# An Efficient Shadow Detection Algorithm and the Direct Surface Rendering Volume Visualisation Model

by
**Mark W. Jones**
Department of Computer Science, University of Wales Swansea
Singleton Park, Swansea SA2 8PP, United Kingdom

**Abstract**

Research into volume graphics have lead us to new methods for the voxelisation of meshes and the ultra high quality rendering of volumetric data. Recently it has been reported by us [1] that our method of voxelisation and subsequent rendering can produce images faster than traditional ray tracing methods. In this paper we present in detail the *Direct Surface Rendering* algorithm and demonstrate how reflections are achieved. We have now extended this rendering method to produce shadows. The extension requires very little extra computation, and allows self-occlusion of objects. The algorithm should be adaptable for more traditional ray tracing, in particular suitable for octree encoded objects, CSG objects and with some precalculation, for most other objects.

**Keywords:** Volume visualisation, voxelisation, shadows, reflections, implicit surfaces, blobby models, rendering.

## 1 Introduction

Volume graphics are usually associated with poor image quality, large scene models, substantial computational complexity and limited application. Research into volume graphics at Swansea University has lead into exciting new areas that may one day yield the results that Jim Kajiya predicted with his statement at Siggraph '91 [2], "... in 10 years, all rendering will be volume rendering." Our research has included investigations into producing a method to realistically morph between two three dimensional data sets whilst retaining object coherency, automatic registration of features in three dimensions to aid the process, voxelisation in order to create the models, and a high quality rendering technique. We hope to apply these methods to the areas of forensic science [3], archaeology, entertainment [4] and medical imaging [5]. The motivation behind this particular work is the fact that a successful application of the above techniques would require a high quality rendering environment to be available. Such an environment needs to be firmly established to demonstrate the viability of these techniques.

The eventual outcome of this line of research is expected to be the provision of a rendering environment for volume graphics that achieves everything traditional rendering environments achieve. Every effect possible using alternative methods should be demonstrable in the volume graphic rendering environment. It is also the goal of this work that in moving to a volume graphic based framework, a time penalty should not be involved, and if possible a more rapid rendering environment should be provided.

Jim Kajiya's statement was based on the belief that volume rendering would be the only method to allow realistic scalability for extremely complex scenes. We believe that not only is this true, but also that volume rendering will be the only method that will allow the consistent application of visual effects to entire complex scenes. We first saw this in our work on voxelisation [1]. For a moderate scene voxelisation and direct surface rendering were quicker to apply than a traditional ray tracer. Voxelisation and direct surface rendering also allowed us to combine data from a scanned source (CT head) and a triangular mesh (chess piece), and apply a consistent shading and reflection model to both. Section 2 will look at the background for volume graphics and their rendering methods. Shadows will also be discussed. Section 3 introduces the direct surface rendering algorithm and describes in detail how the surface is produced, how reflections are produced, and the complexity of the algorithm. Section 4 shows how to achieve shadows in this rendering model. We conclude our work and give insight to possible future directions in Section 5.

## 2 Background

Volume data has been effectively visualised using two main approaches; surface polygonisation [6] and volume rendering [7, 8, 9]. The approaches are distinct insomuch as surface polygonisation treats the data as representing an implicit function and fits a surface approximation to a particular value. However, volume rendering attempts to display all values within the data by compositing sampled colours and opacities, created by a classification process, along the view rays. Essentially surface polygonisation has remained the same with subsequent approaches concentrating on tiling cubes or tetrahedra [10, 11, 12, 13] or removing ambiguities [14, 15]. Whilst the characteristic compositing function of volume rendering has been retained, improvements in the ray traversal algorithm produce images of varying degrees of accuracy for differing computational complexity [16, 17, 18, 19]. The main alternative for volume rendering is splatting [20, 21, 22]. The main problem with surface polygonisation is the fact that the surfaces produced can contain many primitives, and such surfaces are not suited for good high quality ray traced images. The main problem with volume rendering is the fact that when a surface is contained within the data we find that parts of the volume not belonging to that surface still contribute to it [23]. We also have the problem that if we wish to voxelise an object, that is to say convert it from one representation into volume data, we have a large computational overhead, and lose much of the accuracy of the original object.

Our direct surface rendering method overcomes these problems. We previously described how triangular mesh objects can be realistically voxelised and now in this paper we present the method we use to create high quality renderings of the volume data. In addition to handling reflections simply, we present a new algorithm for efficient shadow detection.

Shadow detection is very much a problem for ray tracing. For each surface hit, a ray must be cast to each light source to determine whether there are any obstructions. In the event of an obstruction, the surface is said to be in shadow for that particular light source, otherwise it is lit by that light source. This naive approach is extremely inefficient. One simple acceleration method is to use a *shadow cache* [24]. In this method one simply stores the last object which creates a shadow for a ray. The next ray is tested against this object first. In many scenes, rays close to each other will be occluded by the same object, and so this greatly increases efficiency. The main problem with this method is that when a surface tesselation is being ray traced, the primitives are so small that close rays may not be occluded by the same object. In these cases it is suggested that a *voxel cache* is used. In other words we retain a pointer to the last bounding box that contains the object that occluded a ray. One observation is that this only really works for primary rays. Secondary rays can travel in such vastly differing directions that the last object to occlude a ray probably will not occlude the secondary ray. As will be seen in Section 4 our method avoids these pitfalls.

## 3 Direct Surface Rendering

In this section a new technique for the production of high quality images from volume data is presented. This method follows the ray casting paradigm of Amanatides and Woo [19], and determines the surface normals using the grey-level gradient method. The main departure from previous methods is the fact that this method determines the exact surface boundary using tri-linear interpolation. The normal to the surface boundary is calculated using tri-linear interpolation from the gradients of the eight neighbouring voxels which make up the cube containing the resulting surface. The images resulting from this method are more accurate than those obtained using previous methods, and the surface can be rendered using high quality shading by using the accurate normals obtained directly from the grey-level data.

The surface is defined as a set consisting of all points in the volume space whose values are equal to a predefined threshold value $\tau$, and it may be composed of disconnected pieces. For each pixel in the image plane, a ray is cast into the volume and is traced through cubes on its path until it hits a transverse cube. A cube, bounded by eight voxels, is said to be transverse if at least one of its voxels is inside the surface and one outside the surface. Transverse cubes can be identified by comparing values of its bounding voxels against the threshold value $\tau$ and setting an eight bit flag, $b_8b_7b_6b_5b_4b_3b_2b_1$, as

$$b_i = \begin{cases} 0 & \text{if the value of voxel } i \leq \tau; \\ 1 & \text{if the value of voxel } i > \tau. \end{cases} \tag{1}$$

A cube is transverse if its flag has a value not equal to 0 or 255. All transverse cubes are determined in a pre-processing step to facilitate the use of the fast voxel traversal algorithm. Once a transverse cube is found, a further check is made to see if the ray intersects the surface contained within the cube. The values at the ray entry and exit points, namely $\alpha$ and $\beta$ respectively, are calculated using an interpolation function $f(\delta)$. Given a point $\delta$ lying on one of the six square facets of a cube, $f(\delta)$ is defined as the bilinear interpolation of the values associated with the four voxels which bound the facet. The ray intersects the surface if the ray span defined by $\alpha$ and $\beta$ is transverse, that is, $f(\alpha) \leq \tau \leq f(\beta)$ or $f(\beta) \leq \tau \leq f(\alpha)$ The actual intersection point $\gamma$ on the surface is calculated as

$$\gamma = \alpha + (\beta - \alpha)\left(\frac{f(\gamma) - f(\alpha)}{f(\beta) - f(\alpha)}\right) \tag{2}$$

Once the intersection point $\gamma$ has been found, the surface normal at $\gamma$, and hence the intensity, can be computed. The normal at $\gamma$ is calculated by tri-linearly interpolating from the gradients of the eight voxels bounding the cube. The gradient, $N$ at a voxel located at $(i, j, k)$ is calculated using difference approximation. With the normal, a shading technique, such as Phong shading, can be used to calculate the intensity at $\gamma$. Textures can also be mapped onto the surface for evaluating a second function in the data set. This is of particular use when each voxel is associated with a vector of samples. In computational fluid dynamics, for example, a surface representing constant pressure in a volume can be displayed with temperature mapped on to it.

By using the transverse cube information obtained in the pre-processing step, the fast voxel traversal algorithm allows a very quick traversal with just 2 floating point comparisons, 1 floating point addition, 1 integer addition, 2 integer comparisons for each transverse cube and a bit test and an integer comparison for each non-transverse cube. If a transverse cube $k$ is encountered, the entry point $\alpha_k$ is calculated and $f(\alpha_k)$ is linearly interpolated. A flag is set to indicate the entry point to the next cube to be examined, regardless of whether it is transverse or not. The ray is then continued into the next cube $k+1$ where again the entry point $\alpha_{k+1}$ and $f(\alpha_{k+1})$ are calculated. The ray entry and exit points and their associated values are now known for the ray span in cube $k$ and the surface intersection point can be determined. If the ray span is not transverse, the traversal continues with the next cube, for which the calculated entry point is already known. The result is a very efficient traversal of the empty parts of the volume, and a high quality rendering of the surface.

The algorithm has been implemented in C and tested on a 275MHz Alpha. Rendering times are shown for various sized images for the AVS Hydrogen data set in Table 1. Plotting time against the number of pixels in the image results in a linear function, suggesting the algorithm is perfectly scalable.

| Image size | Time taken (secs.) |
|---|---:|
| $100 \times 100$ | 0.65 |
| $400 \times 400$ | 10.25 |
| $800 \times 800$ | 40.582 |
| $1500 \times 1500$ | 143.244 |
| $2500 \times 2500$ | 398.534 |
| $4000 \times 4000$ | 1017.926 |

Table 1: Table showing rendering times for different sized images.

Reflections can be introduced by calculating the incident secondary ray from the known surface normal and incoming view ray. A secondary ray can then be projected into the volume. The images in this paper have all been produced with up to 10 levels of reflections allowed. (Figure 1(a) shows the UNC CThead rendered with reflections from three mutually orthogonal mirrors). A naive approach to shadows would be to project a shadow ray from each surface intersection in the direction of the light. If there is a surface intersection the point is in shadow, otherwise it is lit. The next section presents a more efficient way of calculating shadows.

## 4 Efficient Computation of Shadows

The problem with calculating shadows is the fact that a ray must be fired towards the light source which must be checked against every object in the scene until an intersection is found. This can be expedited by hierarchical bounding boxes, but still a significant cost remains.

The new algorithm requires that some decision has been made about each cube bounded by eight voxels – it is either inside, outside or contains the surface. Those cubes that contain the surface (transverse cubes) have already been found as part of the pre-processing step. Those cubes that are inside the surface can be found as part of the same pre-processing step.

The shadow algorithm is now simple. We track a ray through the volume from each intersection point towards each light source using the fast voxel traversal algorithm. At each step we check to see if the cube encountered is inside the surface. If it is the point is occluded and we need not make any further tests. If a ray does not encounter any cubes along its path that are wholly interior, we must now check to see if it passes through any transverse cubes. If not the point is lit. It is only when the ray has not passed through an interior cube, and has passed through a transverse cube that it must be determined if the ray has hit the surface in the manner described in the previous section. In this manner, shadows can be computed efficiently for any objects for which the interior can be determined on a regular grid, and is particularly suited to voxel data. It is therefore suited to octree encoded objects, CSG data, and by overlaying an arbitrary regular array of points, any object or scene of objects.

This extension to allow shadows was added to the basic direct surface rendering algorithm along with the reflection model. The resulting volume visualisation environment produces images such as Figures 1(b), 2(a), 2(b) and 3. In order to analyse the method it is useful to compare the results with a standard ray tracing package such as POV-Ray. The queen chess piece was rendered using POV-Ray with no shadows, and the results are shown in Table 2 along with the time to direct surface render the queen with several light sources. All times are for a $500 \times 500$ image.

| Computation | Time taken |
|---|---|
| Direct surface rendering | 23.266 secs. |
| With reflections | 41.182 secs. |
| With 1 light | 78.297 secs. |
| With 2 lights | 101.629 secs. |
| With 3 lights | 119.879 secs. |
| With 4 lights | 143.794 secs. |
| POV-Ray rendering | 120.000 secs. |

Table 2: Table showing direct surface rendering and POV-Ray comparison.

We have also applied the method of direct surface rendering to the rendering of implicit surfaces such as those created using blobby models. Current techniques either render the models by creating a mesh from the data which is then displayed (polygonisation [6]) or by intersecting each ray with the implicit surface using root-finding [25]. The direct surface rendering method offers a new efficient method for creating high resolution images from such data. Voxel data is first created (as in the polygonisation technique) but is then ray traced using the direct surface rendering technique rather than employing root-finding. This results in a great reduction of computation since it is no longer necessary to solve the complex equations involved in intersecting rays with complex implicit surfaces. Figure 2(b) shows some spherical blobs that have been ray traced using this method.

## 5 Conclusions

In this paper we have examined the existing methods for rendering volume models and computing shadows. We have presented the direct surface rendering method, and a new algorithm for the efficient computation of shadows. We have shown the direct surface rendering method as producing ultra high quality images, and when combined with the reflection model and shadow detection algorithm, it produces images that equal any that can be produced by standard existing techniques. We have shown that the direct surface rendering method is effectively scalable according to image size, and is quicker than conventional ray tracing such as that

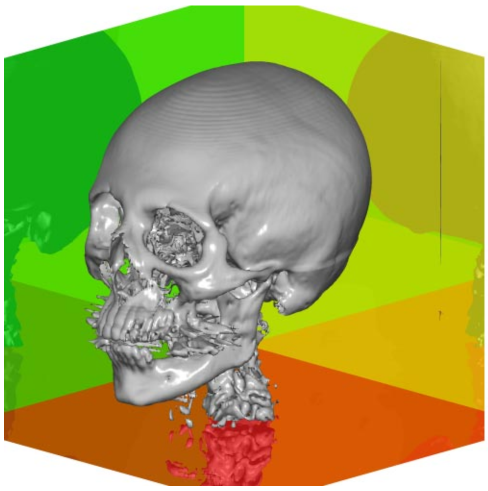
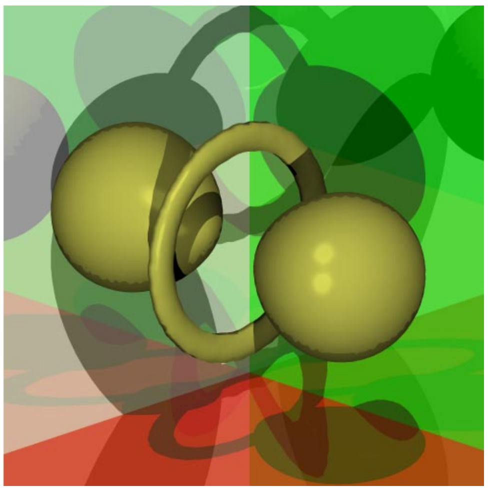

Figure 1: (a) Skull with reflections. (b) Hydrogen with shadows.

as provided by POV-Ray. The algorithm has been extended to allow reflections and shadows. The algorithm for shadow detection is efficient insomuch as a quick test is carried out, and a more accurate test is only carried out if needed. We hope the images provided with the paper demonstrate the application of the method. We also make the observation that the direct surface rendering technique is also suitable for accurately ray tracing blobs, and show an example of such a case.

The method is viewed as a major advance towards the construction of a rendering environment for volume data. We believe the future direction of this work should concentrate on even better realism, in particular combining the model with the radiosity technique.

## References

[1] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, December 1996.

[2] T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.

[3] S. D. Michael and M. Chen. The 3D reconstruction of facial features using volume distortion. In *Proceedings of 14th Annual Conference of Eurograph-*

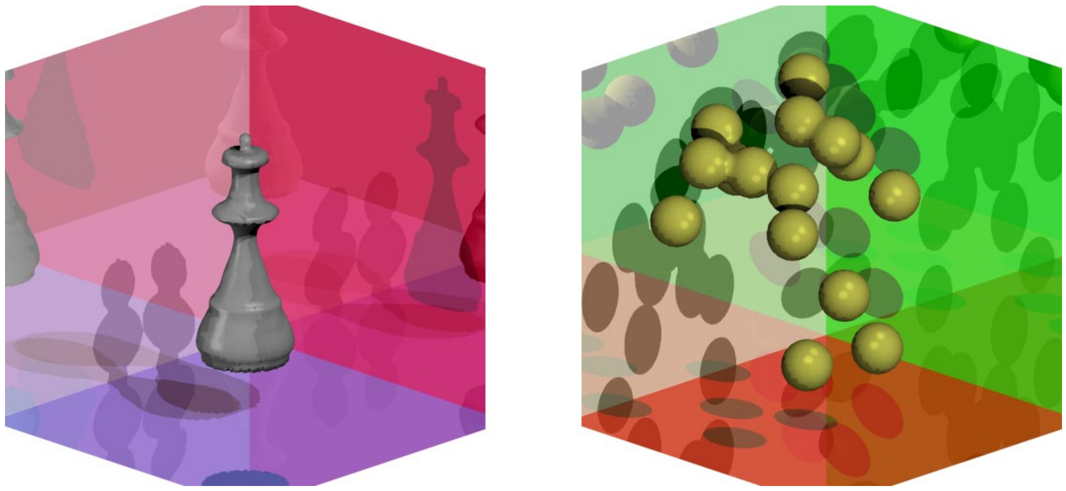
Figure 2: (a) Queen with shadows. (b) Blobs with shadows.

*ics (UK Chapter) (Imperial, March 26-28, 1996)*, pages 297–305, March 1996.

[4] M. Chen, M. W. Jones, and P. Townsend. Methods for volume metamorphosis. In Y. Parker and S. Wilbur, editors, *Image Processing for Broadcast and video production*. Springer-Verlag, London, 1995.

[5] M. Chen, M. W. Jones, and P. Townsend. Volume distortion and morphing using disk fields. *Computers and Graphics*, 20(4), 1996.

[6] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2:227–234, 1986.

[7] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 51–57. ACM SIGGRAPH, New York, August 1988.

[8] L. Carpenter R. A. Drebin and P. Hanrahan. Volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 65–74. ACM SIGGRAPH, New York, August 1988.

[9] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[10] W. E. Lorensen and H. E. Cline. Marching Cubes : A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87 (Anaheim, Calif., July 27-31, 1987)*, volume 21(4), pages 163–169. ACM SIGGRAPH, New York, July 1987.

[11] B. A. Payne and A. W. Toga. Surface mapping brain function on 3D models. *IEEE Computer Graphics and Applications*, 10(5):33–41, September 1990.

[12] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[13] C. Zahlten and H. Jürgens. Continuation methods for approximating isovalued complex surfaces. In *Computer Graphics Forum, Proc. Eurographics '91*, volume 10(3), pages 5–19. University Press, Cambridge, UK., September 1991.

[14] J. Wilhelms and A. V. Gelder. Topological considerations in isosurface generation – Extended abstract. *Computer Graphics*, 24(5):79–86, November 1990.

[15] G. M. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In *Proc. Visualization 91*, pages 83–90. IEEE CS Press, Los Alamitos, Calif., 1991.

[16] R. Yagel and A. Kaufman. Template-based volume viewing. In *Computer Graphics Forum, Proc. Eurographics '92 (Cambridge, UK, September 7-11, 1992)*, volume 11(3), pages C–153–C–167. University Press, Cambridge, UK., September 1992.

[17] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[18] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6:2–7, 1990.

[19] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. *Proc. of Eurographics '87, Amsterdam, The Netherlands*, pages 3–9, August 1987.

[20] L. Westover. Footprint evaluation for volume rendering. In *Proc. SIGGRAPH '90 (Dallas, Texas, August 6-August 10, 1990)*, volume 24(4), pages 367–376. ACM SIGGRAPH, New York, August 1990.

[21] R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proc. Visualization 93*, pages 261–266. IEEE CS Press, Los Alamitos, Calif., 1993.

[22] D. Laur and P Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Proc. SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991)*, volume 25(4), pages 285–288. ACM SIGGRAPH, New York, July 1991.

[23] U. Tiede, K. H. Höhne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3D-rendering algorithms. *IEEE Computer Graphics and Applications*, 10(2):41–53, March 1990.

[24] A. Pearce. A recursive shadow voxel cache for ray tracing. In *Graphics Gems II*, pages 273–274. Academic Press, 1991.

[25] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proc. SIGGRAPH '94 (Orlando, Florida, July 24-July 29, 1994)*, volume 28(2), pages 269–277. ACM SIGGRAPH, New York, July 1994.

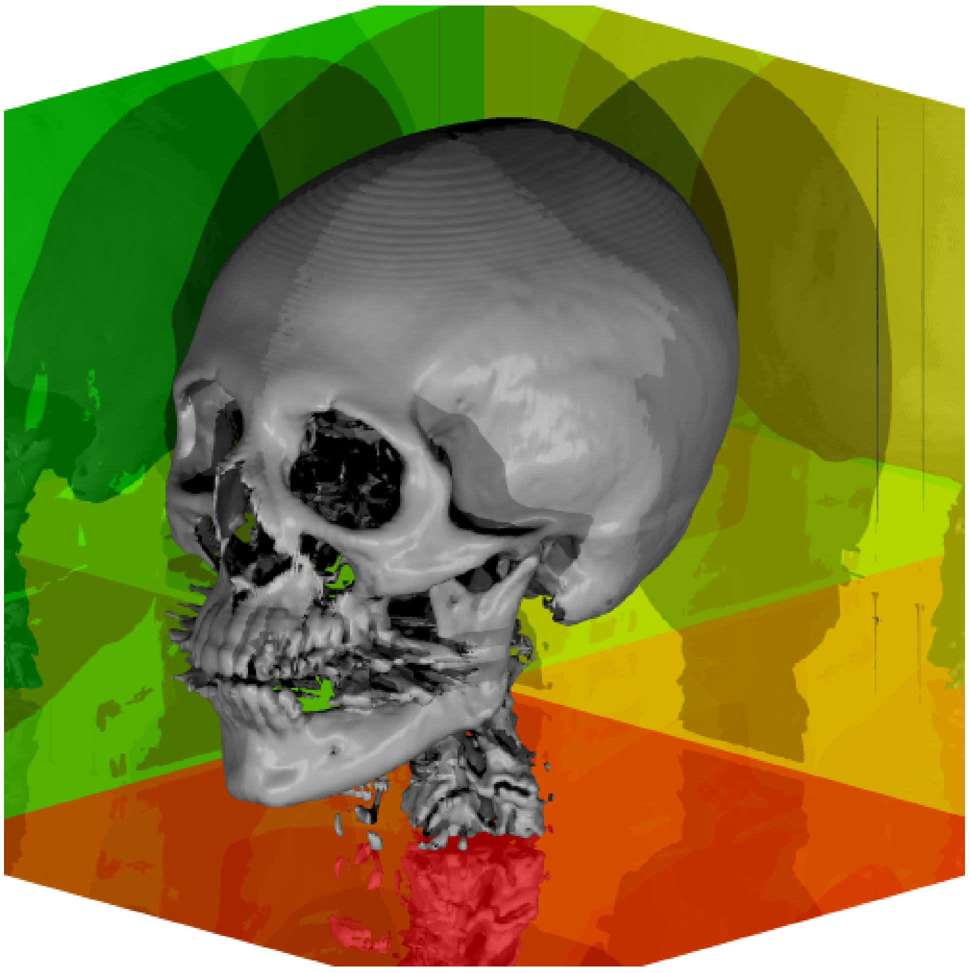
Figure 3: Skull with shadows and reflections.