Computational Visual Media
https://doi.org/10.1007/s41095-019-0155-y Vol. 6, No. 1, March 2020, 53–63

Research Article

Efficient ray casting of volumetric images using distance maps for empty space skipping

Lachlan J. Deakin[1] (✉), **Mark A. Knackstedt**[1]

Abstract Volume and isosurface rendering are methods of projecting volumetric images to two dimensions for visualisation. These methods are common in medical imaging and scientific visualisation.

Head-mounted optical see-through displays have recently become an affordable technology and are a promising platform for volumetric image visualisation. Images displayed on a head-mounted display must be presented at a high frame rate and with low latency to compensate for head motion. High latency can be jarring and may cause cybersickness which has similar symptoms to motion sickness.

Volumetric images can be very computationally expensive to render as they often have hundreds of millions of scalar values. Fortunately, certain materials in images such as air surrounding an object boundary are often made transparent and need not be sampled, which improves rendering efficiency.

In our previous work we introduced a novel ray traversal technique for rendering large sparse volumetric images at high frame rates. The method relied on the computation of an occupancy and distance map to speed up ray traversal through empty regions.

In this work we achieve higher frame rates than our previous work with an improved method of resuming empty space skipping and the use of anisotropic Chebyshev distance maps. An optimised algorithm for computing Chebyshev distance maps on a graphical processing unit is introduced supporting real-time transfer function editing.

Keywords ray casting; volume rendering; isosurface rendering; distance maps

1 Australian National University, Canberra, 2601, Australia. E-mail: L. J. Deakin, lachlan.deakin@anu.edu.au (✉); M. A. Knackstedt, mark.knackstedt@anu.edu.au.

Manuscript received: 2019-12-14; accepted: 2019-12-21

1 Introduction

Volumetric images (volumes) are three-dimensional scalar fields of voxels (**vo**lume + pi**xels**). Volumes can be acquired with imaging technologies such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) which make it possible to see inside opaque physical objects or people.

Computer graphics hardware and pipelines have generally been optimised for rendering surface representations of scenes. Volumes are not surfaces and must be rendered with specialised *volume rendering* techniques. Volume ray casting is a common volume rendering method well suited to acceleration on a Graphical Processing Unit (GPU) [1–3]. Rays are projected from each pixel of a display which pass through volumes and composite the colours and opacities of the voxels they pass through.

Voxels can be assigned a colour and opacity based on their scalar value (and often their gradient magnitude) according to a *transfer function*. Transfer functions can be manipulated by a user to control the appearance and visibility of different materials. Lighting effects such as shadowing, specularity, and scattering can be applied but are not widely used in medical image analysis.

Isosurfaces are surfaces representing constant scalar values within a volume [4]. An isosurface can be extracted as a mesh and rendered using conventional surface rendering techniques. Isosurfaces can also be rendered using ray casting directly on an image without a mesh extraction [5].

Modern augmented reality Head-Mounted Displays (HMDs) which are optically see-through enable new volumetric image visualisation applications. These devices are able to resolve their position as a user moves his/her head and eyes. This makes it possible for a virtual object to be displayed in a user's field of view which may appear fixed in space or potentially aligned to a physical object. For example, a surgeon could visualise a CT or MRI image overlaid on a patient without referring to an external display. This application would provide depth perception of internal structures within the body and a surgeon would not need to look away from their hands.

Images must be displayed on a HMD at high frame rates and with low latency to sufficiently compensate for head and eye motion. Otherwise, the wearer of a HMD may experience cybersickness. Virtual reality HMDs fully replace a user's view of the world and current generation devices require frames to be rendered at 90 Hz. Recent augmented reality HMDs with see-through displays require lower frame rates of around 60 Hz.

Volume rendering can be very computationally expensive, particularly for high resolution images. Rays passing through a volume may require a large number of samples of the volumetric image texture before an output colour is determined or an isosurface is located. Ray casting must be accelerated to higher frame rates for visualisation on HMDs.

Most volumetric images have some portion of *empty space* which does not contribute to the output image. For volume rendering, empty space refers to voxels which have zero opacity as defined by the transfer function. For example, the air surrounding an object and some object materials may be made entirely transparent. Voxels which are not adjacent to surface interfaces are considered empty for isosurface rendering. Empty space can be skipped to reduce the number of texture samples and improve frame rates.

We recently introduced an efficient empty space skipping approach for ray casting [6]. A reduced resolution occupancy map is generated describing which regions of a volume are occupied. A distance map is then generated which encodes the Chebyshev distance to the nearest occupied region. The distance map is used to efficiently leap rays past empty regions during voxel traversal and reduce the number of unnecessary texture samples.

This paper is an extended version of our previous short format paper [6] which provides more details of the method and introduces:

- an optimisation of the ray traversal approach to more effectively skip empty space within and behind objects,
- an algorithm for efficiently generating isotropic and anisotropic Chebyshev distance maps on a GPU for accelerated ray casting, and
- an extended performance analysis of the approach with 2D transfer functions and various occupancy map region sizes.

2 Related work

Empty space can be skipped when ray casting by only sampling between geometry which tightly bounds the occupied space within a volume [5]. The bounding geometry may be defined by coarse bricks or a meshed isosurface. These approaches typically only skip external empty space and may have a high overhead from rasterisation of the bounding geometry. Generating the bounding geometry can also be computationally expensive.

Kruger and Westermann [1] created an *occupancy map* indicating which 8^3 regions of a volume are empty. A ray caster can use the occupancy map to skip sampling points in empty regions and reduce the total number of texture samples. Their occupancy map has a low memory overhead as it has a low resolution compared to the volume.

Distance maps encoding the Chebyshev distance metric to the nearest occupied voxel have proven to be effective for ray casting acceleration [7, 8]. A ray is able to skip many voxels with a single sample of the distance map if far away from an occupied voxel. These approaches can substantially improve frame rates for sparse images but the time needed to generate distance maps can inhibit interactive transfer function editing.

Massive volumetric images which may not fit entirely within GPU memory are becoming increasingly common from simulations and high-resolution imaging technology. They can be rendered by only allocating occupied bricks (regions) on the GPU [3, 9, 10]. Bricks can be referenced in hierarchical structures (such as an octree) and empty bricks are simply skipped as the structure is traversed.

SparseLeap [10] rasterises occupied hierarchically referenced bricks into per-pixel linked lists. Only the occupied segments of rays go through the ray casting pipeline. Their method outperformed octree-based empty space skipping and could be used to render terabyte sized sparse volumes. Bricks are not loaded onto the GPU if they are outside of the field of view or occluded.

Our previous paper described an efficient method of empty space skipping utilising distance maps [6]. Interactive transfer function editing was supported by updating reduced resolution distance maps on the GPU and massive volumes could be rendered by only allocating occupied bricks. Our method outperformed SparseLeap, octree-based, and bounding geometry (isosurface) acceleration on a set of images.

3 Our approach

3.1 Ray traversal

A volumetric image is stored on a GPU as a 3D texture with dimensions $\boldsymbol{d} = [width, height, depth]$. Voxels stored in a texture (texels) can be referenced by texel coordinates which can be unnormalised, $\boldsymbol{u} \in [\boldsymbol{0}, \boldsymbol{d})$, or normalised, $\boldsymbol{t} \in [\boldsymbol{0}, \boldsymbol{1}]$, and are related according to

$$\boldsymbol{u} = \boldsymbol{d} \cdot \boldsymbol{t} \tag{1}$$

Volumes are typically sampled at equidistant points along rays that pass through them. A ray intersecting with a volume from $\boldsymbol{t}_{\text{entry}}$ to $\boldsymbol{t}_{\text{exit}}$ can be sampled at n equidistant points as defined by

$$n = \lceil \overrightarrow{\max}(\boldsymbol{d}) \cdot \|\boldsymbol{t}_{\text{exit}} - \boldsymbol{t}_{\text{entry}}\| \cdot f \rceil \tag{2}$$

where $\overrightarrow{\max}$ is the maximum component operator and f is a sampling factor used for quality adjustment. This equation is viewpoint independent and ensures that there is at least one sample per voxel passed through (on average) provided f is greater than or equal to 1.

The change in normalised texture coordinates between each sample point is

$$\Delta \boldsymbol{t} = \frac{\boldsymbol{t}_{\text{exit}} - \boldsymbol{t}_{\text{entry}}}{n-1} \tag{3}$$

and the i^{th} sampling point along the ray is given by

$$\boldsymbol{t}_i = \boldsymbol{t}_{\text{entry}} + i \cdot \Delta \boldsymbol{t} \tag{4}$$

A volume renderer will sample the volume (with trilinear interpolation) at each $\boldsymbol{t}_i$ from $i = 0$ to $n-1$ and map the sampled values to colours and opacities. This process can become very computationally expensive and memory bandwidth intensive for large images where n is large.

3.2 Gradient map

2D transfer functions which use both voxel scalar values and gradient magnitudes can be used to highlight material boundaries and hide thick homogeneous regions when volume rendering. The gradient magnitude of a voxel can be computed efficiently using the tetrahedron method [11] shown in Algorithm 1 which is written in the OpenGL Shading Language (GLSL). This method requires only 4 voxel neighbour samples unlike the conventional central differences approach which requires 6.

Algorithm 1 Tetrahedron technique for voxel gradient magnitude calculation (GLSL)

```
layout (...) uniform sampler3D image; // global
float gradient_tetrahedron_method(ivec3 pos) {
  ivec2 k = ivec2(1,-1);
  vec3 gradient_dir = 0.25f * vec4(
    k.xyy * imageLoad(image, pos + k.xyy).x +
    k.yyx * imageLoad(image, pos + k.yyx).x +
    k.yxy * imageLoad(image, pos + k.yxy).x +
    k.xxx * imageLoad(image, pos + k.xxx).x);
  return length(gradient_dir);
}
```

Additional 4 texture samples at each sampling point to compute gradients can significantly reduce frame rates. Gradients can instead be precomputed and stored for faster sampling during ray casting.

3.3 Occupancy map

We create an *occupancy map* indicating which $\boldsymbol{B} \in \mathbb{N}^3$ sized regions, or blocks, of a volume are empty given a transfer function or isosurface thresholds. The occupancy map is a simple acceleration structure used to avoid sampling voxel values and gradients in regions where all voxels have zero opacity. It is a foundation for more advanced acceleration structures.

The occupancy map has dimensions $\boldsymbol{d}_{\text{occ}} = \lceil \boldsymbol{d}_{\text{vol}} / \boldsymbol{B} \rceil$. The memory requirement of an occupancy map is trivial for larger block sizes. For example, a block size of $\boldsymbol{B} = 4^3$ requires $1/64^{\text{th}}$ of the memory of an associated volume.

The occupancy map voxel at $\boldsymbol{u}_{\text{occ}}$ is associated with a volume block with extents:

$$\begin{aligned} \boldsymbol{u}_{\text{vol}}^{\min} &= \boldsymbol{B} \cdot \boldsymbol{u}_{\text{occ}} \\ \boldsymbol{u}_{\text{vol}}^{\max} &= \min(\boldsymbol{u}_{\text{vol}}^{\min} + \boldsymbol{B}, \boldsymbol{d}_{\text{vol}}) - 1 \end{aligned} \tag{5}$$

Texture sampling coordinates on the volume are mapped to the occupancy map according to

$$\begin{aligned} \boldsymbol{t}_{\text{occ}} &= \frac{\boldsymbol{d}_{\text{vol}}}{\boldsymbol{B} \cdot \boldsymbol{d}_{\text{occ}}} \boldsymbol{t}_{\text{vol}} \\ \boldsymbol{u}_{\text{occ}} &= \boldsymbol{d}_{\text{occ}} \cdot \boldsymbol{t}_{\text{occ}} = \frac{\boldsymbol{d}_{\text{vol}}}{\boldsymbol{B}} \boldsymbol{t}_{\text{vol}} \end{aligned} \tag{6}$$

This mapping is necessary as $\boldsymbol{B}$ may not evenly divide $\boldsymbol{d}_{\text{vol}}$ so $\boldsymbol{t}_{\text{vol}}$ and $\boldsymbol{t}_{\text{occ}}$ may not be equivalent.

The exact method of computing an occupancy map depends on whether it will be used for volume rendering or isosurface rendering. If volume rendering, a region is considered occupied if it contains any voxels with non-zero alpha as set by the transfer function. The occupancy map must be updated when the transfer function changes.

An isosurface exists in a region (and it is considered occupied) when the isosurface threshold lies between the minimum and maximum voxel values within the region. An isosurface may exist at the boundary between two regions so a single voxel halo around each region must also be considered when generating an occupancy map for isosurface rendering.

An occupancy map can be generated from a volume with a simple compute shader. Each shader invocation evaluates a unique volume block for occupancy and writes the result to the associated occupancy map voxel. The performance and limitations of such a simple compute shader is evaluated in Section 4.3.

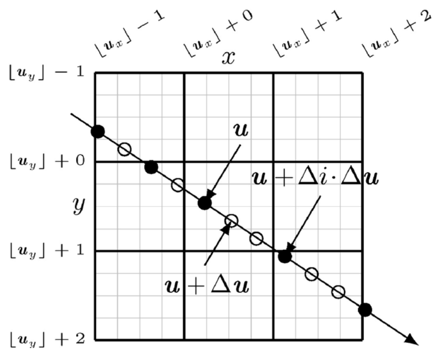
3.4 Efficient ray traversal

If an occupancy map voxel at $\boldsymbol{u}_{\text{occ}}$ indicates an empty volume block then the ray segment within that block can be skipped. Figure 1 shows a 2D schematic of a ray passing through an occupancy map. The sampling points along the ray are at $\Delta\boldsymbol{u}$ intervals. The number of steps, Δi, to reach a sampling point in the next region from $\boldsymbol{u}$ needs to be computed.

The first sampling point outside of the region on dimension $j \in \{x, y, z\}$ is

$$\Delta i_j = \begin{cases} \lceil (1 + \lfloor \boldsymbol{u}_j \rfloor - \boldsymbol{u}_j) / \Delta\boldsymbol{u}_j \rceil, & \text{if } \Delta\boldsymbol{u}_j > 0 \\ \lceil (\lfloor \boldsymbol{u}_j \rfloor - \boldsymbol{u}_j) / \Delta\boldsymbol{u}_j \rceil, & \text{otherwise} \end{cases} \tag{7}$$

which is determined geometrically from Fig. 1. This



$\lfloor \boldsymbol{u}_x \rfloor - 1$
$\lfloor \boldsymbol{u}_x \rfloor + 0$
$\lfloor \boldsymbol{u}_x \rfloor + 1$
$\lfloor \boldsymbol{u}_x \rfloor + 2$
x
$\lfloor \boldsymbol{u}_y \rfloor - 1$
$\lfloor \boldsymbol{u}_y \rfloor + 0$
y
$\lfloor \boldsymbol{u}_y \rfloor + 1$
$\lfloor \boldsymbol{u}_y \rfloor + 2$
$\boldsymbol{u}$
$\boldsymbol{u} + \Delta i \cdot \Delta\boldsymbol{u}$
$\boldsymbol{u} + \Delta\boldsymbol{u}$

Fig. 1 Schematic (2D) of a ray traversing a volume (grey grid) and its occupancy map (black grid) with $\boldsymbol{B} = (4, 4)$. The filled points are sampled on the occupancy map. Filled and unfilled points are sampled on the volume in occupied regions.

can be vectorised to give the steps on each dimension:

$$\begin{aligned} \Delta\boldsymbol{i} &= (\Delta i_x, \Delta i_y, \Delta i_z) \\ &= \lceil (\text{step}(\Delta\boldsymbol{u}) + \lfloor \boldsymbol{u} \rfloor - \boldsymbol{u}) / \Delta\boldsymbol{u} \rceil \end{aligned} \tag{8}$$

where step is the Heaviside step function. The first sampling point outside of the region surrounding $\boldsymbol{u}$ will be the minimum Δi_j, thus

$$\Delta i = \max\left(\overrightarrow{\min}(\Delta\boldsymbol{i}), 1\right) \tag{9}$$

which is clamped to a minimum of 1 as $\Delta\boldsymbol{u}$ will be infinite on one dimension for an axis-aligned ray.

The $\Delta\boldsymbol{u}$ for a ray passing through the occupancy map is computed as

$$\Delta\boldsymbol{u}_{\text{occ}} = \frac{\boldsymbol{d}_{\text{vol}}}{B} \Delta\boldsymbol{t}_{\text{vol}} \tag{10}$$

which follows from Eq. (6).

The occupancy map is repeatedly sampled and Eq. (9) applied to skip the ray forward until an occupied region is found. The volume is then sampled at each successive sampling point. A ray will skip more empty voxels for every occupancy map sample if $\boldsymbol{B}$ is larger. However, it will not be able to skip as close to occupied voxels as more empty voxels may be included in occupied regions. This suggests there may be an optimal $\boldsymbol{B}$ which will be explored in Section 4.2.

A sampling point near the edge of an empty block may map to a non-zero opacity if a neighbouring block is occupied due to trilinear interpolation of the volume. Similarly, an isosurface may exist on the boundary between an occupied and an empty region. To account for this, the ray step number is decremented after an occupied region is found according to

$$\Delta i_{\text{backward}} = -\lceil f \rceil \tag{11}$$

where f is the sampling factor from Eq. (2). This equation ensures the ray goes back at least one voxel length. The ray must not go back as far as the previous furthest i sampled on the volume.

The occupancy map is only sampled again if $\boldsymbol{u}_{\text{occ}}$ changes when i is incremented and the previously sampled voxel had zero opacity. Empty space skipping resumes as early as possible and the occupancy map is not sampled more than necessary. This method of resuming empty space skipping is faster than our original approach which resumed occupancy map sampling after a set length of empty voxels [6].

3.5 Chebyshev distance map

The Chebyshev distance metric is the greatest difference between two points along any dimension and is defined as

$$D_{\text{Chebyshev}}(\boldsymbol{p}_1, \boldsymbol{p}_2) = \overrightarrow{\max}(|\boldsymbol{p}_1 - \boldsymbol{p}_2|) \tag{12}$$

where $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are two points of interest. A *distance map* can be derived from an occupancy map representing the Chebyshev distance, D, to the nearest occupied block for every block. A ray can skip at least D blocks on any dimension before it may reach an occupied block.

The number of steps to the next sampling point D blocks away on dimension $j \in \{x, y, z\}$ is determined from Fig. 1 as

$$\Delta i_j = \begin{cases} \lceil (D + \lfloor \boldsymbol{u}_j \rfloor - \boldsymbol{u}_j) / \Delta \boldsymbol{u}_j \rceil, & \text{if } \Delta \boldsymbol{u}_j > 0 \\ \lceil (1 - D + \lfloor \boldsymbol{u}_j \rfloor - \boldsymbol{u}_j) / \Delta \boldsymbol{u}_j \rceil, & \text{otherwise} \end{cases} \tag{13}$$

Vectorising this equation gives:

$$\Delta \boldsymbol{i} = \lceil (\text{step}(-\Delta \boldsymbol{u}) + \text{sgn}(\Delta \boldsymbol{u}) \cdot D + \lfloor \boldsymbol{u} \rfloor - \boldsymbol{u}) / \Delta \boldsymbol{u} \rceil \tag{14}$$

which substitutes Eq. (8). This enables multiple blocks to be skipped with a single sample of the distance map. The number of skipped sampling points is not constrained by the size of $\boldsymbol{B}$ as with the occupancy map method.

Distance maps are not commonly used for ray casting acceleration because of the computational complexity of generating them which must be done every time the transfer function changes. Previous papers using Chebyshev distance maps for ray casting acceleration generated them at the full resolution of the volume with a serial algorithm [7, 8].

Our method of skipping rays forward with a Chebyshev distance map is simpler than previous approaches [7, 8] and supports lower resolution distance maps which reduces memory overhead and their computation time. In Section 4 we examine the impact on ray casting performance when using a reduced resolution distance map.

We have adapted the algorithm of Saito and Toriwaki [12] to efficiently transform an occupancy map into a Chebyshev distance map using the parallel capability of a GPU. The transformation is decomposed into three one-dimensional transformations which are all trivially parallelised as they operate on rows, columns, and then slices independently.

The first transformation is shown as a GLSL compute shader in Algorithm 2. This approach adapts the first transformation described in Ref. [12] for Chebyshev distance rather than squared Euclidean distance. The first transformation can safely read and write from the occupancy map if it is also bound as the distance map.

Algorithm 2 The first of the occupancy map to Chebyshev distance map transformations (GLSL)

```
// Shader globals
layout (local_size_y = ..., local_size_z = ...) in;
layout (..., r8ui) uniform uimage3D dist; // distance map
layout (..., r8ui) uniform uimage3D occ; // occupancy map
// with occupied voxels set to 255 and empty voxels 0

// First transformation
// Invoke for all columns and slices
void main() {
  // Initialisation
  ivec3 pos = ivec3(0, gl_GlobalInvocationID.yz);
  ivec3 dim = imageSize(dist);
  uint D1 = imageLoad(occ, pos).x;

  // Forward pass
  for (pos.x = 1; pos.x < dim.x; ++pos.x) {
    uint D = min(D1 + 1, imageLoad(occ, pos).x);
    imageStore(dist, pos, uvec4(D));
    D1 = D;
  }

  // Backward pass
  for (pos.x = dim.x - 2; pos.x >= 0; --pos.x) {
    uint D = min(D1 + 1, imageLoad(dist, pos).x);
    imageStore(dist, pos, uvec4(D));
    D1 = D;
  }
}
```

The second transformation is shown in Algoritihm 3 and has several differences from the original approach in Ref. [12]. The columns from transformation 1 are copied to a 1D buffer in the original approach. We instead read directly from the output of transformation 1 and write the output to a *swap image* with matching dimensions to the distance map. Another change is the transformation runs in a single pass and zigzags out from each voxel until the minimum distance is found. This reduces the number of IO operations and is faster than the original algorithm.

The third transformation is similar to the second but is applied on slices rather than columns. Algorithm 3 describes the minor changes to the second transformation for it to function as the third. Distances output from the second transformation are read from the swap image and the final Chebyshev distance is written to the distance map.

The distance map does not need to be a separate texture to the occupancy map as it can simply overwrite it. However, the swap image still doubles the overall memory requirements of this acceleration structure compared to just using an occupancy map. Distances are stored as unsigned 8-bit integers to

Algorithm 3 The second and third of the occupancy map to Chebyshev distance map transformations (GLSL)

```
// Shader globals
layout (local_size_x = ..., local_size_z = ...) in;
layout (..., r8ui) uniform uimage3D dist; // distance map
layout (..., r8ui) uniform uimage3D swap; // swap image

// Second transformation
// Invoke for all rows and slices
void main() {
  ivec3 pos = ivec3(gl_GlobalInvocationID);
  ivec3 dim = imageSize(dist);
  for (pos.y = 0; pos.y < dim.y; ++pos.y) {
    uint D = imageLoad(dist, pos).x;

    // Zig-zag out from pos in search of minimum D
    for (int n = 1; n < D; ++n) {
      if (pos.y >= n) {
        uint D_n = imageLoad(dist,
          ivec3(pos.x, pos.y - n, pos.z)).x;
        D = min(D, max(n, D_n));
      }
      if ((pos.y + n) < dim.y && n < D) {
        uint D_n = imageLoad(dist,
          ivec3(pos.x, pos.y + n, pos.z)).x;
        D = min(D, max(n, D_n));
      }
    }
    imageStore(swap, pos, uvec4(D));
  }
}

// Third transformation is 2nd transform with changes:
//  * Replace .y swizzles for .z and search on z axis.
//  * Read from swap and write to the distance map.
//  * Invoke for all rows and columns.
```

reduce memory and bandwidth requirements which limits the maximum distance to 255. This seems sufficient for most images given the distance map is at a lower resolution than the volumetric image being rendered.

3.6 Anisotropic Chebyshev distance map

A ray exiting an occupied region will initially only skip small distances as D is small. Ideally, the ray would skip a large distance when exiting an occupied region provided the next occupied region in the direction of the ray is far away. This can be accomplished using *anisotropic* Chebyshev distances.

Ray directions can be grouped into 8 octants in 3D defined by the sign of each component $(\pm x, \pm y, \pm z)$. Distance maps representing the distance to the nearest occupied region for each directional octant are needed. These can be generated by modifying the isotropic Chebyshev distance map transformations (Algorithms 2 and 3) to scan in single directions only.

The transformations could be applied in sequence to generate the distance maps for each directional octant which would require a total of 24 transformations. However, the first and second transformations only need to run 2 and 4 times respectively (rather than 8) as the intermediate distance maps they generate can be reused.

The ray caster simply needs to index the correct anisotropic distance map texture and can then apply the same ray traversal approach using Eq. (14) as described in Section 3.5. The index only needs to be computed once for each ray as the direction is fixed for the entire traversal.

Es and İşler [8] take the acceleration structure even further and compute the *extended* anisotropic Chebyshev distance. This represents the distance that can be skipped on the x, y, and z axis separately for each octant. This approach is effective for skipping empty spaces which are thin in some dimensions. However, it requires 24 times the memory of the original isotropic Chebyshev distance map approach described in Section 3.5. We did not implement this approach as the performance gains would likely be quite marginal or possibly worse given the higher memory bandwidth requirements.

4 Results and discussion

4.1 Testing methodology

Maximising rendering frame rates is the primary motivation for developing our ray casting method. This section will discuss the improvements in frame rate with the occupancy map and distance map acceleration structures described in the previous section. The update time for acceleration structures and their memory requirements are also important considerations in assessing the viability of our method.

Three publicly available [13] volumetric images with a range of dimensions were evaluated:

- a small image of a “present” with thin occupied layers,
- a “stag beetle” with large exterior empty regions and some internal empty space, and
- a large “king snake” with an internal skeleton composed of fine complex geometry.

The dimensions of each image are shown in Table 1.

The volumes were rescaled to an unsigned 8-bit representation and normalised before upload to the

Table 1 Dimensions of evaluated volumetric images and normalisation constants

Image	Dimensions	Voxels	n_0	n_1
Present	492×492×442	107M	0	4095
Stag Beetle	832×832×494	342M	0	2538
King Snake	1024×1024×795	833M	0	255

GPU using the equation:

$$v_{8\text{bit}} = \text{clamp}((v - n_0)/(n_1 - n_0) \cdot 255, 0, 255) \quad (15)$$

where n_0 and n_1 are image specific normalisation constants specified in Table 1.

Volume renderings of the evaluated images are shown in Fig. 2 which are rendered using our open source volume renderer [14]. The images have a wide range of occupied voxel proportions and are rendered with both 1D and 2D (gradient-based) transfer functions.

Voxels are mapped to an opacity (alpha value) with a simple equation:

$$\begin{aligned} a_{\text{value}} &= \text{clamp}((v - v_0)/(v_1 - v_0), 0, 1) \\ a_{\text{gradient}} &= \text{clamp}((g - g_0)/(g_1 - g_0), 0, 1) \quad (16) \\ a &= a_{\text{value}} \cdot a_{\text{gradient}} \end{aligned}$$

where v is the sampled voxel value, g is the gradient, and v_0, v_1, g_0, g_1 are parameters which define the transfer function. Voxel values and gradients are normalised between 0 and 1 from their underlying 8-bit representation when sampled.

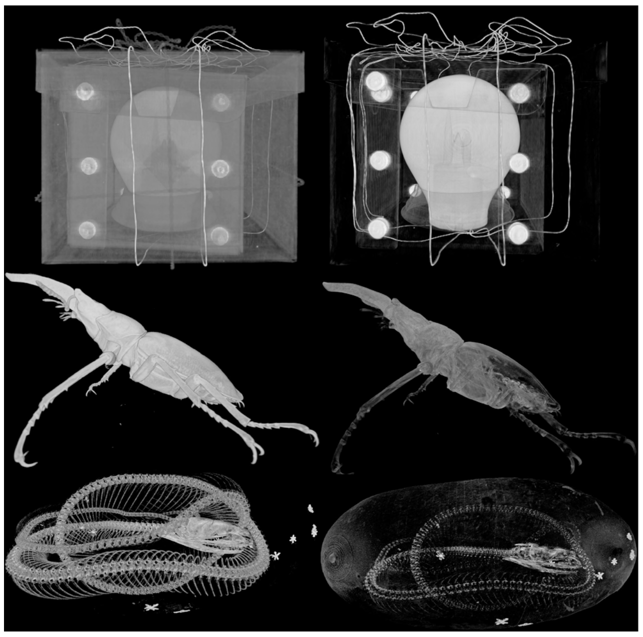


Fig. 2 Volume rendered images of the "present", "stag beetle", and "king snake" images from top to bottom. Rendered with 1D transfer functions (left) and 2D transfer functions (right).

Transfer functions are generally not defined by such simple functions. They are usually created through some complex user interaction and stored in a 2D texture for efficient lookup. For consistency with other volume renderers, Eq. (16) is applied to precompute the opacity for each unique intensity and gradient pair which is stored in a 256×256 texture. Colour is also stored and is set as greyscale with intensity proportional to the alpha value.

The transfer functions evaluated for each image are defined in Table 2. If g_0 and g_1 are not specified then the transfer function is 1D, $a_{\text{gradient}} = 1$, and the gradient is not sampled during ray casting.

Frame rates are measured by rotating volumes at a rate of 90° every 60 frames about their vertical axis. The average frame rate is taken over 1000 frames. Volumetric images were scaled to occupy 1 cubic meter and positioned at a distance of $\sqrt{3}$ meters from the camera in the virtual coordinate space. The camera has a perspective projection with a 1 radian horizontal and vertical FOV. In this configuration the entire volume is within the viewport and no clipping occurs.

Early ray termination acceleration is disabled for all tests which stops rays when an opacity limit is reached [1]. This ensures rays traverse all the way to back of the volume regardless of the transfer function. The sampling factor f from Eq. (2) is 1.

Baseline frame rates with no ray casting acceleration enabled are shown in Table 2. Images were rendered to a 1200×1200 viewport which is the approximate display resolution for a single eye on a modern virtual reality HMD. If the frame rates are halved to account for the second eye display then only the low resolution "present" image with a 1D transfer function renders fast enough without acceleration.

4.2 Frame rate improvements

Frame rates were measured for the images with each empty space skipping acceleration structure discussed

Table 2 Unaccelerated baseline frame rates for each image with a 1D and 2D transfer function (TF). Images are rendered to a 1200×1200 viewport with an NVIDIA GeForce GTX 1080

Image	TF	v_0	v_1	g_0	g_1	Occupancy	Frames (s^{-1})
Present	1D	0.071	1.0	—	—	7.13%	223
	2D	0.071	1.0	0.06	0.1	1.85%	66.8
Beetle	1D	0.086	1.0	—	—	3.97%	75.3
	2D	0.086	1.0	0.1	0.3	1.31%	19.6
Snake	1D	0.400	0.8	—	—	0.67%	28.1
	2D	0.200	0.8	0.06	0.12	0.55%	9.55

in Section 3 and are shown in Table 3. The maximum average frame rate with each acceleration structure is shown which is determined over a range of block sizes from 2^3 to 6^3. Using 2D transfer functions significantly reduces baseline frame rates even though precomputed gradients are used.

Table 3 reveals the occupancy map structure gives a substantial relative performance boost from the baseline frame rate in all cases. It is already fast enough to render most of the images on a HMD (except the snake with a 2D transfer function).

The frame rate is always improved when using an isotropic Chebyshev distance map compared to an occupancy map for acceleration. The improvement ranges from 4.5% for the “king snake” image with a 2D transfer function to 56% for the “beetle” image with a 1D transfer function.

Frame rates are improved by up to 20% with an anisotropic Chebyshev distance map compared to an isotropic distance map. However, for some images the performance improvement is negligible. The largest relative frame rate increase occurs on the snake image which shows that anisotropic Chebyshev distance maps are most effective for skipping empty space near fine complex geometry.

Figure 3 shows the relative frame rate compared to the baseline with each acceleration structure as a function of the block size. Frame rates with occupancy map ray traversal on the snake image worsen at larger block sizes. This can be explained by the high geometric complexity of the snake which results in a lot of empty space being included in occupied blocks at larger block sizes. Conversely, the beetle image benefits from larger block sizes when using an occupancy map as the large open regions exterior to the beetle can be skipped with fewer samples of the occupancy map.

Table 3 Volume rendering frame rates accelerated with an Occupancy Map (OM), Chebyshev Distance (CD), and Anisotropic Chebyshev Distance (ACD) maps

Image	TF	Occupancy	Frame rate (frames/s)			
			Base	OM	CD	ACD
Present	1D	7.13%	223	349	408	440
	2D	1.85%	66.8	271	298	301
Beetle	1D	3.97%	75.3	412	644	675
	2D	1.31%	19.6	204	239	239
Snake	1D	0.67%	28.1	195	228	255
	2D	0.55%	9.55	86.2	90.1	108

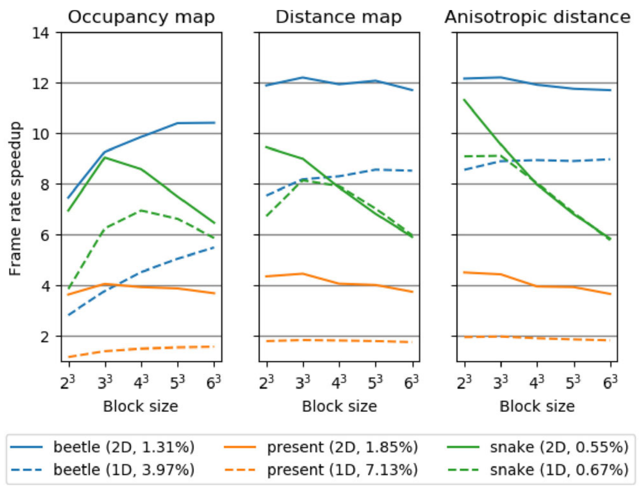


Fig. 3 Relative frame rate speedup from unaccelerated baseline for each acceleration structure with our ray traversal approach.

Frame rates are consistently improved with an isotropic distance map but degrade as the block size increases. This makes intuitive sense as rays are not able to skip as close to occupied voxels. A very small block size may not be practical as more memory and computation time is required (particularly for anisotropic distance maps). The additional performance with an anisotropic distance map compared to an isotropic distance map is quite marginal in most cases. Given this, it is unlikely that using extended anisotropic Chebyshev distance maps (see Section 3.6) would substantially improve frame rates.

Figure 4 qualitatively shows the number of texture samples with each ray casting acceleration structure. The difference gets progressively smaller between each acceleration structure. This indicates the approach may be reaching its performance limit and further suggests extended anisotropic distance maps may not provide meaningful performance gains.

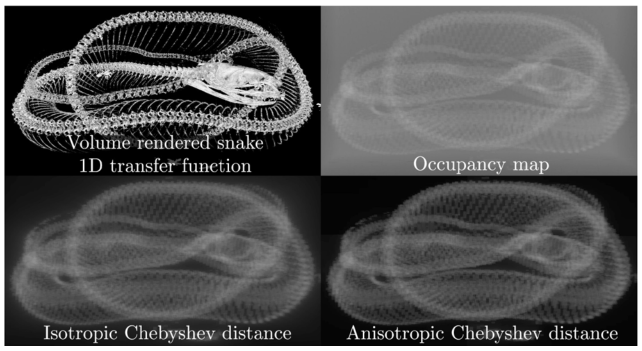


Fig. 4 Relative number of total texture samples (white is more) with each acceleration structure for the “king snake” image.

4.3 Distance map update time

Occupancy and distance maps must be updated whenever the transfer function is changed. The delay between adjusting a transfer function and seeing the resulting rendered image should be low to support interactive transfer function editing.

The time to generate occupancy maps for the evaluation images is shown for multiple block sizes in the left of Fig. 5 (note that the y scale is logarithmic). Occupancy maps can take much longer to generate with 2D transfer functions as more texture samples are required to determine the opacity of each voxel. The occupancy map update time worsens for block sizes over 4^3 as GPU resources are not as well utilised by the compute shader described in Section 3.3. The work needed for each individual occupancy map region could be divided up with a more complicated compute shader, although we did not explore this.

Isotropic or anisotropic Chebyshev distance maps are computed from the occupancy map for the faster ray casting approaches. The time to generate these distance maps (inclusive of the time taken to update the occupancy map) is also shown in Fig. 5 over a range of block sizes.

For a cubic block size of $B = (b, b, b)$, the occupancy and distance map have only $1/b^3$ of the number of volume voxels. Thus it is expected for the computation time to increase exponentially as b approaches 1 which is observed in Fig. 5. Distance maps are generated faster from occupancy maps with higher occupancies

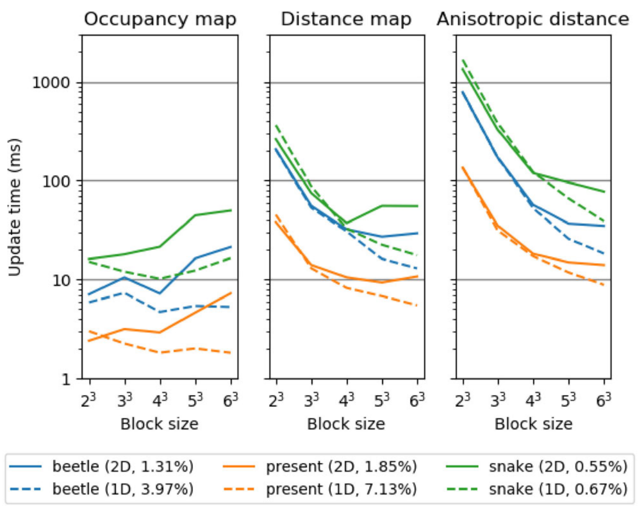


Fig. 5 Update time for occupancy and distance maps for the evaluated images with several cubic block sizes. Distance map update time is inclusive of the occupancy map update time.

as the average distance from each voxel that must be searched in Algorithm 3 is reduced.

The total time to generate distance maps is not necessarily minimised by choosing a large block size due to the overhead of generating the occupancy map. The isotropic Chebyshev distance is computed most efficiently when using a 2D transfer function with a block size of 4^3 or 5^3 as seen in Fig. 5. The isotropic distance map update time with a 4^3 block size on the large "king snake" image is only 37 milliseconds which enables highly interactive transfer function editing.

Anisotropic Chebyshev distance maps take much longer to compute than isotropic distance maps. Isotropic Chebyshev distance maps at 4^3 or 5^3 could be utilised for fast interactive transfer function prototyping. After a suitable transfer function has been identified, an isotropic or anisotropic distance map with a smaller block size could be generated to improve ray casting performance.

4.4 Limitations

In our previous paper a large sparse volume which did not fit directly in GPU memory was rendered by only allocating occupied regions on the GPU [6]. A distance map does not require much memory if the block size is large; however, this can limit potential performance gains as seen in Fig. 3.

Performance could be improved by using multi-resolution distance maps. Sparse higher resolution distance maps could be allocated only near occupied regions to enable rays to skip closer to occupied space without drastically increasing memory requirements. However, we have not explored methods of efficiently generating or effectively utilising sparse multi-resolution distance maps.

5 Conclusions

This work accelerates ray casting for volume or isosurface rendering by utilising a low resolution Chebyshev distance map for empty space skipping of sparse volumes. This work improves on the performance of our earlier approach [6] with more efficient methods of resuming empty space skipping and updating distance maps. Large sparse volumes can be rendered at high frame rates even with computationally expensive gradient-based transfer functions.

Distance maps can be generated at reduced resolutions with a low memory requirement and still provide excellent frame rate improvements. They can be updated in real-time enabling interactive transfer function editing for volume rendering.

Anisotropic Chebyshev distance maps were evaluated but the frame rate improvements were marginal in most cases. This suggests acceleration with distance maps may be reaching a performance limit. Faster sampling within occupied regions without compromising rendering quality is the next challenge for volume rendering acceleration.

Acknowledgements

This work is funded by the Australian Government Research Training Program (AGRTP) with additional support from the Australian National Laboratory for X-ray Micro Computed Tomography.

References

[1] Kruger, J.; Westermann, R. Acceleration techniques for GPU-based volume rendering. In: Proceedings of the 14th IEEE Visualization, 38, 2003.

[2] Stegmaier, S.; Strengert, M.; Klein, T.; Ertl, T. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In: Proceedings of the 4th International Workshop on Volume Graphics, 187–241, 2005.

[3] Beyer, J.; Hadwiger, M.; Pfister, H. State-of-the-art in GPU-based large-scale volume visualization. *Computer Graphics Forum* Vol. 34, No. 8, 13–37, 2015.

[4] Parker, S.; Shirley, P.; Livnat, Y.; Hansen, C.; Sloan, P.-P. Interactive ray tracing for isosurface rendering. In: Proceedings of the Visualization '98, 233–238, 1998.

[5] Hadwiger, M.; Sigg, C.; Scharsach, H.; Bühler, K.; Gross, M. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum* Vol. 24, No. 3, 303–312, 2005.

[6] Deakin, L.; Knackstedt, M. Accelerated volume rendering with Chebyshev distance maps. In: Proceedings of the SIGGRAPH Asia Technical Briefs, 25–28, 2019.

[7] Sramek, M.; Kaufman, A. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Transactions on Visualization and Computer Graphics* Vol. 6, No. 3, 236–252, 2000.

[8] Es, A.; İşler, V. Accelerated regular grid traversals using extended anisotropic chessboard distance fields on a parallel stream processor. *Journal of Parallel and Distributed Computing* Vol. 67, No. 11, 1201–1217, 2007.

[9] Gobbetti, E.; Marton, F.; Guitián, J. A. I. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer* Vol. 24, No. 7, 797–806, 2008.

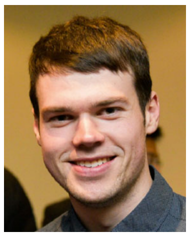
[10] Hadwiger, M.; Al-Awami, A. K.; Beyer, J.; Agus, M.; Pfister, H. SparseLeap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics* Vol. 24, No. 1, 974–983, 2018.

[11] Quilez, I. Normals for an SDF. 2018. Available at http://iquilezles.org/www/articles/normalsSDF/normalsSDF.htm.

[12] Saito, T.; Toriwaki, J.-I. New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. *Pattern Recognition* Vol. 27, No. 11, 1551–1565, 1994.

[13] Klacansky, P. Open scientific visualization datasets. 2019. Available at https://klacansky.com/open-scivis-datasets.

[14] Deakin, L. VkVolume. 2019. Available at https://github.com/LDeakin/VkVolume.



Lachlan J. Deakin (BEng Mechatronics, ANU) is a Ph.D. student at the Department of Applied Mathematics at the Australian National University. He previously worked for FEI and Thermo Fisher Scientific developing tools for the analysis of massive volumetric images. He specialises in high performance computing on clusters and GPUs.

Mark A. Knackstedt (BEng ChemEng, Columbia; Ph.D. ChemEng, Rice) is a professor at the Department of Applied Mathematics at the Australian National University. He has led a group working for 20 years in the field of digital materials technology based on 3D multiscale imaging, analysis, and modelling. The key paradigm of the technology is to "image and compute"—imaging the material, performing 3D time series imaging of experiments (e.g., flow, mechanical deformation), and building calibrated numerical simulations of the physical processes. He has helped to translate the technology into tangible commercial outcomes in the energy industry.