will pass through and as such acceleration structures become necessary for performant visualization.

2.4 Volume Rendering Methods

In this section we will discuss methods of volume visualization both as an isosurface or through DVR. These methods range from purely CPU based to GPU based and from rasterization based to ray tracing based.

2.4.1 Structured Volume Rendering Methods

Levoy first introduced volume rendering using a software ray tracer [7]. This early method was largely brute force in nature and consisted of simply tracing rays into a volume

with a fixed step size and interpolating a value for the sample position from nearby voxels. This method was so computationally expensive that on contemporary machines render times for a 113^3 volume and 256^2 viewing plane were as high as forty minutes per image.

Isosurfaces may first be extracted from a volume data set using an algorithm such as marching cubes [9] or through tree-traversal methods such as those using an octree [24]. Since these methods produce a polygonal mesh the final visualization methods of such implementations do not fall within the scope of this paper. However, it should be noted that creating an explicit isosurface in memory can take significant amounts processing time.

Parker et al. [13] present a distributed shared memory approach to ray tracing an isosurface representation of the "visible woman" data set which is a 910MB 512x512x1734 structured volume. An implicit isosurface is generated at rendering time via trilinear interpolation of the grid cells. As an optimization a three level hierarchical grid was used to allow for empty-space skipping. Through utilization of up to 128 CPUs interactive frame rates were achieved.

Wald et al. [21] introduced a CPU-based method of interactively visualizing implicit isosurfaces for structured volumes using a min-max KDTree. By taking advantage of SIMD processor extensions for both traversal and voxel intersection they were able to achieve interactive frame rates for a 640x480 screen size. By leveraging OpenRT, which their approach was built built for, they were able to achieve roughly linear scaling across a small cluster of nodes.

Knoll et al. [6] developed a CPU-based system using a min-max BVH structures for spatial domain decomposition of structured volumes to produce both isosurface renderings and DVR renderings. Their system also focused on coherent traversal of the MMBVH structure allowing for packets of rays to traverse the tree as a group. Such coherent traversal systems enable high utilization of SIMD lanes in a processor lending to the strong performance of the solution.

2.4.2 Unstructured Volume Rendering Methods

Shirley et al. [17] introduced the projected tetrahedron algorithm for DVR visualization of tetrahedral grids. In this method each tetrahedral cell is be approximated through transparent triangles. Each transparent triangle approximation is then passed off to a GPU for rasterization. The results are not perfectly accurate, however they are visually similar to more exact methods and are able to be rendered significantly faster than contemporary methods.

Marmitt et al. [10] use an interesting approach for direct volume rendering of tetrahedral volumes. First an initial intersection is found using some well known acceleration structure such as a KDtree or BVH. Plücker coordinates are used to determine whether an oriented line passes clockwise or counter clockwise around another oriented line or even whether or not they intersect. After the initial intersection is found the tetrahedral mesh is traversed in Plücker space simplifying mesh traversal down to an average of 2.67 tests on average for each traversal step.

Childs et al. [2] introduced a parallel shared memory algorithm for rendering massive structured and unstructured volume data sets. Their algorithm relies on generating a logically structured grid and rendering it slice by slice to produce the final image. These steps are intended to be distributed across many CPUs in a cluster and as such there is an extra back-to-front compositing step to complete rendering the final image. The algorithm allowed for sample-based volume rendering large data sets in seconds per frame rather than minutes or hours.

Wald et al. [20] worked forward from the techniques in [21] using an MMBVH rather than an MMKDTree. Their approach is limited to tetrahedral meshes and only supports isosurface rendering, but is also expanded to support time varying meshes. The approach also relies on a specialized frustum traversal of the tree which may be difficult to integrate into an existing ray tracing framework such as OSPRay.

Muigg et al. [11] introduced a method of visualizing complex polyhedral grids. Such grids made up of mixed polyhedra previously needed to be transformed into homogeneous

tetrahedral grids prior to visualization. Their approach makes clever use of a two sided face sequence list (TSFSL) to traverse between cells without needing to traverse some external acceleration structure to find the next cell. After determining the exit face the entrance face of the next polyhedra is easily determined by walking the TSFSL. This quick volume traversal allowed for interactive volume visualization of complex polyhedral meshes on contemporary hardware. Although this method as currently implemented makes use of a GPU for ray-casting it would be quite interesting to implement and test its performance on modern CPU hardware to circumvent any bottlenecks due to bus speed or GPU memory size constraints.

Any of the above techniques that can be applied to hexahedral grids may also be applied to regular and rectilinear grids. This is due to the fact that regular and rectilinear grids are simply special cases of hexahedral grid for which *a priori* knowledge is available about their structure. However, applying such methods to the aforementioned structured grid types is likely to yield worse performance than using a specially tailored algorithm.

Chapter 3

Analysis of Unstructured Grids

Scientists often use unstructured polyhedral meshes to represent volume data sets in cases where a structured mesh cannot meet their demands. The key advantages of unstructured meshes are accuracy and variable element size. However to gain these advantages costs are paid in terms of the models themselves becoming more complex. In general we can say that the trade off between unstructured and structured grids is that unstructured grids are more accurate while structured grids are less complex.

Next we will explore these trade offs in further detail from the perspective of a simulation author or scientist generating volume data from physical phenomena. Later we will discuss the impact the changes to mesh structure have on visualization.

3.1 Advantages of Unstructured Volumes

3.1.1 Variable Detail

Simulations may not have uniform information frequency across the entire simulation domain and real physical measurements almost certainly do not. Since a regular grid allows for no variation in cell size or orientation it can be difficult to match a one to physical phenomena or simulations with high cell-to-cell variance in information frequency.

Unstructured grids on the other hand have no true limits on variance in cell size and orientation. Such meshes can simply connect the vertices into a collection of general polyhedral shapes. It should be noted that polyhedra generated through these methods can be concave, which requires special consideration when rendering. Figure **??** illustrates this difference albeit generalized to a two dimensional data set.

Consider a basic point cloud and an intention to represent it as a volume data set. If we attempt to represent it as a structured grid it is highly likely that we will need to resample the data values from their original positions and values into the grid positions and new values. Depending on data irregularity it could be necessary to use a very fine resolution structured grid. Alternatively we could convert the point cloud into a tetrahedral mesh. In this case we can faithfully represent the original data without introducing any error, but we would need to keep track of connectivity information for every cell.

3.1.2 Accuracy

As noted in Section 3.1.1 and Figure **??** unstructured meshes are in the majority of cases a more accurate representation of the original data than structured meshes are. Since a structured mesh will contain the weighted averages of the data points at each grid point some amount of data accuracy will be lost. This loss in accuracy can be mitigated by increasing mesh granularity but there will continue to be some small amount of loss.

Unstructured grids on the other hand have no need to recompute the values from the original positions into new grid positions because the original positions do not need to change. Because the values are not recomputed no accuracy can be lost. Due to this property an unstructured grid can tightly fit any simulated data set and as such can more closely mimic physical phenomena.

3.2 Disadvantages of Unstructured Volumes

3.2.1 Connectivity Information Must Be Stored

Since connectivity information in an unstructured grid is not implicitly defined as it is in structured grids node-to-node connections must be stored explicitly. This causes a significant increase in file storage overhead for unstructured grids. For example a tetrahedral grid will need to store the connectivity information for each individual tetrahedron. Given a mesh containing two tetrahedra and assuming each index value is stored as a 32bit integer we would have 32 bytes of storage overhead. Assuming each vertex is stored as four 32bit floats (one for scalar field value and three for position) we would have a vertex storage cost

of 80 bytes because 3 vertices must be shared between the two tetrahedra for the mesh to be continuous. This is a 40% storage overhead which is no small amount when scientists are already having to make choices between saving fewer simulation time steps or running smaller simulations due to output file size.

It should be noted however that there is storage overhead in a regular grid as well, it is just much harder to quantify. Since a very high granularity mesh may be necessary in order to accurately represent high frequency data it is highly probable that many grid cells throughout a grid made artificially fine to meet such a need could be wasted null values taking up storage. The amount of overhead in this case of course will vary wildly based on factors such as grid granularity and simulation frequency variance.

3.2.2 Irregular Layout

Unstructured grids are highly irregular by their very nature. We cannot make assumptions on the size of the cells making up the grid or their orientations. Worse yet, unlike structured grids we cannot assume that spatial locality implies memory locality; cells that are close spatially may be far apart in memory.

3.3 Impacts on Visualization

We will now discuss the impacts that using an unstructured grid has on visualization when compared to using a structured mesh.

3.3.1 Impacts on Cell Intersection

In the case of a structured mesh cells can be located in constant time for all meshes with the same dimensionality. Given that the granularity of the structured grid is known we can determine which cell a given sample position or ray will pass through with some trivial math. All structured grids can have the cell in which a sample position resides found in constant time by simply scaling the world coordinates for xyz into the model coordinates of the mesh and then determining which cell on that dimension those coordinates correspond to through simple arithmetic.

No such equations exist for cell location in an unstructured grid. As such with no acceleration structures an unstructured grid would have a computational complexity when finding a cell of $O(n)$ where n is the number of cells. If spatial partitioning tree is employed we are able to computational complexity of cell retrieval becomes $O(log_a(n))$ where a is the arity of the tree and n is the cell count.

Given that cell location times increase as mesh complexity increases sampling cells in massive models can become prohibitively expensive. This is especially true as larger viewing planes or higher sampling rates come into play. However, some possible optimizations have been employed to overcome this hurdle, such as packet tracing[20] and schemes which keep track of neighboring cells[11].

It should noted however that many structured grid visualization systems will use an acceleration structure so this is not a particularly huge impact. Structured grid visualization systems often use acceleration structures to facilitate empty space skipping for performance reasons. Acceleration structures used for structured grids however will in general be much more shallow than an acceleration structure used for an unstructured grid since its focus will be only for empty space skipping and not for leaf retrieval. Structured grid systems can afford to build their acceleration structures over relatively large blocks of cells and then simply do brute force calculations within the block taking advantage of high data locality and cache coherency.

3.3.2 Connectivity Information and Ray Tracing

The connectivity information stored by an unstructured grid can be used by a ray tracer to facilitate walking the grid. These methods use the connectivity information to generate lists of shared faces such as the TSFSL data structured used in [11]. Once a list of shared faces is available it is very simple to walk the unstructured grid and avoids repeatedly traversing the acceleration structure.

Cell projection methods such as [17] on the other hand are able to ignore most of the extra connectivity information in the mesh. The only connectivity information needed by cell projection methods is the face definitions. This is due to each polygon being projected

and rasterized independently of the others.

3.3.3 Grid and Cell Concavity

Unstructured volumes may have holes or gaps in the underlying mesh which can cause problems for many visualization algorithms. The main issue with concave meshes in regards to cell projection methods is in visibility ordering. Improper ordering can cause significant rendering artifacts, however mesh convexification algorithms can be used to eliminate such cases[3][15].

Unstructured grids commonly consist partially or entirely of concave polyhedra. Many algorithms, both ray tracing and rasterization based do not support concave polyhedra. As such, concave polyhedra need to be split to decompose them into convex polyhedra.

Chapter 4

Visualizing Unstructured Grids With Min Max BVH

4.1 Method Overview

The following chapters will discuss the our work and findings in [14] Our approach is motivated by the approaches found in [6] and [20]. We extend the two methods to accomplish both isosurface and DVR visualization of unstructured grids using the MMBVH data structure. Our algorithm does not currently handle general polyhedral grids and such an implementation is left as future work.

Our implementation uses two layers of BVH to facilitate integration into the OSPRay rendering framework. This is due to the embree[23] BVH which OSPRay is designed around having no support for extension to an MMBVH.

To circumvent this limitation we build our own MMBVH and have OSPRay treat it as if it were an axis aligned hexahedral surface. If a ray fired by OSPRay intersects the surface, the ray is passed off to an internal intersection handler to do all volume integration. Due to this implementation the highly performant hybrid traversal routines in the embree core are unfortunately not usable and our MMBVH implementation does all of the heavy lifting.

At a high level our algorithm is performed in 7 steps.

1. Read volume data into memory.
2. Build the MMBVH tree over all cells in the grid.
3. Pass the MMBVH as a black-box to OSPRay so that an embree BVH may be built.
4. Cast rays via OSPRay. On a hit within the embree tree intersection and traversal are delegated to our internal MMBVH.
5. Traverse the MMBVH in the appropriate manner for the visualization method (isosurface or DVR).

6. Use the collision information generated in the previous step to generate an accumulation buffer.
7. From the accumulation buffer generate a frame.

Steps one through three only need to be performed once per run of the software and as such are not factored into render speed measurements. Steps four through seven on the other hand are performed for each frame and are included in our frame timings.

4.2 The Min-Max BVH

The MMBVH structure is the linchpin of our visualization algorithm. Our MMBVH implementation is similar to those found in [20] and [6], however we create our own traversal kernels for both isosurface and DVR visualization.

Our binary MMBVH has a worst case cell intersection time of $O(n)$ due to bounding volume overlap. In the worst case sibling MMBVH nodes would overlap 100% of the time and in such a case a ray would be forced to test against all MMBVH nodes. However, in practice our MMBVH has a height of $\theta(log_2(n))$ and little cell overlap and as such intersection scales similarly to the expected scaling of a BST of the same height at $O(log_2(n))$.

4.2.1 Constructing the Min-Max BVH

The MMBVH is built over *all cells* of the polyhedral grid. Building over all cells allows us to in the case of isosurface visualization dynamically change the rendered isovalue or render multiple isovalues from the same volume without reconstructing the MMBVH. In cases such as AMR data where multi-part data sets are common we build an individual MMBVH for each piece as well as a parent MMBVH containing all pieces.

Due to MMBVH build times not being the focus of our work and the fact that their construction is a one time cost we chose to implement our MMBVH builder as a simple scalar single threaded process. Although build times with such an implementation are less than stellar they are acceptable for our purposes. However it should be noted that as cell count increases build times do begin to become prohibitive as exhibited by the MMBVH build for the earthquake data set taking minutes to complete. Even though the MMBVH

is built with purely scalar code the memory layout is SIMD friendly and so it is it is used natively by our ISPC traversal and intersection code.

The MMBVH is constructed using both a SAH[4] and a splitting criterion. The SAH test simply states that a node will be split only if the total surface area of the two newly created nodes will be less than the surface area of the currently existing node. Our splitting criterion is that no leaf node will contain more than seven cells. As such a cell will be split even if the SAH test fails if it will contain more than seven grid cells.

4.2.2 Memory Layout

Since we do not guarantee that our MMBVH will be balanced we cannot use the common array storage scheme for binary trees in which a node stored at array index i can find its children at indices i and i+1 without wasting potentially significant amounts of memory space. Instead while still storing our nodes as a single contiguous array we make use of a reference value to locate the next node. We guarantee that sibling nodes will be stored contiguously to one another and as such only need to store a single reference because we can easily jump to the next child from the first. A single MMBVH node contains upper and lower bound information for the volume as well as upper and lower bound information for the scalar field and finally a child count and reference to the left child. The ISPC implementation of the MMBVH node structure can be seen in Figure 4.1.

Our MMBVH is built around a single scalar field. Volumes in which multiple scalar fields exist will necessitate construction of a unique MMBVH for each scalar field and each field would effectively be treated as its own OSPRay `Volume`.

The `prim_count` value is used to differentiate intermediate and leaf nodes. A node with `prim_count` of 0 is considered to be an intermediate node and a node with any other `prim_count` is a leaf node. We do not need to store a special count for children of intermediate nodes because we guarantee that any intermediate node will always have two children. Since we dedicate only 3 bits of storage to `prim_count` a leaf may only reference up to 7 primitives.

The `ref` value is dual purpose. If the node is designated as a leaf node `ref` will be

a reference from the beginning of the array of primitives and from `prim_count` we will know how many primitives we may read ahead from that index without issue. If the node is designated as an intermediate node `ref` will be a reference from the beginning of the MMBVH array to the first child node.

```
struct MinMaxBVH2Node {
  float bounds_low[3];
  float scalar_range_low;
  float bounds_high[3];
  float scalar_range_high;
  int64 prim_count : 3;
  int64 ref        : 61;
};
```

Figure 4.1: Data Layout of an MMBVH node.

4.2.3 Handling Time Varying Meshes

There are three obvious avenues for handling time varying data sets.

1. Rebuild the MMBVH each time the scalar field changes.
2. Update the portions of the MMBVH that change along with the scalar field changes.
3. Build a separate MMBVH for each time step at start-up.

Due to our scalar single threaded MMBVH build option 1 is not viable. Even the smallest data sets we work with take a time equivalent to that of rendering many frames. As such, without improving our MMBVH build times we cannot use this option.

Option 2, while attractive in that at a glance it seems to do the least amount of actual work is often more expensive than option 1. Since we may need to move entire sub-trees around in memory in unpredictable ways the cost is often very high.

Option 3, while at a glance is the most expensive also offers the best runtime performance. We simply pay the computation costs at once during start-up and never again during runtime. We do pay a cost in keeping extra trees in memory, but this can be mitigated by paging non-adjacent time steps out of RAM.

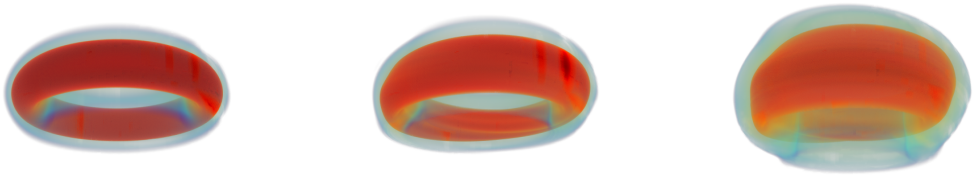


Figure 4.2: Three time steps of the "Fusion" data set visualized using our DVR renderer. Each time step is represented by its own MMBVH and switching between them is handled with a simple pointer swap by the viewer.

Although the cost of storing many trees can be quite high we still choose option 3. This option is the best tailored to our goals as an interactive system and given that unlike GPU based implementations we have easy access to all system memory available we prioritize freeing up processor clock cycles during interaction over memory usage.

4.3 Implicit Isosurface Ray Tracing

Bucky Ball (Bucky) (177k tetrahedra) | Turbulent Jet (T-jet) (1M tetrahedra) | Jets (12M tetrahedra) | San Francisco Bay (SF1) (14M tetrahedra) | Japan Earthquake (47M hexahedra)

Figure 4.3: The five data sets used in our experiments (in increasing geometric complexity), in this figure rendered as isosurfaces using an ambient occlusion renderer.

As the name suggests; in implicit isosurface visualization we never actually create an explicit isosurface in memory. Instead we visualize a virtual isosurface as an artifact of MMBVH traversal and cell intersection.

4.3.1 Tree Traversal

Traversal of the MMBVH structure is implemented with ISPC to take full advantage of all vector units available in the target hardware. To accomplish this we create mirror

implementations of the C++ versions of each structure necessary for operating on the MMBVH. No actual copies of the structures are made, pointers are simply passed from one execution environment to the other. We evaluate two potential methods of MMBVH traversal both of which follow the general structure found in Algorithm 4.1.

Algorithm 4.1: Implicit Isosurface MMBVH Traversal

```
node = root;
if isLeaf(node) then
    for each primitive p do
        intersect(p, ray.t)
    end
    return ray
else
    Ray a = ray; Ray b = ray
    if inRange(iso, node.left.isoRange) then
        traverse(a, node.left)
    end
    if inRange(iso, node.right.isoRange) then
        traverse(b, node.right)
    end
    if a.t < b.t then
        return a
    else
        return b
    end
end
```

The first potential method is pure SPMD traversal. In this method each program instance traverses the MMBVH independently and keeps a program-instance-unique traversal stack. The SPMD algorithm is trivial to implement in ISPC by simply giving all traversal state variables the *varying* storage qualifier which allows each SIMD lane to perform its own independent traversal.

Data Set	Size	SPMD	Packet	Gain
Bucky	177k tets	68.9	103.2	50%
Jets	1M tets	79.9	92.0	15%
T-jet	12M tets	90.7	109.4	21%
SF1	14M tets	60.2	78.7	31%
Earthquake	47M hexes	14.1	46.3	228%

Table 4.1: A comparison of both traversal algorithms on the *Xeon Node* (specifications in Chapter 5). Measurements are for primary rays only.

The second method Packet Traversal [22] groups rays into packets of size equal to the width of the SIMD lanes of the target machine. ISAs supporting AVX/AVX2 will create packets of eight rays while SSE machines will produce packets of four rays. When traversing as a packet all rays must be attempting intersection of the same bounding volume or cell in unison. The packet traversal algorithm follows the same basic structure as the SPMD algorithm, however all traversal state tracking variables use uniform types and must be treated as if they were memory shared between multiple concurrent threads of execution. If a ray has terminated earlier than its peers either by exiting the volume or successfully finding the closest possible intersection along that ray for the isovalue of interest that SIMD lane will still continue to traverse with the group, but all new calculation results will be thrown away. Our implementation of packet traversal also makes use of the concept of speculative decent as seen in the motivating works. If any ray in a packet is found to need to descend the tree *all* companion rays will also descend the tree.

Our methodology for comparing the two methods was to collect performance metrics in terms of millions of rays per second for each of our test data sets and to then compare their average value. We also took into account how well each algorithm scaled as cell count and tree complexity increased. These figures can be seen in Table 4.1.

Through our measurements we observed that the packet traversal algorithm is more performant with all of our test data sets and scales significantly better as well. In particular packet traversal achieves the highest performance increase with our largest data set. The significant increase in percentage speedup from packet traversal of the largest data set is that the scalar overhead of our traversal algorithm is dwarfed by the amount of traversal

computations necessary for such a large tree. From these results we elected to implement the packet algorithm as our traversal method for all future measurements.

4.3.2 Empty Space Skipping

Since our MMBVH is built over all cells of the unstructured grid it stands to reason that some cells and by some extension some sub-trees of the MMBVH will not contain the isovalue of interest. Such subs-trees can be considered to be empty space for the purposes of visualizing the appropriate isosurface. As an optimization we delay our ray-box intersection tests such that they are only executed if the isovalue for the current surface is within the min and max range of the sub-tree represented by the current MMBVH node. If the sub-tree cannot contain the isovalue we simply skip that entire sub-tree. This allows us to dynamically skip potentially large areas of the grid and is often a significant performance gain.

4.3.3 Isosurface Intersection

When intersecting a cell we calculate an intersection point with the virtual surface through the use of the Neubaur method [12]. We first test that the ray intersects a given cell, generating t_{in} and t_{out} in the process. We then apply Neubauer's method with N=2 to generate t_{hit} through successively subdividing the line segment intersecting the cell and interpolating along it. Pseudo-code for the algorithm can be found in Algorithm 4.2. This algorithm is repeated for all cells with potentially contain the appropriate isosurface and the closest hit (smallest t_{hit}) is returned to the OSPRay renderer after a shading normal is generated.

4.3.4 Shading

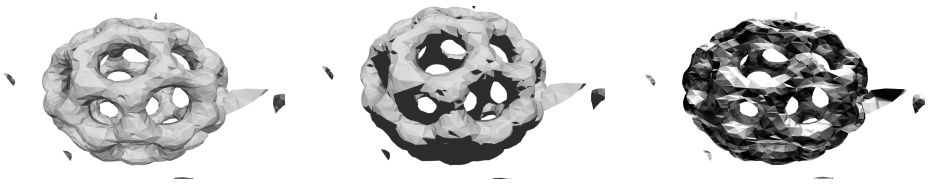
$$\delta_h[f](x) = f(x + \tfrac{1}{2}h) - f(x - \tfrac{1}{2}h)$$

Figure 4.4: The Central Difference Equation

Algorithm 4.2: Surface Intersection Algorithm For Implicit Isosurfaces

```
t_in = inf; t_out = -inf;
for each cell plane do
    t_phit = intersect(ray, plane)
    t_in = min(t_phit, t_in) t_out = max(t_phit, t_out)
end
if t_in > t_out then return NO_HIT;
t_0 = t_in; t_1 = t_out; v_0 = interpolate(ray, cell, t_0); v_1 = interpolate(ray, cell, t_1);
for i=1..N do
    t = t_0 + (t_1 - t_0) (iso - v_0) / (v_1 - v_0)
    if sign(interpolate(ray, cell, t)) == sign(v_0 - iso) then
        t_0 = t; v_0 = interpolate(ray, cell, t)
    else
        t_1 = t; v_1 = interpolate(ray, cell, t)
    end
end
t_hit = t_0 + (t_1 - t_0) (iso - v_0) / (v_1 - v_0)
return t_hit
```

If an intersection with the virtual isosurface is found we use the central differences method (See Figure 4.4) on each dimension to approximate a surface normal. The surface normal generated is used by the pre-implemented OSPRay renderer currently in use for shading. No modifications to the existing renderers were necessary for our methods. Examples of different renderers and shading models being applied to the same isosurface with the same light and camera can be seen in Figure 4.5



(a) (b) (c)

Figure 4.5: The bucky ball volume data set rendered with (a) The OSPRay OBJ renderer without shadows, (b) The OSPRay OBJ renderer with shadows, (c) The OSPRay ambient occlusion renderer with 16 AO samples.

4.3.5 Multiple Isosurfaces From a Single Volume

When rendering multiple implicit isosurfaces from a single volume we perform traversal for all isovalues of interest in unison. At each traversal step where an isovalue needs to be considered we simply consider all isovalues of interest in round robin fashion. In the event that any intersections are found we return the intersection with the smallest t_{hit} value to the renderer using the isovalue index in the round-robin queue as a surface ID. It is then the renderer's decision whether the ray should continue traversing the tree (as in the case of a transparent isosurface) or if traversal should terminate (opaque isosurface or the renderer has decided that any further contribution from the ray would be negligible). A rendering of the jets data set with multiple isosurfaces can be seen in Figure 1.1(a).

4.3.6 Dynamic Isosurface Rendering

Building our MMBVH over all cells in the grid implies that we can represent all possible isosurfaces for a given scalar field with a single MMBVH. As such we can simply change the isovalue of interest at runtime to produce a different isosurface. There is no appreciable performance impact beyond extra memory use in allowing this due to the ability to skip entire sub-trees if the current isovalue cannot exist in that sub-tree as shown in Section 4.3.2. An example of three different isosurfaces all generated from the T-jet can be seen in Figure 4.6.

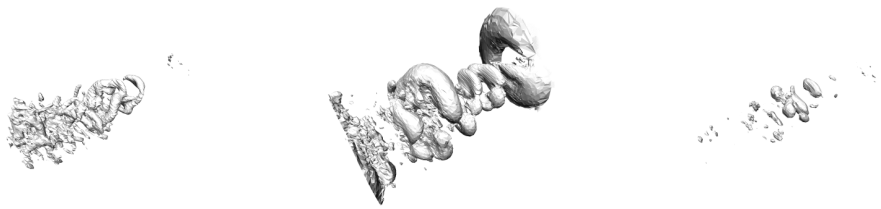
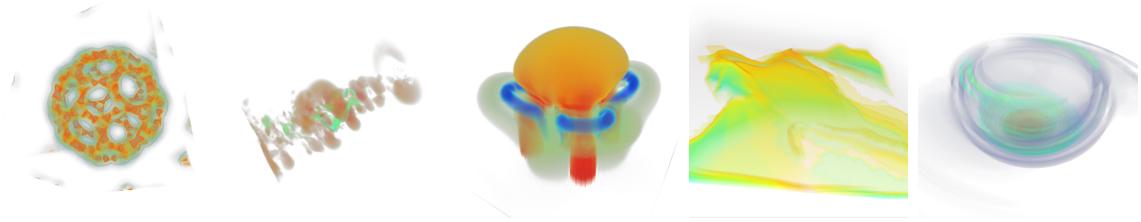


Figure 4.6: T-jet rendered with isovalues of (a) -0.0390374, (b) 0.00290815, (c) 0.0364645



Bucky Ball (Bucky) (177k tetrahedra) | Turbulent Jet (T-jet) (1M tetrahedra) | Jets (12M tetrahedra) | San Francisco Bay (SF1) (14M tetrahedra) | Japan Earthquake (47M hexahedra)

Figure 4.7: The same data sets as in Figure 4.3, this time rendered with direct volume ray casting using appropriate transfer functions.

4.4 Direct Volume Ray Casting

We implement a sample-based visualization algorithm for unstructured grids using the same MMBVH structure that is used for implicit isosurface visualization. Although the data structures used remain the same the way we act on them changes somewhat drastically. Through this we extend the existing OSPRay DVR visualization capabilities to include visualization of homogeneous unstructured tetrahedral or hexahedral grids.

4.4.1 Tree Traversal

Traversal of the MMBVH for sample-based DVR is still packet based as in the case of implicit isosurface visualization. However, given a packet of rays we generate multiple sample position packets (SPP). Each SPP traverses the tree with the same speculative decent rules as in the case of isosurface visualization, however instead of ray-box intersection tests we are performing point-in-box tests for each bounding volume.

In an attempt to minimize the number of tree traversal operations due to sampling we extended our concept of SPP to include multiple samples of each ray. In this case we group N sample positions from each ray in a packet. A packet of M rays each with N sample positions will have NM samples in the SPP. In this case we still speculatively descend if *any* sample in the SPP needs to descend the tree.

The performance impact of this change was highly dependent on the sample group size. Sample groups that were too large caused performance drops. We speculate that these performance drops were likely due to samples within the same group existing in different

	Group Size					
Data Set	1	4	8	16	24	32
Bucky	54.6	**56.5**	56.0	54.9	52.4	49.6
Jets	5.2	5.2	**5.4**	5.3	5.2	5.1
T-jet	4.0	4.4	**4.5**	**4.5**	4.4	4.3
SF1	**9.9**	9.6	9.5	9.0	8.7	8.4
Earthquake	12	13.7	15.9	17.7	**17.8**	17.3
Average difference from baseline across data sets						
	0	+.74	+1.12	**+1.14**	+.56	-.2

Table 4.2: A comparison of different sample group sizes for each data set as measured on our *Xeon Node*. Values are in millions of rays per second. The values in bold represent the highest achieved for the data set.

sub-trees. It is also notable that optimal SPP width varied on a per volume basis. However a general trend can be seen in that for most models an SPP width greater than 1 was optimal. We chose for this work to choose the SPP width that produced the best results on average across our suite of test volumes yielding an SPP width of 16 and an average increase of 1.14 million rays per second. Our measurements can be seen in Table 4.2.

We also found an interesting trend in the relation between the fixed step size used and SPP width. Step size and SPP width seem to have a strictly inverse relationship with regards to performance. As SPP width increases we must reduce step size or we will suffer performance losses. This is also likely due to the pairing of a wide SPP and large step length causing samples in the same group to be located in different sub-trees of the MMBVH Our measurements exhibiting this relation can be found in Table 4.3.

4.4.2 Cell Sampling

Cell sampling is handled in a relatively simple manner. We determine if sample P exists within a cell by applying the signed planar distance method to each face F of the polyhedron. If the sign of the distance for all faces of the polyhedron are the same P exists within the polyhedral cell. This method is simple, reasonably efficient and can be produces correct results for all polyhedra without holes. Finally, given that P exists within the cell we calculate the sample value at P by trilinear interpolation from the scalar field values constituent to the cell.

Step size	Group Size 1	4	8	16	24	32
2.5	2.4	2.9	3.1	3.2	**3.3**	**3.3**
5	3.5	3.9	**4.1**	**4.1**	**4.1**	4.0
7.5	4.5	4.6	**4.7**	**4.7**	**4.7**	4.5
10	5.2	5.3	**5.4**	5.3	5.2	4.9
12.5	5.9	**6.0**	**6.0**	5.8	5.7	5.5
15	**6.5**	**6.5**	**6.5**	6.3	6.1	6.0

Table 4.3: Performance scaling of sample group size against ray marching distance on the *Xeon Node*. All measurements are in millions of rays per second with the Jets data set. The values in bold are the highest throughput achieved for the step size. For reference, the ray marching distance used for the Jets data set in Table 4.2 was 10.

$$D = N^F \cdot (\{P_x, P_y, P_z\} - \{C_x^F, C_y^F, C_z^F\})$$

Figure 4.8: Signed point-to-plane distance formula. Note that C^F is the center point of the plane on which Face F exists and N^F is the normal from that same plane.

4.4.3 Frame Accumulation

Unfortunately since we generate samples along each ray at a fixed interval this can exhibit pronounced wave front like artifacts (See Figure 4.9 for an example of this artifact). To alleviate this artifact we take advantage of an accumulation buffer. Each frame the starting sample position is jittered randomly between the eye position and the point at our fixed sample distance along the ray. In this way as the image resolves we manage to stochastically sample along the ray producing high quality images while maintaining an interactive environment.

4.4.4 Adaptive Empty Space Skipping

By virtue of our acceleration structure we gain for free the ability to skip what we term as *static empty spaces*. A *static empty space* is an empty space which will exist for all renderings of a given volume and are a feature of the grid topology.

A *dynamic empty space* is an empty space which is created at runtime due to interaction with a user. In our case the only method to create a dynamic empty space is to adjust the

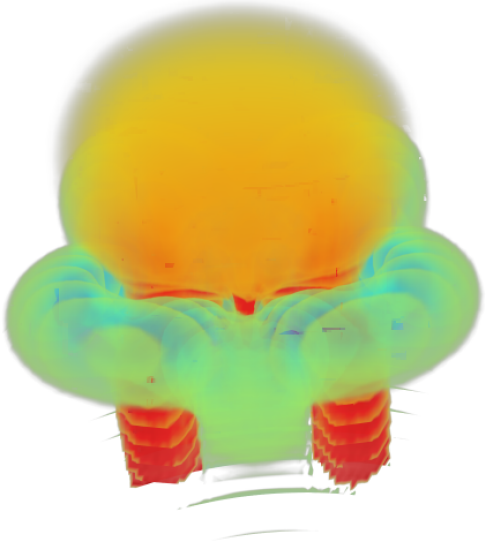


Figure 4.9: The wave front artifact initially exhibited by our DVR algorithm.

transfer function used to assign RGBA values to each sample. Examples of dynamic empty spaces created through transfer function manipulation can be seen in Figure 2.2.

These empty spaces can be trivially skipped similarly to how empty spaces are skipped in our isosurface visualization method. As we traverse the tree we check at each attempted descent whether or not *all potential sample values* in the sub-tree will evaluate to an RGBA value with an opacity of 0. If the sub-tree will always have an opacity of 0 for the current transfer function it will be skipped since it is empty space for our purposes.

Chapter 5

Results and Discussion

5.1 Test Machine Descriptions

Our two test machines are as follows:

Machine Name	Xeon Node	Phi Node
CPU	2x Intel®Xeon®E5-2687W v3	2x Intel®Xeon®E5-2680
CPU Clock	3.1GHz	2.7GHz
Memory	128GB	46GB
Coprocessor	N/A	3x Intel®Xeon Phi™7120A

Table 5.1: Specifications for the two test machines used in our experiments.

5.2 Collected Results

All performance results were measured when rendering at a resolution of 1080p. All results were measured by producing the same images on both machines. All measured images can be found in the section of their respective scene. Unless otherwise noted all values are in millions of primary rays per second.

For results pertaining to the *Phi Node* all measurements were taken using only the coprocessors as execution units. The main CPUs were used only to interact with the coprocessors. *Phi Node* results also use 100 samples per pixel per frame as opposed to a single sample per pixel per frame for the *Xeon Node*.

Distributed rendering in an MPI parallel execution environment is possible and has been tested and shown to be working. However this execution environment was not included in our performance testing due to the MPI environment in OSPRay not being ready for rigorous use at the time of this research.

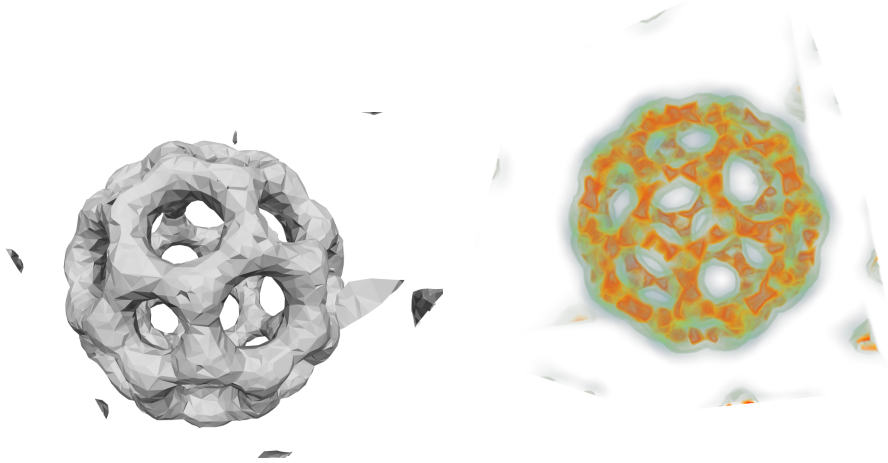
Data Set	Size	Xeon Node		Phi Node	
		ISO	DVR	ISO	DVR
Bucky	177k tets	103.2	54.9	365.0	25.3
Jets	1M tets	92.0	5.3	282.0	9.7
T-jet	12M tets	109.4	4.5	418.9	9.8
SF1	14M tets	78.7	9.0	190.8	12.1
Earthquake	47M hexes	46.3	17.7	101.6	16.7

Table 5.2: Performance measurements for our two test machines in both isosurface visualization and DVR visualization.

5.3 Buckyball Scene

At 177k tets the buckyball is by far our simplest test scene. For this simple scene we are able to perform well beyond our performance targets for interactivity. In fact, for lower resolutions such at 1024^2 we actually achieve *real time* levels of performance on both of our test machines for both visualization methods.

Given that the model is so simple we are able to generate a tree with minimal bounding volume overlap. Due to this tree traversal overhead is kept minimal which is a large boon to performance for both visualization methods.

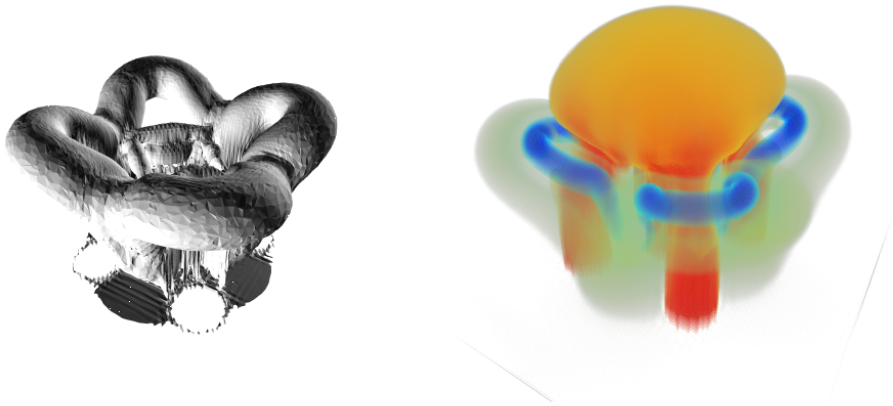


(a) (b)

Figure 5.1: The buckyball data set rendered (a) as an isosurface using a 16 sample ambient occlusion renderer, (b) Via DVR

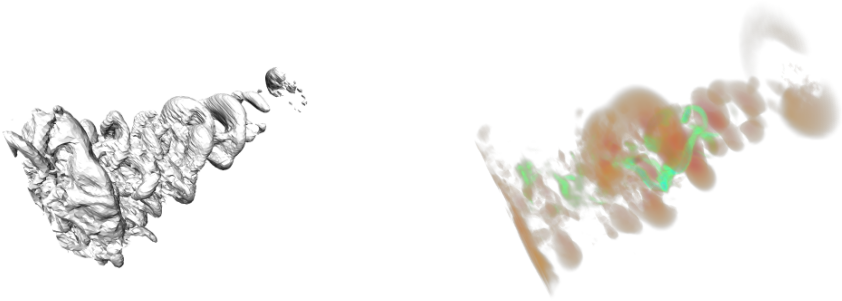
5.4 Jets & T-jet Scene

The Jets and T-jet data set were more problematic. We were able to achieve real time performance for isosurface rendering yet again. However DVR performance dipped down into the range of simply being interactive at 1080p.



(a) (b)

Figure 5.2: The jets data set rendered (a) as an isosurface using a 16 sample ambient occlusion renderer, (b) Via DVR



(a) (b)

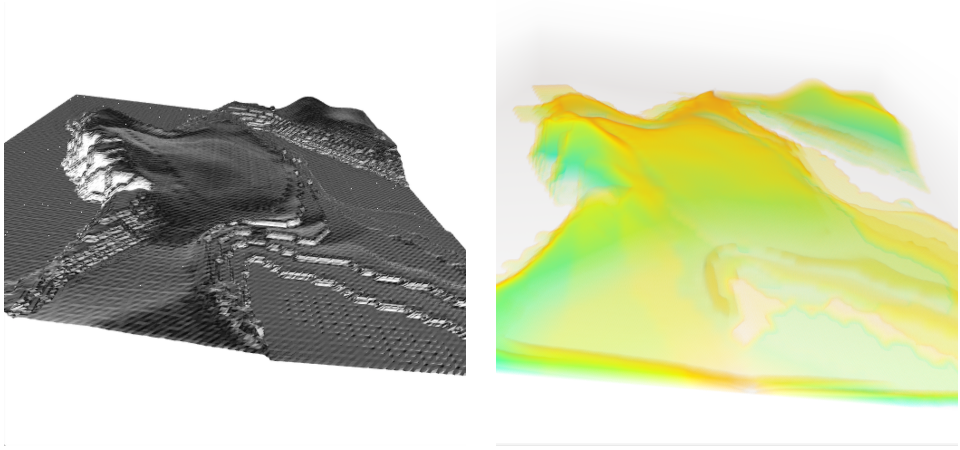
Figure 5.3: The T-jet data set rendered (a) as an isosurface using a 16 sample ambient occlusion renderer, (b) Via DVR

We observed that the MMBVH built by our splitting criterion exhibited high levels of sibling overlap. As a result of this both visualization methods suffer significant performance

loss. However, because our DVR visualization stresses tree traversal much more intensely a badly formed tree impacted performance far more drastically.

5.5 SF1 Scene

Performance for the SF1 scene was still well within the bounds of real time performance for isosurface rendering. We also performed quite well under DVR visualization despite SF1 being a scene that actually suffers due to our usage of wide SPPs. SF1 likely suffers a performance loss due to wide SPPs due to the fact that the data set itself is rather thin; it is highly likely that we often performing point in box tests for sample positions that are well outside the bounds of the volume.



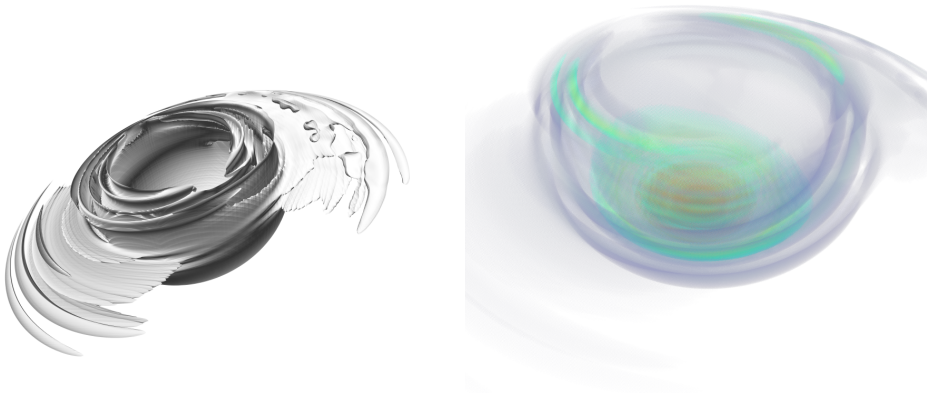
(a) (b)

Figure 5.4: The jets data set rendered (a) as an isosurface using a 16 sample ambient occlusion renderer, (b) Via DVR

5.6 Earthquake Scene

We continued to achieve realtime performance for isosurface visualization on both of our test machines with our largest and only hexahedral data set. The earthquake data set is nearly an ideal example of a complex data set that also lends itself well to our methods due to our ability to construct a very low overlap MMBVH for it.

The earthquake data set also performs quite well under DVR. We achieve 17.7 and 16.7 million rays per second for our Xeon and Phi nodes respectively. We attribute this again



(a) (b)

Figure 5.5: The jets data set rendered (a) as an isosurface using a 16 sample ambient occlusion renderer, (b) Via DVR

to the earthquake data set lending itself to build a tree with low overlap as well as having a fairly well balanced cell size across all cells in the grid, which allowed us to tune the sampling step size to the SPP width very easily.

5.7 CPU vs Coprocessor

The Phi Node outperforms the Xeon node quite well for isosurface rendering, which is well within expectations considering the amount of compute horsepower that the Phi Node has at its disposal. The only thing working against the Phi Node with regards to isosurface rendering is their limited memory. Unlike system memory which can be increased even into the terabyte scale the coprocessors are limited to their on board memory. Large enough data sets will begin to suffer from needing to repeatedly shuffle data across the PCI-Express bus in which case the Xeon Node would begin to outperform the Phi Node.

With regards to DVR performance the Phi Node did not meet initial expectations. Upon further analysis we conclude that our algorithm is simply not well suited to the hardware design which struggles with algorithms that jump around the memory space frequently or have large amounts of branching. This is due to design decisions made for the coprocessors themselves; they are simply not designed for this particular task. An algorithm designed with their architecture in mind from the beginning would likely perform quite well, but

would suffer from the same eventual pitfalls due to data set size as mentioned for the isosurface visualization method.

5.8 Comparison to other CPU based methods

Similarly to other implicit isosurfacing alternatives our method does not require extensive pre-computation of the volume data to extract an isosurface. Isosurfaces are generated on the fly and never exist explicitly in memory.

Unlike other implementations our method enables visualization of an isosurface or a DVR volume using a single common data structure. This makes it feasible to rapidly and dynamically swap between volume and isosurface visualization during run time as well as to mix isosurface and volume visualization of the same data set with much lower memory overhead.

5.9 Comparison to GPU based methods

Unlike the popular projected tetrahedra technique we do not use an approximation. Rather than splatting polygons that are roughly similar to the shapes of the cells making up the data set we directly sample the data for each pixel.

Although similar methods have been developed for GPUs they still suffer from the inherent constraints of current GPU technology. GPU memory is orders of magnitude lower than the maximum available host memory on almost nay given system. GPUs are also not available on all systems and as such cannot be used on general compute nodes while ours requires no specialized hardware whatsoever.

Chapter 6

Conclusions

6.1 Conclusions

Our work is a direct continuation to the research found in [6] and [20]. We have presented a technique for interactive visualization of both isosurface and DVR visualization of homogeneous unstructured grids. We used packet traversal of a MMBVH in a ray tracer implemented with an SPMD compiler to achieve high performance across a wide range of CPUs and Intel Xeon Phi coprocessors while avoiding the need to write specialized code for each platform. We find our method to perform interactively and with minimal memory overhead on general compute nodes without the need for a GPU.

The method as presented performs well for isosurface rendering and can also be used for direct volume ray tracing without the need for separate data structures. Although initial performance for DVR visualization was lower than desired through clever use of progressive refinement we are able to tolerate very high step sizes while still producing high quality output.

6.2 Future Work

6.2.1 In-situ visualization

Due to our method being capable of working on the unmodified unstructured grid cells it is well suited to eventual incorporation into an *in-situ* visualization routine. With further effort MMBVH build times could be drastically improved which would allow for quick visualization of simulation results while the simulation is running.

6.2.2 Hybrid Traversal Methods and Other Acceleration Structures

We strongly believe that hybrid traversal methods as found in embree[23] could significantly improve performance of our approach. It is left as future work to explore patching embree or developing our own similar library for the MMBVH.

It is also desirable to explore adapting other acceleration structures to take advantage of the properties of a min-max tree. The properties of some other spatial acceleration structures could be highly desirable.

6.2.3 Further Exploration of wide SPPs

Wide SPPs were a very late addition to our method and as such did not receive quite the same level of polish as the rest of the algorithm. It is left as future work to explore improvements such as adaptive scaling of SPP width to improve performance as well as exploration of the impact of the number of rays in a packet.

Appendix A

Glossary of Terms

- Bounding Volume Hierarchy - A tree of nested bounding volumes used for spatial subdivision.
- Isosurface - A surface generated from a volume data set for which all points along the surface have the same isovalue.
- Isovalue - The value of a scalar field for which an isosurface is generated.
- KDTree - K Dimensional Tree used for spatial subdivision. Space is split along an axis at each tree level. The chosen axis can be picked randomly, in a round robin order, or through the use of heuristics.
- Min-Max Tree - A Min-Max tree is a special form of any other spatial tree in which the minimum and maximum values contained in the sub-tree of a node are stored as part of it.
- OSPRay - An Intel developed open source ray tracing frame work.
- Transfer function - A function through which a scalar value is translated into an RGBA value.
- Voxel - Voxel is a portmanteau of volume and pixel. As this suggests, it is the basic unit of representation in a 3D grid. A voxel is a single cell in a 3D grid.
- Packet Tracing - Packet tracing is the act of ray tracing a scene using groups of rays which will all consider intersection with the same object at the same time. In such schemes ray coherence is of great importance.
- Plücker Coordinates - Plücker coordinates are used to specify directed lines in three dimensional space.
- Plücker Space - A geometric space defined by Plücker coordinates.

Bibliography

[1] OSPRay Ray Tracing Framework. http://www.ospray.org, 2015. Last Accessed: 2015-07-15.

[2] Hank Childs, Mark Duchaineau, and Kwan-Liu Ma. A Scalable, Hybrid Scheme for Volume Rendering Massive Data Sets. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '06, pages 153–161, 2006.

[3] JooL.D. Comba, JosephS.B. Mitchell, and CludioT. Silva. On the convexification of unstructured grids from a scientific visualization perspective. In Georges-Pierre Bonneau, Thomas Ertl, and GregoryM. Nielson, editors, *Scientific Visualization: The Visual Extraction of Knowledge from Data*, Mathematics and Visualization, pages 17–34. Springer Berlin Heidelberg, 2006.

[4] Martin Eisemann, Thorsten Grosch, Marcus Magnor, and Stefan Mller. Automatic Creation of Object Hierarchies for Ray Tracing Dynamic Scenes. In *IN WSCG SHORT PAPERS PROCEEDINGS*, 2007.

[5] Fan Guo, Hui Li, William Daughton, and Yi-Hsin Liu. Formation of hard power laws in the energetic particle spectra resulting from relativistic magnetic reconnection. *Phys. Rev. Lett.*, 113:155005, Oct 2014.

[6] Aaron Knoll, Sebastian Thelen, Ingo Wald, Charles D. Hansen, Hans Hagen, and Michael E. Papka. Full-resolution Interactive CPU Volume Rendering with Coherent BVH Traversal. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium*, pages 3–10, 2011.

[7] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, May 1988.

[8] Luiz Alberto Lima and Rui Bastos. Seismic data volume rendering. Technical report, 1998.

[9] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[10] Gerd Marmitt and Philipp Slusallek. Fast Ray Traversal of Tetrahedral and Hexahedral Meshes for Direct Volume Rendering. In *Eurographics/IEEE-VGTC Symposium on Visualization (EuroVIS)*, pages 235–242, 2006.

[11] Philipp Muigg, Markus Hadwiger, Helmut Doleisch, and Eduard Gröller. Interactive Volume Visualization of General Polyhedral Grids. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2115–2124, 2011.

[12] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. Cell-based First-hit Ray Casting. In *Proceedings of the Symposium on Data Visualisation 2002*, VISSYM '02, pages 77–ff, 2002.

[13] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive Ray Tracing for Isosurface Rendering. In *Proceedings of the Conference on Visualization '98*, VIS '98, pages 233–238, 1998.

[14] Brad Rathke, Ingo Wald, Kenneth Chiu, and Carson Brownlee. SIMD Parallel Ray Tracing of Homogeneous Polyhedral Grids. In C. Dachsbacher and P. Navrtil, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2015.

[15] Stefan Röttger, Stefan Guthe, Andreas Schieber, and Thomas Ertl. Convexification of unstructured grids. In *Proc. Vision, Modeling and Visualization (VMV) 2004*, pages 283–292, November 2004.

[16] G D Rubin, C F Beaulieu, V Argiro, H Ringl, A M Norbash, J F Feller, M D Dake, R B Jeffrey, and S Napel. Perspective volume rendering of ct and mr images: applications for endoscopic imaging. *Radiology*, 199(2):321–330, 1996. PMID: 8668772.

[17] Peter Shirley and Allan Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Proceedings of the 1990 Workshop on Volume Visualization*, VVS '90, pages 63–70, 1990.

[18] E. Steen and B. Olstad. Volume rendering of 3d medical ultrasound data using direct feature mapping. *Medical Imaging, IEEE Transactions on*, 13(3):517–525, Sep 1994.

[19] Daniel J. Valentino, John C. Mazziotta, and H.K. Huang. Volume rendering of multimodal images: application to mri and pet imaging of the human brain. *Medical Imaging, IEEE Transactions on*, 10(4):554–562, Dec 1991.

[20] Ingo Wald, Heiko Friedrich, Aaron Knoll, and Charles D Hansen. Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes. Technical Report UUSCI-2007-003, SCI Institute, University of Utah, 2007. (conditionally accepted at IEEE Visualization 2007).

[21] Ingo Wald, Heiko Friedrich, Gerd Marmitt, Philipp Slusallek, and Hans-Peter Seidel. Faster Isosurface Ray Tracing using Implicit KD-Trees. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):562–573, 2005.

[22] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001. (Proceedings of Eurographics 2001).

[23] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree - A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 33, 2014.

[24] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, July 1992.