An Analysis of Region Clustered BVH Volume Rendering on GPU

D. Ganter[†] and M. Manzke[‡]

School of Computer Science and Statistics, Trinity College Dublin, Ireland

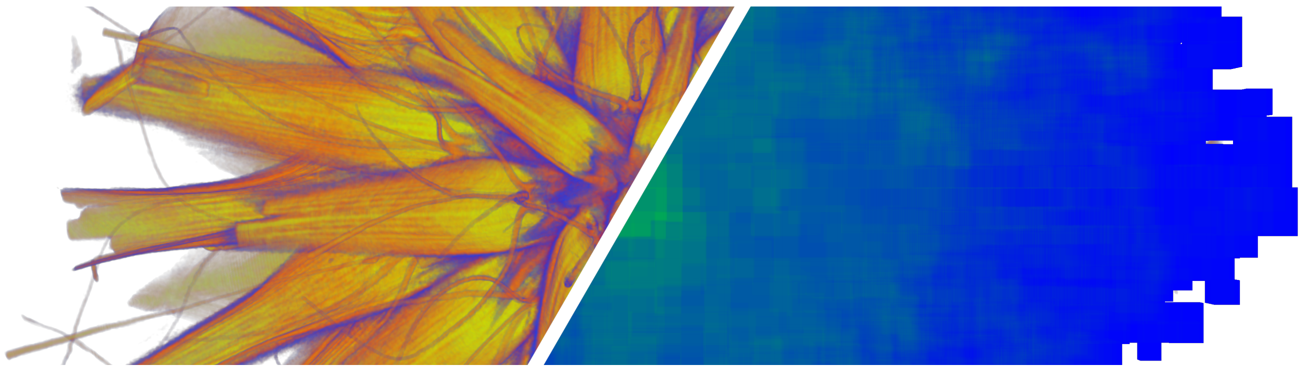

Figure 1: *Example volume rendering of a Flower μCT scan (left) and an indicative heat-map showing the depth-complexity bounding-volume-hierarchy leaf nodes (right) for one particular transfer function, with early-ray-termination disabled to visualise the geometry.*

Abstract

We present a Direct Volume Rendering method that makes use of newly available Nvidia graphics hardware for Bounding Volume Hierarchies. Using BVHs for DVR has been overlooked in recent research due to build times potentially impeding interactive rates. We indicate that this is not necessarily the case, especially when a clustering algorithm is applied before the BVH build to reduce leaf-node complexity. Our results show substantial render time improvements for full-resolution DVR on GPU in comparison to a recent state-of-the-art approach for empty-space-skipping. Furthermore, the use of a BVH for DVR allows seamless integration into popular surface-based path-tracing technologies like Nvidia's OptiX.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ *Computer graphics; Scientific visualization;*

1. Introduction

In almost all recent GPU Direct Volume Rendering (DVR) applications, octrees in some way or form are the prevalent acceleration data-structures used. There are clear benefits that octrees provide to DVR; easy-to-implement empty-space-skipping (ESS), clear and defined volume paging and caching, and trivially sub-sampled data support are to name but a few. Octrees have long been a natural acceleration data-structure for volumes on GPU thanks to the clear, defined and easy to implement subdivision pattern, with predictable traversal times and well researched algorithms.

Bounding Volume Hierarchies (BVHs) have also been investigated as a form of acceleration for DVR, however most of this research has been geared toward CPUs rather than GPUs. BVHs are designed to handle regions of varying size by using Axis-Aligned Bounding Boxs (AABBs) to spatially group surface data, like polygons. While AABBs transition nicely to the regular grid structure of volume data, little research has been done on their performance for volume rendering on GPU. This is partly due to the impression that BVH build times impede interactive exploration via transfer function updates. Additionally, it seems wasteful to create a BVH tree around groups of adjacent active regions that may be considered dense and thus may as well share a leaf node to reduce build and traversal complexity. Recent advancements in GPU technology have provided hardware-based BVH traversal and ray-primitive in-

[†] ganterd@scss.tcd.ie
[‡] manzkem@scss.tcd.ie

tersections which has the potential to make the GPU a more viable candidate for BVH DVR.

In this paper we investigate the characteristics of BVHs on GPU in terms of render-performance, render-complexity and build times, and compare against a recent state-of-the-art approach. We present a method to cluster acceleration structure leaves that has a significant impact on render times due to reduced tree and depth-complexity, with leaf-count reductions of up to 50% in the average case, improving render times by roughly 10-15%. We propose that using BVHs on GPU for DVR is now a viable approach and that tree-build times do not impede interactive exploration and are in fact one of the lowest costing stages of the pipeline. We finally show that render times using BVHs can be 20-40% faster than a state-of-the-art implementation with less deviation from the average during exploration.

2. Background

Volume rendering, and more specifically DVR, has been a well researched topic in the field of computer graphics. Fundamentally it is the accumulation of light through a potentially heterogeneous medium represented as a regular grid of data. This data is resampled as we step along a ray accumulating colour and opacity [Lev88; EHK*06].

CPU volume rendering has also been well researched, but in general has been focused more on large-scale volume data [WJA*17; WUP*18] citing the larger RAM capabilities and removing the need to transfer data to GPU. Such work has used BVHs to accelerate data traversal [KTW*11; KWN*13; KWN*14]. These approaches all work well for large data and can scale well to clusters of systems, however in this work we focus on a single CPU - single GPU system. In this scenario the GPU currently massively surpasses the CPU in parallelism and is rapidly improving in terms of memory capacity and bandwidth.

GPU volume rendering has come a long way and a comprehensive state of the art report has been presented by Beyer et al [BHP15]. The works that we find most relevant are Fogal's Tuvok renderer [FK10; FSK13], Liu et al [LCDP13] and Hadwiger et al [HBJP12; HAB*18]. All of these works share a common theme of some form of hierarchical data-structure. For example [FK10; FSK13; LCDP13] use octrees for ESS and data paging/sampling. [HBJP12] instead use a multi-level hierarchical data structure to facilitate paging of peta-scale volumes, and expand upon that work [HAB*18] by using an octree-like occupancy histogram tree to generate occupancy geometry for rasterised ESS traversal. It is clear that octrees have been the data structure of choice for large scale volume rendering in most relevant works [CNLE09; RV06; LCDP13; FK10; FSK13; HBJP12], chosen for it's logarithmic search times and inherent adaptability for level-of-detail (LOD) data. However, as recent works make note [LBG*16; HAB*18], octrees can be more of a hindrance for dense regions of the volume where overhead is introduced traversing from brick to brick in sparse volumes with potential thin strands of opaque media.

2.1. Nvidia OptiX & RTX

In terms of actual BVH research, there has been recent work into hardware acceleration of construction [DTM18], traversal and intersection test [LSL*13], and some of these concepts have finally been implemented in consumer hardware. With surface path-tracing being the topic of an immense amount of research, it was only a matter of time before some of these concepts were introduced as fixed-function hardware. While the OptiX SDK by Nvidia [PBD*10] has been around for a while now, it has been utilising the massive parallel compute power that was already present in hardware and exposed in CUDA, although hiding the specifics of the underlying hardware. Only recently has this been accelerated by implementing some core path-tracing concepts like BVH traversal, AABB and ray-triangle intersection hardware with the new RTX line of GPUs.

While octrees may be a more trivially suitable candidate for DVR, we feel that a proper investigation and evaluation of BVHs in the context of GPU DVR is now warranted. This means analysing the traversal characteristics and performance of both octrees and BVH. For interactive DVR exploration especially, how build times and traversal change with respect to transfer-function updates are vitally important to the end-user scenario.

3. Evaluation of BVHs for DVR on GPU

There are many reasons why a BVH approach may be chosen over a more regular acceleration data-structure such as an octree. For example, even if both an octree and a BVH share the same leaf-size in a sub-divided volume, sparsely populated data may require fewer inner tree nodes to be traversed to achieve the same amount of skipped empty space. Secondly, using a BVH allows for the seamless integration into existing path-tracing tools, such as Nvidia's Optix which can then be used in production renderers for offline path-tracing when volumetric media needs to be used — clouds or smoke for example. Following that, by using Optix, much of the hard work can be accomplished by the existing library, and offloaded to hardware accelerated implementations.

However, advantages aside, as discussed previously there has been little investigation into performance characteristics of BVH DVR on GPUs. We feel the need to emphasise again that — especially with new hardware acceleration — an evaluation of BVHs needs to be performed. In particular, we are interested in two major parts of the interactive DVR pipeline: transfer function editing and spatial exploration. In the first example, a user may tweak a transfer function making certain data less or more visible. When data that was previously completely transparent becomes visible, this means that the acceleration data-structure needs to be updated. In the case of the BVH, this potentially means that a new leaf-node needs to be added to the scene and the hierarchy needs to be updated. It is this build time that needs to be evaluated if BVH DVR is to be a viable candidate.

For the second task — spatial exploration — we need to determine how BVH traversal handles many potentially transparent regions of volume data that may be redundantly touching (more about that in section 4) creating additional depth complexity. In a current state-of-the-art approach [HAB*18] rasterisation of occupancy geometry — which can loosely be thought of as an octree although the actual geometry skips levels depending on the transfer function — generates a list of ray-segments which can be traversed in order to efficiently skip empty space. Because these ray-segments can be compressed on the fly, continuous regions of active leaves can compressed into a single ray segment. In comparison to BVH ray-traversal, exiting one leaf and entering an adjacent leaf can require a full or partial restart of the BVH search. We evaluate the impact this has on DVR with both opaque and mostly transparent transfer functions which generate the same amount of leaves, only differing in the amount of early terminated rays. In the next section, we also show how we can mitigate this effect.

4. Region Clustering for BVH

A core contribution of our work is reduction of data structure complexity by spatially clustering active subdivision leaves in the volume. There are two major advantages that justify this step: Firstly, BVH construction complexity can be reduced substantially, facilitating faster refit or rebuild times when the transfer function changes, although in section 6 we show that BVH build times are quite fast even without this step.

Secondly — and perhaps more importantly — BVH traversal complexity can be reduced substantially. By clustering active leaves in the subdivision we can massively reduce the amount of BVH leaves, reducing the complexity of the hierarchy and thus ray traversal times, demonstrated in section 6.2. These benefits do however come at a cost, which is the actual clustering phase which we perform on the CPU. We face two major challenges in this task.

4.1. 3D Summed Area Table

The first challenge is the complexity and performance of finding ideal or close to perfect clustering that minimises the leaf node count. A simple naive single-pass approach could begin by traversing the bricks in a scanline fashion, attempting to group as many bricks as possible. While simple and easy to implement, this has the major drawback of potentially increased fragmentation as the scanline proceeds, resulting in larger clusters at the beginning of the scanline and many smaller clusters towards the end. In our solution we use a greedy-like algorithm to attempt to cluster bricks in descending size using a copy of a vector of booleans representing the active leaves in the subdivision, coupled with a 3D version of a summed area table[Cro84] (3DSAT).

The process begins by creating a vector of booleans — or a long bit string — that represents the active leaves in the subdivision. We show 2D example of this in figure 2 — labeled 'active brick mask'. We then fill a 3DSAT the same dimensions as the subdivision grid (step 1 in figure 2). If a leaf is active the sum is increased by 1 and the general process for populating an SAT is used to generate the 3DSAT. When clustering the leaves we query the 3DSAT to determine if there are the same amount of active leaves as the amount of leaves that we want to cluster — for example if we are attempting to cluster a 3^3 group of leaves, query the 3DSAT for the sum of all active leaves in an 3^3 window (step 2 in figure 2). If the result is exactly 3^3 we know that all the leaves in the region are active.

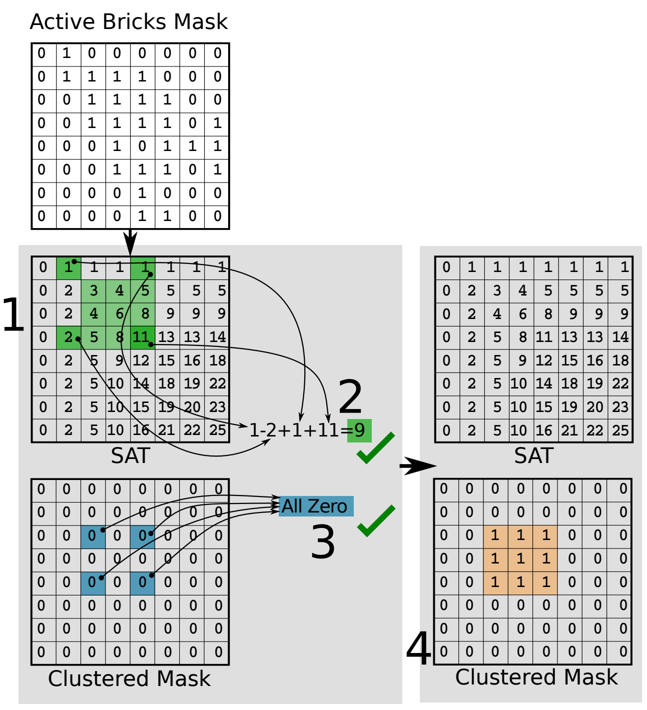

Figure 2: *Example of summed-area-table clustering, shown here in a 2D example attempting to cluster a 3x3 region. This process is explained in section 4.1.*

We also check the 8 corners of the 3^3 window against the clustered leaves mask (step 3 in figure 2). If all 8 corners are zero, this region has not been clustered before. When a region is clustered, the clustered leaves mask is updated so that all leaves grouped by the cluster are marked as 'clustered' — or 1 — (step 4 in figure 2) and are therefore not considered in following checks. This sliding window can be thought of as a form of convolution kernel, albeit in serial. We outline this process in algorithm 1. It's important to note about this algorithm that a copy of the active list of bricks is used. When a cluster is found, the leaves that are covered by that cluster are marked as inactive in the copied list, effectively acting as a mask preventing those leaves being added to a different cluster.

4.2. Brick Pool

The second challenge we face lies with the brick pool: What is the easiest way for the brick pool to handle clustered regions of ESS information? The short answer is that we decouple the ESS from the actual DVR sampling. In our solution we take inspiration from [HBJP12] and [HAB*18], who separate the ESS information from the actual volume sampling layer. This means that the BVH can use a fine-grained leaf size and the volume pool can use a brick size that suits I/O needs or page table complexity, a trait that can be important in massive-scale DVR. While [HAB*18] use a multi-level page structure, for our purposes we used a minimised version, We discuss in more detail about our implementation of brick pool and page table in section 5.3.

Algorithm 1: Greedy algorithm to cluster regions of ESS leaf nodes. Note that *clusterSizeList* is in descending order and that *maxClusterSize* is a volume size limited, user defined variable.

```
activeList = getActiveLeaves();
sat = generateSAT(activeList);
clusteredMask = bitmask(activeList.size());
clusterSizeList = {maxClusterSize .. 1};
clusterList = { };
for c in clusterSizeList do
    area = c * c * c;
    for z in numLeaves.z - c do
        for y in numLeaves.y - c do
            for x in numLeaves.x - c do
                vec3 min = {x, y, z};
                vec3 max = min + {c, c, c};
                if !cornersActive(clusteredMask, min, max) then
                    continue;
                end
                if sat.sumBetween(min, max) == area then
                    clusterList.push(min, max);
                    clusteredMask.setOnes(min, max);
                end
            end
        end
    end
end
```

5. Implementation

5.1. Occupancy Information & OptiX

Our method begins with generating the occupancy information. The volume is subdivided into bricks of identical sizes, storing some information about the brick contents; i.e. min/max value or a bit-mask of value ranges. In our implementation we just used min/max values for a brick. This information will be tested against the transfer function when there is an interaction. This content information can be saved to disk for future re-use of the volume.

When the transfer function is updated the content information array is tested in parallel — using OpenMP in our implementation — and the active/inactive flags are stored in a bit-mask array. At this point, if clustering is not enabled, the active/inactive flags can then be used to generate bounding box information — in this implementation we use world-space min/max coordinates of each region as an AABB. This information is stored in a buffer which effectively represents separate primitives in an OptiX geometry instance.

5.2. Clustering

If clustering *is* enabled we use the method outlined in section 4 and listed in algorithm 1 to generate a vector of min-leaf-index to max-leaf-index bounds which is then used to generate AABB min/max bounds. In our implementation this clustering method is single-threaded and a potentially naive method of solving the task, however this should be seen as a proof-of-concept method to accelerate stages later in the BVH DVR pipeline like build times and ray-traversal in both opaque and transparent volumes. Regardless of if clustering is enabled or not, the AABB information is stored in an OptiX buffer and used as geometry primitives in a *single* geometry instance that has a *single* material assigned to it. During development this was found to be much more performant than a geometry instance per AABB.

5.3. Brick Pool, Page Table & Sampling

The underlying sampling technique used in both our OptiXDVR approach and our implementation of SparseLeap [HAB*18] is a simplified version of Hadwiger et al's earlier work [HBJP12]. When sampling we perform a look-up into a page table. This look-up determines if the region that the sample resides is active. This look-up also gives an offset into a large brick-pool where the actual volume data resides. As noted in SparseLeap [HAB*18], this allows the disconnection of ESS and sampling/paging.

Since the efficient sampling of volumes is not the primary focus of our work relative to empty-space-skipping, our implementation of this method is simplified by assuming two things. Firstly, we assume that the volumes are small enough to not use a multi-level paging technique, and as such just use one page table for the whole volume. This still allows considerably large volumes to be visualised. Secondly, we assume that the volume content information is known at run-time and there are no ‘unknown’ regions of the volume. Regardless of whether this is done offline or in a pre-processing step, we don’t evaluate streaming incomplete data, although the underlying sampling can be updated to facilitate this.

It is important to again stress at this time that the sampling part of our DVR implementation can be made more complex to accommodate the above requirements if needed. This would require few changes to the BVH ESS part of the pipeline.

6. Results & Evaluation

In this section we evaluate the performance characteristics of our BVH clustering approach when using different parameters, i.e. volume, transfer function, clustering, leaf size, brick size, etc. We additionally compare our approach to our implementation of SparseLeap [HAB*18]. We have implemented this to the best of our ability using the pseudo-code provided and observed much of the same characteristics has shown in their evaluations. We used the same OpenGL ARB extension *ARB_fragment_shader_interlock* to process occupancy geometry in order per-fragment. For linked-list generation we took inspiration — like [HAB*18] — from [YHGT10].

While SparseLeap is implemented in OpenGL and our BVH DVR approach is using OptiX, most of the underlying code for occupancy information, paging, etc, are common to both, minimising differences as much as possible and making fair comparisons. We briefly discuss the underlying sampling method in section 5.3. The OpenGL SparseLeap implementation and the BVH OptiX implementation use the *exact* same underlying implementation and data for the page table and brick-pool, the only differences being that

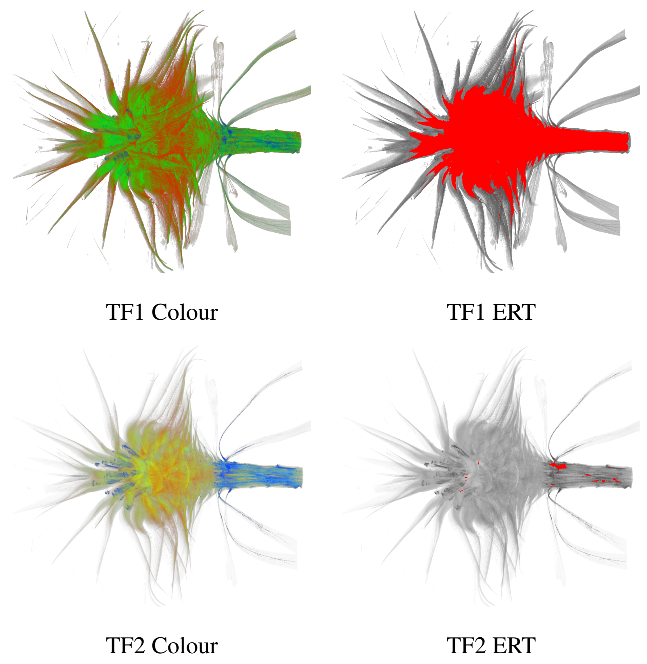

Figure 3: *Examples of the flower dataset used with the two different transfer functions. Note that the ERT rays are highlighted in red and are substantially less prevalent in the second transfer function. Also note that both transfer functions set the exact same amount of active/inactive regions, thus the only difference is amount of ERT.*

SparseLeap uses OpenGL textures and the BVH DVR uses *rtBuffer* objects and *rtTextureSampler*. We time the appropriate sections using the system clock, starting and stopping a timer before and after the evaluated stats. For OpenGL, *glFinish()* is called before starting the timer and before stopping the timer.

The test system used an Intel Xeon E5-1620 v2 and an Nvidia GeForce RTX2080 using Ubuntu 18.04. The OptiX SDK was version 6.0, OpenGL version was 4.5, using Nvidia driver 418.43.

6.1. Data-sets

To best maintain a level of fair comparisons we use a multitude of data-sets during experimentation. The primary data-set shown in the paper is a μCT scan of a flower with a resolution of 1024^3 8-bit integers obtained from UZH [UZH]. Examples of this data-set are shown in figure 3. The beechnut data-set also obtained from UZH is 1024^2x1546 16-bit unsigned integer, and although not shown in any of the images or figures, exhibited similar results to the Flower data-set. Both of these data-sets obtain a decent level of clustering while maintaining regions of thin strands that are difficult to cluster.

In contrast we also use a frame from a Supernova simulation previously obtained from UC Davis. This data is up-scaled from 432^3 to 2048^3 8-bit unsigned integer data. The Supernova with the tested transfer function exhibited large amounts of cluster-able regions. An example of this data-set is shown in figure 5. The website for this data-set was no longer available when last checked, but copies of the data can be supplied on request.

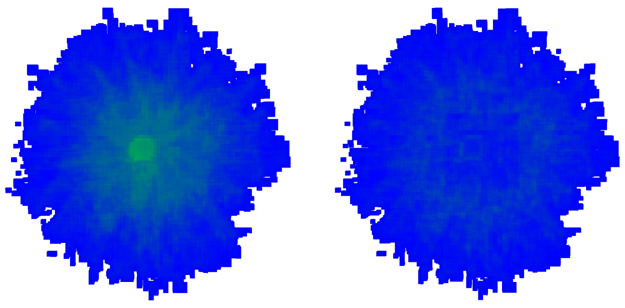

Figure 4: *Depth complexity comparison of a head-on view of the flower data-set with a leaf size of* 16^3 *voxels. See section 6.2 for explanation of heat-map colour coding, and table 1 for statistics on the amount of clusters present.*

Both the Flower and the Supernova data-sets are chosen to represent varying degrees of cluster-ability, the percentages of which can be seen in the '% of B_{Active}' column in 1. However, in addition to different data-sets, we find it important to emulate the interactive nature of DVR applications by using different transfer functions on the same volume. In figure 3 we show two different transfer functions on the same volume. This is necessary to evaluate the impact that large amounts of sub-division leaf nodes has on depth complexity for both the SparseLeap implementation and the BVH OptiX DVR, as such TF1 has a high ratio of early-ray-termination (ERT) relative to rays which actually sample the volume (rays which enter at least one active leaf). TF2 is quite the opposite with very little ERT, allowing rays to traverse through the volume almost entirely. It is important to note that both TF1 and TF2 exhibit the *exact* same amount of active leaves in the subdivision and only differ in opacity accumulation.

6.2. Clustering

In table 1 we show comprehensive statistics about both the leaf subdivision and clustering statistics of our method. We can see that for the flower data-set — using both transfer functions — a reduction to between approximately 40% to 50% of BVH leaf nodes is achieved. For the Supernova data-set which has much more contiguous space, we see a more substantially reduced complexity to between 5% and 35% of the original leaf count relative to unclustered.

We show visual comparisons of depth complexity both with and without clustering enabled in figures 4 and 5. The colours range from blue to red representing a depth complexity of 1 to *MaxDC* which is defined as the manhattan distance from one corner to the opposite corner of the volumes going by bricks, i.e. $numLeaves.x + numLeaves.y + numLeaves.z$. Note that none of the displayed images ever reach a depth complexity of 100% since there is always a portion of the volume that is skipped and the experiments are mostly run on the horizontal x-plane.

Using the flower data-set we can see in figure 4 that there is an

	B_{size}	B_{Total}	B_{Active}	(% of B_{Total})	T_{tf}	$B_{Clusters}$	(% of B_{Active})	$T_{Cluster}$
Flower	128	512	261	50.98%	0.01ms	139	53.26%	0.05ms
	64	4,096	1,126	27.49%	0.08ms	566	50.27%	0.37ms
	32	32,768	4,883	14.90%	0.73ms	2,606	53.37%	1.89ms
	16	262,144	22,058	8.41%	1.58ms	10,605	48.08%	17.63ms
	8	2,097,152	112,139	5.35%	7.48ms	43,603	38.88%	112.46ms
Supernova	128	4,096	811	19.80%	0.03ms	268	33.05%	0.08ms
	64	32,768	5,324	16.25%	0.14ms	1,043	19.59%	1.67ms
	32	262,144	4,883	14.63%	0.60ms	5,232	13.64%	13.00ms
	16	2,097,152	290,864	13.87%	5.98ms	23,947	8.23%	107.33ms
	8	16,777,216	2,262,811	13.49%	27.68ms	112,801	4.98%	1090.02ms

Table 1: *Statistics on subdivision leaf clustering for different leaf-sizes (‘B_{size}’) showing the total number of subdivision leaves (‘B_{Total}’), the number of active leaves (‘B_{Active}’) for the given transfer function and that number as a percentage of the total number of leaves (‘% of B_{Total}’), the time taken to test all leaves against the transfer function (‘T_{tf}’), the amount of clustered leaves (‘$B_{Clusters}$’), also represented as a percentage of the amount of active leaves (‘% of B_{Active}’) and the amount of time taken to cluster the leaves (‘$T_{Cluster}$’).*

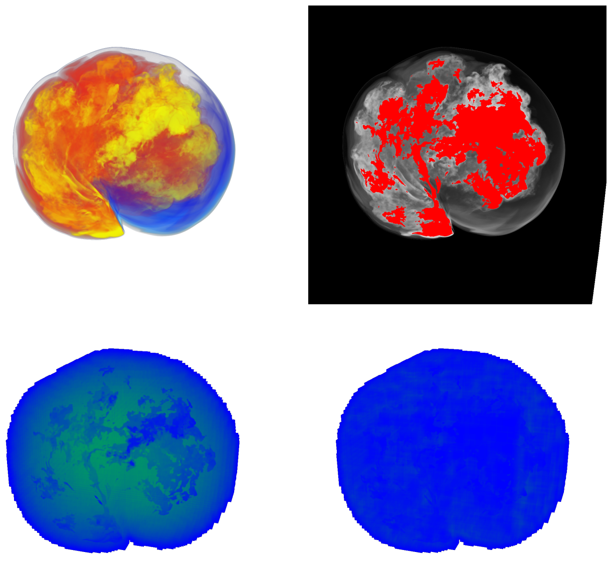

Figure 5: *Depth complexity comparison of a head-on view of the supernova data-set with a leaf size of 32^3 voxels. This shows the final render (top-left), the ERT (top-right), the depth complexity with clustering off (bottom-left) and on (bottom-right).*

observable reduction in depth complexity straight through the middle of the volume. There is however a substantial amount of non cluster-able regions — or regions that were already relatively low in depth-complexity — around the fringes of the volume. Looking at figure 3 which shows a side-view of the volume, we can see that the strands of active regions are thin enough to make it difficult to cluster. On the other hand, using something like the Supernova data-set shown in figure 5 which exhibits large amounts of adjacent subdivision leaves, we can see a considerable reduction of depth complexity throughout the volume. These claims are backed up by statistics in table 1 and the render performance benefits can be observed in figure 6.

6.3. BVH Build Times

BVH build times have been a major factor in the lack of adoption for DVR. To evaluate the actual implications, we vary the transfer function to increase the amount of active leaves in the subdivision. In figure 7 we show the build times with a varying active leaf rate of approximately 8% to 100% for a leaf-size of 16^3 using the flower data-set. Note that this time *includes* the time it takes for creating AABB information and uploading to the GPU. We can see that there is a relatively linear increase in time, but remains well below 20ms for this data-set.

This only tells part of the story however. In table 1 we show data relating to clustering time for different configurations. We can see that, for the most part, the time taken to cluster a volume dwarfs the time taken for the BVH build. It is important to note however, that our clustering implementation is naive since it is a proof-of-concept for leaf complexity in BVH building and ray traversal, and can potentially be improved upon substantially.

We also compare the BVH build times to the occupancy geometry generation step of SparseLeap [HAB*18]. We observed that both the occupancy tree update time (which included geometry emission) and the occupancy geometry render order time were relatively consistent at 16ms (shown in figure 7) and 2ms respectively.

6.4. BVH Traversal Costs

An important part of evaluation is BVH traversal performance for DVR using OptiX. We can roughly estimate the portion that belongs to OptiX based on the difference of rendering a volume *with* and *without* sampling the data while varying the brick size. This was achieved by stubbing the sampling code in the first-hit OptiX program by use of a run-time flag. In figure 8 we show the results of this experiment. As expected, we can see the rising cost of BVH traversal (‘Not Sampled’) as the brick size decreases and the subdivision count increases. We can see this has an impact on the end render time (‘Sampled’). Interestingly, and again as expected, removing the cost of BVH traversal from the overall render time, the difference reveals the actual cost of sampling the volume, which has a mostly reducing trajectory. This justifies using smaller bricks

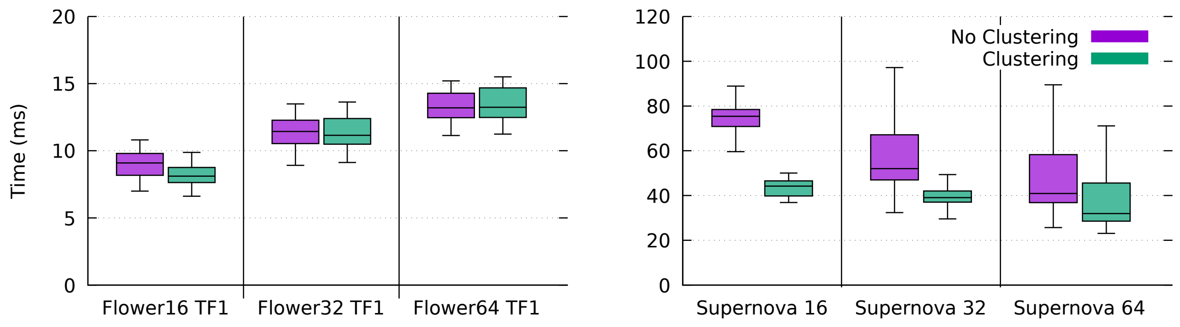

Figure 6: *Difference in render times with clustering off/on for differing subdivision leaf sizes of* 16^3*,* 32^3 *and* 64^3 *for the Flower data-set using transfer function 1 ('TF1') shown in figure 3 and the Supernova data-set shown in 5. Massive stability in render times can be seen by using clustering for the Supernova. In the Flower case, we see improvements even for the mostly opaque transfer function. This is important so as BVH traversal complexity is reduced in the event many leaf nodes are not needed for rendering, rather than having a deep hierarchy where most of the leaves aren't even touched.*

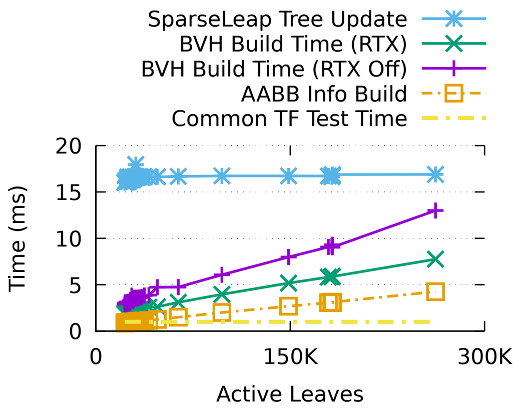

Figure 7: *Comparison of times for different renderers and configurations with varying amount of active subdivision leaves. Both the SparseLeap and OptiX DVR implementations share the same transfer-function leaf-test code. Because the SparseLeap tree size doesn't change, the update time remains almost constant. We show results for the BVH build time with and without RTX enabled which includes the time taken to generate and upload the AABB bounds ('AABB Info Build') and thus should be offset by this value to find the actual build time.*

for tighter ESS granularity, but indicates the requirement for a reduction in data-structure traversal complexity. For this reason that a clustering approach is proposed.

6.5. With & Without RTX

A core evaluation that we performed is the actual benefit of the new RT cores hardware. Starting with the first step in the pipeline, we look at BVH build times. In figure 7 we show the build times — including AABB information creation and upload — with and without RTX. Interestingly, although we are not aware of any advertised hardware for BVH construction, there is a substantial performance

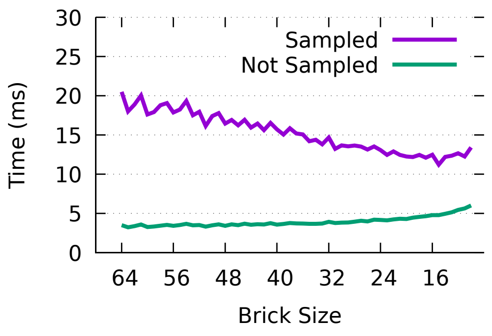

Figure 8: *Average render times Flower data-set using TF2 (no ERT) with sampling on and off to evaluate the portion of rendering responsible for BVH traversal and scheduling in OptiX.*

increase observed — dropping from approximately 10ms to less than 5ms when removing AABB information build time from the respective BVH build times.

The major part of the pipeline that we expected RTX to show massive performance benefit is during rendering. In figure 9 we show the difference in *just traversal* times to highlight any improvement between configurations. We can see substantial benefits to using the RT core hardware acceleration, especially as the leaf size reduces, improving ESS granularity but maintaining a steady level of performance.

6.6. Render Times

Finally, and potentially most importantly, we evaluate the actual performance of DVR for both our clustered BVH approach and our implementation of SparseLeap. In figure 10 we show render-time results using both TF1 and TF2 to highlight the differences between performance when ERT is a prevalent feature during ray traversal. In all of these tests, the underlying sampling method used a brick pool with a brick size of 32^3, which we find to be the most perfor-

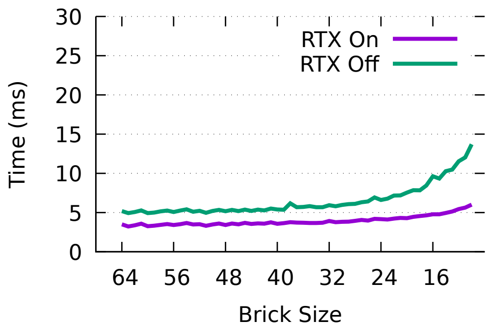

Figure 9: *Difference in BVH traversal times with RTX off/on for differing subdivision leaf sizes for the Flower data-set. Volume sampling is stubbed to get an estimation of just BVH performance. Clustering is not enabled for this experiment to force a larger amount of leaf nodes. See table 1 for indicative leaf counts, This shows clear benefit for using the RT core hardware when spatially subdividing a volume.*

mant in this case for both approaches. The results were obtained by performing 50 full rotations in 360 steps with the volume filling as much of the view-port without being cut off, timing the appropriate stages using the method outlined at the beginning of this section. Examples of frame 90 — a side-on view of the volume — can be seen in figure 3.

In almost all cases we see a substantial performance benefit using the clustered BVH approach in terms of average render times. The exception to this rule is when there is little ERT and the leaf size is relatively large. In the case of SparseLeap we can see that a small leaf-size can sometimes be a hindrance during rendering. A smaller leaf-size in general means a finer level of granularity, but for a relatively fragmented volume such as the flower, segment counts can become quite substantial creating extra work for the fragment shader, showing there is a balance to strike.

In comparison, we see that the average render performance for clustered BVH DVR is significantly better in most cases than the best-case SparseLeap configuration. For the flower volume using TF1 we observe an almost 40% improvement in render times, a statistic echoed using TF2. Another important quality that is often overlooked is render performance stability. In figure 10 we can see that while there is deviation of minimum and maximum render times from the average, it is quite stable relative to SparseLeap. we consider this an important quality in DVR when exploring a volume.

7. Conclusions & Future Work

We have shown that BVHs are an extremely viable candidate for DVR on modern GPUs that implement new hardware for ray-tracing, giving us tight-wrapping empty-space-skipping structures with little effort. We have presented a method of leaf-clustering to help ray-traversal performance during rendering and prove the benefits in terms of both depth-complexity and traversal times. We note that while clustering improves the aforementioned stages the actual clustering times can become a bottleneck for transfer function update times. Furthermore, since the 3DSAT needs to be the same dimensions as the amount of subdivisions, it can become quite memory intensive with massive volumes or fine grained subdivisions. It is necessary to keep in mind that this is a relatively naive clustering implementation used as a proof-of-concept to demonstrate a method of reducing leaf complexity for BVHs. Their performance may be improved with more efficient algorithms, potentially using fast convolution kernels and using the massive parallelism on the GPU. It is important to re-iterate that one of the main reasons BVHs were previously avoided for interactive DVR was build-times. We have shown that these build times — including the necessary steps to facilitate the build — are highly interactive and should not be considered a limiting factor.

While all of this work focuses on the ESS stage of DVR, we of course observed that the main bottleneck in volume rendering is I/O, both in terms of loading data from disk/network and then uploading to the GPU. While our work does not consider this field relevant to it's contributions, it is nonetheless a vital consideration for any large-scale DVR. It is also important to reiterate that — in our implementation — there is a separation between ESS information and the actual volume data insofar that the sampling and the space-skipping do not necessarily need to share the same data, a trait also present in [HBJP12; HAB*18].

The code for this work has been made publicly available at *github.com/ganterd/optixdvr*, and we encourage others to use, build upon and compare against the implementation.

Acknowledgements

We acknowledge the Computer-Assisted Paleoanthropology group and the Visualization and MultiMedia Lab at University of Zurich (UZH) for the acquisition of the μCT data-sets. The authors wish to thank Seán Martin and Seán Bruton for their valuable input. This research has been conducted with the financial support of Science Foundation Ireland (SFI) under the Grant Number 13/IA/1895.

References

[BHP15] BEYER, JOHANNA, HADWIGER, MARKUS, and PFISTER, HANSPETER. "State-of-the-Art in GPU-Based Large-Scale Volume Visualization". *Computer Graphics Forum* 34.8 (2015), 13–37. ISSN: 1467-8659. DOI: 10.1111/cgf.12605 2.

[CNLE09] CRASSIN, CYRIL, NEYRET, FABRICE, LEFEBVRE, SYLVAIN, and EISEMANN, ELMAR. "GigaVoxels: Ray-guided Streaming for Efficient and Detailed Voxel Rendering". *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*. I3D '09. New York, NY, USA: ACM, 2009, 15. ISBN: 9781605584294. DOI: 10.1145/1507149.1507152 2.

[Cro84] CROW, FRANKLIN C. "Summed-area tables for texture mapping". *ACM SIGGRAPH Computer Graphics* 18.3 (1984), 207–212. ISSN: 00978930. DOI: 10.1145/964965.808600 3.

[DTM18] DOYLE, MICHAEL J., TUOHY, CIARAN, and MANZKE, MICHAEL. "Evaluation of a BVH construction accelerator architecture for high-quality visualization". *IEEE Transactions on Multi-Scale Computing Systems* 4.1 (Jan. 2018), 83–94. ISSN: 23327766. DOI: 10.1109/TMSCS.2017.2695338 2.

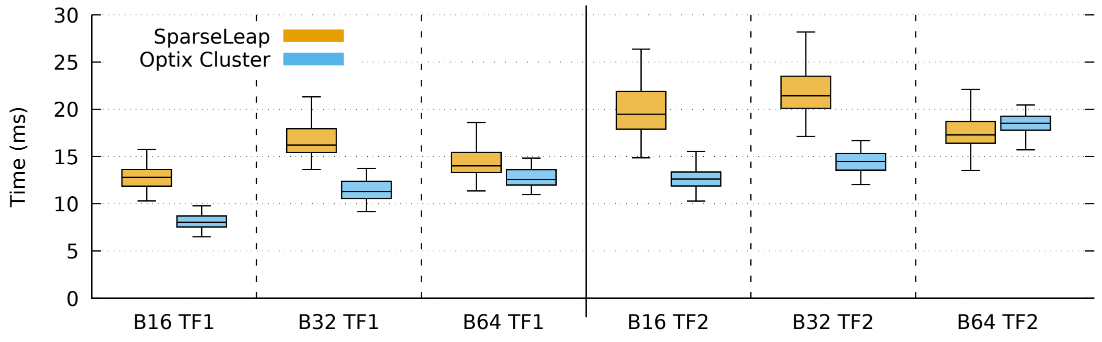

Figure 10: *Render times comparison of SparseLeap[HAB*18] to our approach using the Flower data-set with different leaf sizes ('Bxx') and the two different transfer functions ('TF1' and 'TF2'). The underlying sampling layer used a brick size of* 32^3. *Note that the SparseLeap times do not include the geometry rasterisation step required for when the camera moves.*

[EHK*06] ENGEL, KLAUS, HADWIGER, MARKUS, KNISS, JOE M., et al. *Real-time volume graphics*. CRC Press, 2006, 29–es. ISBN: 0111456789. DOI: 10.1145/1103900.1103929. arXiv: arXiv:1011.1669v3 2.

[FK10] FOGAL, THOMAS and KRÜGER, JENS. "Tuvok - An Architecture for Large Scale Volume Rendering". *15th Vision, Modeling and Visualization Workshop '10*. 2010, 139–146. ISBN: 9783905673791. DOI: 10.2312/PE/VMV/VMV10/139-146 2.

[FSK13] FOGAL, THOMAS, SCHIEWE, ALEXANDER, and KRUGER, JENS. "An analysis of scalable GPU-based ray-guided volume rendering". *IEEE Symposium on Large Data Analysis and Visualization 2013, LDAV 2013 - Proceedings*. IEEE, Oct. 2013, 43–51. ISBN: 978-1-4799-1659-7. DOI: 10.1109/LDAV.2013.6675157 2.

[HAB*18] HADWIGER, MARKUS, AL-AWAMI, ALI K., BEYER, JOHANNA, et al. "SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering". *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), 974–983. ISSN: 10772626. DOI: 10.1109/TVCG.2017.2744238 2–4, 6, 8, 9.

[HBJP12] HADWIGER, MARKUS, BEYER, JOHANNA, JEONG, WON KI, and PFISTER, HANSPETER. "Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach". *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), 2285–2294. ISSN: 10772626. DOI: 10.1109/TVCG.2012.240 2–4, 8.

[KTW*11] KNOLL, AARON, THELEN, SEBASTIAN, WALD, INGO, et al. "Full-resolution interactive CPU volume rendering with coherent BVH traversal". *IEEE Pacific Visualization Symposium 2011, PacificVis 2011 - Proceedings*. IEEE. 2011, 3–10. ISBN: 9781612849324. DOI: 10.1109/PACIFICVIS.2011.5742355 2.

[KWN*13] KNOLL, AARON, WALD, INGO, NAVRÁTIL, PAUL A., et al. "Ray tracing and volume rendering large molecular data on multi-core and many-core architectures". *Proceedings of the 8th International Workshop on Ultrascale Visualization - UltraVis '13*. New York, New York, USA: ACM Press, 2013, 1–8. ISBN: 9781450325004. DOI: 10.1145/2535571.2535594 2.

[KWN*14] KNOLL, AARON, WALD, INGO, NAVRATIL, PAUL, et al. "RBF volume ray casting on multicore and manycore CPUs". *Computer Graphics Forum* 33.3 (June 2014), 71–80. ISSN: 14678659. DOI: 10.1111/cgf.12363 2.

[LBG*16] LABSCHUTZ, MATTHIAS, BRUCKNER, STEFAN, GROLLER, M. EDUARD, et al. "JiTTree: A Just-in-Time Compiled Sparse GPU Volume Data Structure". *IEEE Transactions on Visualization and Computer Graphics* 22.1 (Jan. 2016), 1025–1034. ISSN: 1077-2626. DOI: 10.1109/TVCG.2015.2467331 2.

[LCDP13] LIU, BAOQUAN, CLAPWORTHY, GORDON J., DONG, FENG, and PRAKASH, EDMOND C. "Octree rasterization: Accelerating high-quality out-of-core GPU volume rendering". *IEEE Transactions on Visualization and Computer Graphics* 19.10 (2013), 1732–1745. ISSN: 10772626. DOI: 10.1109/TVCG.2012.151 2.

[Lev88] LEVOY, MARC. "Display of Surfaces from Volume Data". *IEEE Computer Graphics and Applications* 8.3 (May 1988), 29–37. ISSN: 02721716. DOI: 10.1109/38.511 2.

[LSL*13] LEE, WON-JONG, SHIN, YOUNGSAM, LEE, JAEDON, et al. "SGRT: a mobile GPU architecture for real-time ray tracing". *Proceedings of the 5th High-Performance Graphics Conference on - HPG '13*. New York, New York, USA: ACM Press, 2013, 109–119. ISBN: 9781450321358. DOI: 10.1145/2492045.2492057 2.

[PBD*10] PARKER, STEVEN G., BIGLER, JAMES, DIETRICH, ANDREAS, et al. "OptiX: A General Purpose Ray Tracing Engine". *ACM Transactions on Graphics* 29.4 (2010), 1. ISSN: 07300301. DOI: 10.1145/1833351.1778803 2.

[RV06] RUIJTERS, DANIEL and VILANOVA, ANNA. "Optimizing GPU Volume Rendering". *Winter School of Computer Graphics (WSCG)* 14 (2006) 2.

[UZH] UZH. *Research Datasets*. https://www.ifi.uzh.ch/en/vmml/research/datasets.html. Accessed: 2019-01-05 5.

[WJA*17] WALD, I, JOHNSON, G. P., AMSTUTZ, J, et al. "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization". *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), 931–940. ISSN: 10772626. DOI: 10.1109/TVCG.2016.2599041 2.

[WUP*18] WU, QI, USHER, WILL, PETRUZZA, STEVE, et al. "VisIt-OSPRay : Toward an Exascale Volume Visualization System". *Eurographics Symposium on Parallel Graphics and Visualization* Vi (2018). DOI: 10.2312/pgv.20181091 2.

[YHGT10] YANG, JASON C., HENSLEY, JUSTIN, GRÜN, HOLGER, and THIBIEROZ, NICOLAS. "Real-Time Concurrent Linked List Construction on the GPU". *Computer Graphics Forum* 29.4 (Aug. 2010), 1297–1304. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2010.01725.x 4.