Volumetric Ray Tracing

1062[1]

[1]OTH Regensburg, Germany

Abstract

—START OF REVIEW—

1. Introduction

In Computer Graphics, objects usually are represented as a set of geometric primitives (e.g. triangles), displaying the surface of the object in question. However, this approach is not always suitable. For example, if the original data representation of the object is volumetric data (which might be produced by medical 3d scans [Lev88]), the traditional rendering technique would necessitate the creation of an intermediate surface representation that might introduce unwanted artifacts. Another issue arises if the object has no well-defined surfaces to which geometric primitives could be fitted, such as a cloud [KVH84]. In such cases, volumetric ray tracing might be used, a technique in which rays are cast through a volume which contains information about color and opacity, sampled at various points within the volume, accumulated and projected on a 2d image (see figure 1) [Lev88]. A description of this algorithm is presented in this paper, alongside several strategies for improving computational speed, such as (probabilistic) early termination of a ray, and sampling at lower resolutions within the volume.

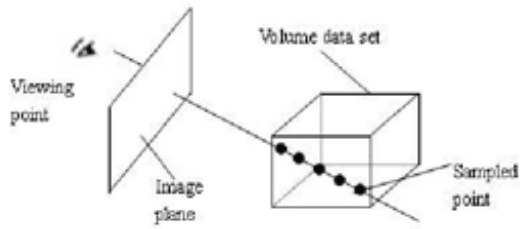

Figure 1. Ray Casting scheme [App15]

Figure 1: *Illustration of the ray casting process.*

2. Basic Volumetric Ray Tracing Algorithm

2.1. Underlying Volume Data

In this paper, we consider the topic of synthesizing a 2d image from a 3d volume. A volume can either be some continuous, ternary function (such as perlin or simplex noise [Per85], which can be used to render volumetric clouds [Häg18]), or a discrete 3d array [Lev88]. Similar to a 2d image that consists of pixels that can be addressed by a 2d vector $\vec{x} \in \mathbb{N}^2$, a 3d array consits of voxels that are adressed by a 3d vector $\vec{x} \in \mathbb{N}^3$. The color of a voxel at position $\vec{x}$ is called $c(\vec{x})$, the opacity $\alpha(\vec{x})$. The opacity is a scalar value, the color may also be a scalar value (for grayscale volumes) or a 3d vector (for colorful volumes). In the following, all values are assumed to be normalized to between 0 and 1. To project this volume to a image, for each pixel in the image a ray is cast through the volume and sampled at multiple, evenly spaced points on the ray (see figure 1) [Lev90].

2.2. Interpolation

The sample points on the ray are in general part of $\mathbb{R}^3$. This is no problem if the volume is described by a continuous function, which is defined everywhere. However, if it is a discrete 3d array only defined for points in $\mathbb{N}^3$, the value of the volume at the sample point must be interpolated. This interpolation is done over the 8 closest voxels to the sample point (usually with a trilinear interpolation). The opacity values can be interpolated regularly, but the color values must be weighted with the respective opacity values before interpolation [WMG98]. To see why this is necessary, consider figure 2, which presents a simplified 2d example of a completely opaque, white object behind completely transparent empty space. Naively interpolating in this volume would result in two sample points, one completely white and opaque, the other gray and semitransparent. This gray point is not present in the original data and is an unwanted artifact. Weighting the colors with their opacity prevents this from happening.

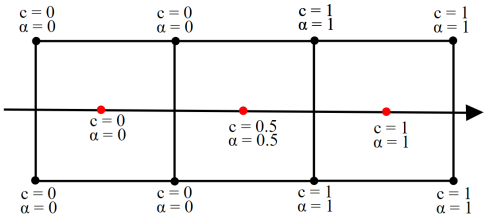

Figure 2: *Simplified visualization of naive interpolation in 2 dimensions. The black dots represent the voxels, the red dots the sample points. Notice that the second sample point has a gray color, even though the original data only has white voxels and blck, transparent voxels, which shouldn't affect the final color. Adapted from Wittenbrink et al. [WMG98]*

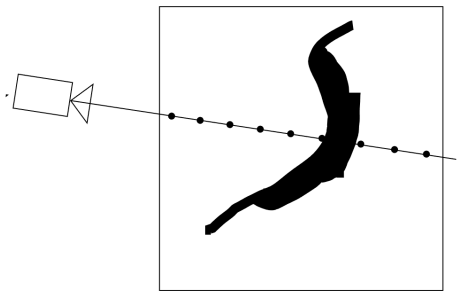

Figure 3: *The black, opaque object blocks parts of the volumes behind it. The samples behind the object (from the cameras perspective) are not visible and therefore irrelevant.*

2.3. Compositing Of Multiple Sample Points

To compose the various sampled points on the ray to a single pixel, the points one after the other are alpha blended together. This compositing can be done front to back [Sab88, UK88] (starting with the sample point closest to the camera, blending it with the second, then the third, and so on), or back to front [DCH88, Lev88] (vice versa). Usually, the back to front approach is chosen since it allows to optimize the computation [Lev90] (see below). In the following, $c(i)$ is the color of the i-th sample point, and $\alpha(i)$ the opacity. The accumulated opacity

$$\beta(i) = 1 - \prod_{j=1}^{i}(1-\alpha(j)) \tag{1}$$

is the fraction of light absorbed between the first and i-th sample point. The final color c_f, which is displayed in the projected image, is then

$$c_f = \sum_{i=1}^{n}((1-\beta(i)) * c(i)) \tag{2}$$

if the ray has n sample points [DH92].

3. Optimization Strategies

3.1. Adaptive Termination Of Ray Tracing

When looking at the compositing process described above, it becomes apparent that the contribution of a sample point to the final result decreases, the more other opaque sample points lie before it [Lev90]. For an extreme example, consider figure 3. Here, all sample points behind the fully opaque black wall contribute nothing to the final value and can therefore be ignored. In other words, going back to equation (2), the ray casting can be terminated after the accumulated opacity reaches 1, without changing the image quality. If some errors in the image are acceptable, the threshold value for terminating the ray tracing might be chosen to be somewhat lower.

3.2. Ray Termination With Russian Roulette

The above described termination after a certain threshold is reached creates a bias in the synthesized image [AK90]. One way to avoid this is to use Russian Roulette to decide when to terminate a ray [DH92]. Unlike the above described approach, Russian Roulette terminates a ray only with a certain likelihood once the accumulated opacity reaches the threshold. The weight of the surviving rays is increased proportionally to compensate for the terminated rays.

3.3. Pyramid Data Structures

The following optimizations require a data structure called a pyramid [DH92]. Similar to a set of mip-maps with decreasing resolutions, a pyramide is a set of volumes with decreasing resolutions. The different volumes of which the pyramid consists are called the levels of the pyramid. The lowest level (called level 0) is equal to the original volume. In general, level $n+1$ is half the size of level n, and each voxel in level $n+1$ contains some information about 8 voxels in the lower volume [Lev90]. What information exactly is stored in the levels depends on what specific pyramid is used. There are different kinds of pyramids that can be used, the first we consider is the so called average pyramid, which, as the name suggets, stores the average of the 8 lower voxels in each voxel of the higher level (see figure 4). —END OF REVIEW—

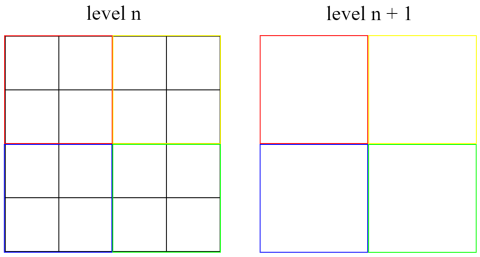

Figure 4: *Simplified 2 dimensional example of 2 levels of a pyramid. Each pixel in level n +1 contains 4 pixel of the lower level. In 3 dimensions, 8 voxels of the lower level are contained in one voxel of the upper level.*

3.4. Presence Acceleration

References

[AK90] ARVO J., KIRK D.: Particle transport and image synthesis. *SIGGRAPH Comput. Graph. 24*, 4 (sep 1990), 63–66. URL: https://doi.org/10.1145/97880.97886, doi:10.1145/97880.97886. 2

[App15] APPA S.: Ray casting for 3d rendering – a review. 1

[DCH88] DREBIN R. A., CARPENTER L., HANRAHAN P.: Volume rendering. *SIGGRAPH Comput. Graph. 22*, 4 (jun 1988), 65–74. URL: https://doi.org/10.1145/378456.378484, doi:10.1145/378456.378484. 2

[DH92] DANSKIN J., HANRAHAN P.: Fast algorithms for volume ray tracing. In *Proceedings of the 1992 Workshop on Volume Visualization* (New York, NY, USA, 1992), VVS '92, Association for Computing Machinery, p. 91–98. URL: https://doi.org/10.1145/147130.147155, doi:10.1145/147130.147155. 2

[Häg18] HÄGGSTRÖM F.: Real-time rendering of volumetric clouds, 2018. 1

[KVH84] KAJIYA J. T., VON HERZEN B. P.: Ray tracing volume densities. *SIGGRAPH Comput. Graph. 18*, 3 (jan 1984), 165–174. URL: https://doi.org/10.1145/964965.808594, doi:10.1145/964965.808594. 1

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37. doi:10.1109/38.511. 1, 2

[Lev90] LEVOY M.: Efficient ray tracing of volume data. *ACM Trans. Graph. 9*, 3 (jul 1990), 245–261. URL: https://doi.org/10.1145/78964.78965, doi:10.1145/78964.78965. 1, 2

[Per85] PERLIN K.: An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (jul 1985), 287–296. URL: https://doi.org/10.1145/325165.325247, doi:10.1145/325165.325247. 1

[Sab88] SABELLA P.: A rendering algorithm for visualizing 3d scalar fields. *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988). 2

[UK88] UPSON C., KEELER M.: V-buffer: visible volume rendering. *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988).

[WMG98] WITTENBRINK C., MALZBENDER T., GOSS M.: Opacity-weighted color interpolation for volume sampling. In *IEEE Symposium on Volume Visualization (Cat. No.989EX300)* (1998), pp. 135–142. doi:10.1109/SVV.1998.729595. 1, 2