A Lipschitz Method for Accelerated Volume Rendering[$]

Barton T. Stander John C. Hart

School of EECS
Washington State University
Pullman, WA 99164-2752
{bstander,hart}@eecs.wsu.edu

Abstract

Interpolating discrete volume data into a continuous form adapts implicit surface techniques for rendering volumetric iso-surfaces. One such algorithm uses the Lipschitz condition to create an octree representation that accelerates volume rendering. Furthermore, only one preprocessing step is needed to create the Lipschitz-octree representation that accelerates rendering of iso-surfaces for any threshold value.

Keywords: volume rendering, ray casting, Lipschitz condition, octree.

1 Introduction

Due to the immense size of volumetric data, volume rendering is a notoriously slow process. Preprocessing raw volumetric data into a more efficient representation, such as a polygonal surface [14, 9] or an octree [8], accelerates volume rendering of specific isovalue surfaces. We concentrate on accelerating the ray-casting technique for volume rendering [7] as opposed to the "splatting" technique [3]. This paper adapts implicit surface ray-tracing algorithms [6, 4] for preprocessing volumetric data to accelerate iso-surface ray-casting for general threshold values.

Section 2 outlines relevant methods currently used for accelerating the rendering of volumetric data and implicit surfaces, and defines the Lipschitz condition. Section 3 applies a trilinear interpolation function to volumetric data, computes a Lipschitz bound for the interpolation function, and merges the resulting Lipchitz bounds into an octree. Section 4 explains how the Lipschitz condition can be used to accelerate ray traveral, and presents our algorithm for accelerated volume rendering of any iso-surface using Lipschitz methods. Section 5 documents the results of the implementation and Section 6 outlines directions for further research.

[$]Research supported in part by NSF under grants #CCR-9309210 and #CDA-9121675.

2 Previous Work

The following discussions summarize some of the many algorithms relevant to the Lipschitz-based volume rendering method described in the rest of the paper. These include techniques for rendering both volumetric isosurfaces and implicit surfaces.

2.1 Volume Rendering

Volume rendering discussions often utilize the duality of voxels and cells. Voxels are points whereas cells are regions. Eight adjacent voxels in a cubical lattice define the corners of a cubic cell. Furthermore, cells may be grouped together to form larger regions. The voxels in the following examples have been classified and have an opacity associated with them, where an opacity of zero indicates complete transparency. The opacity in a cell is an interpolation of the voxels it contains. A cell is transparent if and only if all of its voxels have an opacity of zero. The following algorithms accelerate volume rendering by culling large regions of transparent cells from consideration during ray traversal.

2.1.1 Octree-Based Volume Rendering

Levoy [8] uses an octree to accelerate volume rendering of a specific classification V of opacities. This method constructs a second binary volume $B_0 : \mathbf{Z}^3 \rightarrow \{0,1\}$ of cell resolution $X \times Y \times Z$ consisting of entries corresponding to the cells of the opacity volume $V : \mathbf{Z}^3 \rightarrow \mathbf{Z}$ of voxel resolution $(X+1) \times (Y+1) \times (Z+1)$. For simplicity, we assume the original volume is padded such that I, J and K are powers of two, and $I = J = K$. The binary value of an entry in B_0 is zero if and only if the cell it corresponds to in V is transparent. An octree of binary cell volumes $B_m, (0 \leq m \leq M)$ is defined then by

$$B_m(x, y, z) = \max_{i,j,k \in \{0,1\}} B_{m-1}(2x + i, 2y + j, 2z + k). \quad (1)$$

from level B_1 up to a top level B_M consisting of a single cell bounding the entire volume.

Ray traversal begins at the top level $m = M$ of the octree. As the ray enters the current cell in B_m, the cell's value is checked. If zero, then the entire cell is transparent, none of its voxels need be checked, and ray traversal proceeds to the next cell in B_m if it shares the same parent, and if not, then ray traversal proceeds to the parent of the next cell, in B_{m+1}. In the other case where the value of the current cell in B_m is one, if $m > 0$ then the cell is subdivided into eight smaller cells from B_{m-1}. Otherwise we are at the bottom level of the octree and the opacity of the current cell is sampled.

This algorithm saves time by culling large transparent portions of the volume from consideration, and accelerates rendering times between 20% and 50% of the ray-marching rendering times [8]. On the down side, the octree must be reconstructed every time the isosurface threshold is altered, which increases the accelerated rendering time by an additional 32%.

2.1.2 Min-Max Octrees

Min-max octrees provide the spatial subdivision advantages of an octree without the overhead of regenerating the structure whenever the isosurface threshold is changed. Min-max octrees were first used in volume rendering to enhance the polygonization speed of an isosurface [13] and were later surveyed as a multiresolution acceleration technique, called homegeneity-acceleration, for volume ray tracing [2].

A min-max octree consists of two octrees. Each cell in the $B^+_m, 0 \leq m \leq M$ octree consists of the maximum value of its children whereas each cell in the $B^-_m, 0 \leq m \leq M$ octree consists of the minimum value of its children. Ray intersection proceeds as described in the previous section, except that one subdivides an octree cell if and only if the threshold T is no greater than the B^+ cell value and no less than the B^- cell value.

The min-max octree structure was shown to accelerate ray tracing by 30% of the speed of simple ray marching [2].

2.1.3 Ray Acceleration by Distance Coding

Zuiderveld *et. al.* [16] also use a binary volume B whose entries correspond to cells in the opacity volume V, and an entry in B is zero if and only if its corresponding cell in V is transparent. Their method then applies a distance transform to B, creating D, a volume whose entries consist of a pseudo-Euclidean distance from that cell to the nearest corresponding non-transparent cell in V. Details of the distance transform can be found in [1]. A faster but more space consumptive version of the distance transform is described in [16] as adapted from [10].

Ray traversal begins at the point where the ray enters the volume. At a given point on the ray, the distance volume D yields the distance d from that point to the nearest non-transparent cell. If $d > 0$, then for a length of d, the ray intersects only transparent cells, which may be culled from consideration, and traversal continues at a point d units further along the ray. If $d = 0$ then the current cell is not transparent and its opacity is sampled from V.

Like the previous octree method, distance coding enhances volume rendering by providing effective space leaping, accelerating the rendering time to 28% of the ray marching time [16]. The down side is the time and space expense of recomputing the distance transform for each new isovalue threshold, which increases the accelerated rendering time by an additional 85%.

2.2 Lipschitz Methods for Rendering Implicit Surfaces

A map between metric spaces $f : \mathbf{D} \to \mathbf{R}$ is *Lipschitz* if and only if

$$||f(x) - f(y)|| \leq L||x - y|| \quad \forall x, y \in \mathbf{D} \quad (2)$$

for some positive L. A function is Lipschitz if its derivative is bounded. The minimum L satisfying (2) over a given domain $D \in \mathbf{D}$ is known as the *Lipschitz constant* of f, denoted $\text{lip}_D f$. In practice, one usually needs only a *Lipschitz bound,* a value $L \geq \text{lip } f$ such that (2) holds.

The spatial Lipschitz condition is used to accurately triangulate deformed, intersecting surfaces in [11] and a spatio-temporal Lipschitz condition is used to detect collisions of parametric surfaces in [12]. A temporal Lipschitz condition is used for accelerating the rendering of an animated sequence of volumes in [15].

Of particular interest are applications of the Lipschitz condition for rendering implicit surfaces, such as [6] and [4]. An implicit surface is defined by a function $f : \mathbf{R}^3 \to \mathbf{R}$ as the set of points $\{\mathbf{x} : f(\mathbf{x}) = T\}$ for some threshold T. Our algorithm adapts these implicit surface rendering techniques to volumetric data. These techniques are described in the following sections.

2.2.1 L-G Surfaces

Kalra and Barr [6] use the Lipschitz condition of a function and its gradient to guarantee correct ray intersections with its implicit surface. The method consists of a root finding algorithm which is made more efficient by an octree structure.

The octree structure is based on a domain-specific Lipschitz bound. If (but not "and only if") the value at one corner of an octree cell C is below the threshold T whereas the value at another corner is above the threshold T, then the cell "straddles" the implicit surface. Otherwise, let L be a Lipschitz bound over the domain C, let point $\mathbf{x}_0 \in C$ be its centroid and let d be the distance from $\mathbf{x}_0$ to the farthest corner of C. Then the condition

$$|f(\mathbf{x}_0) - T| > Ld \qquad (3)$$

guarantees that the octree cell C is either entirely interior or entirely exterior with respect to the implicit surface, and may be culled from consideration when intersecting a ray with the implicit surface. If (3) fails, then the octree cell is subdivided and the process recurses on its children until all octree cells either straddle or do not intersect the implicit surface, or the cells subdivide down to some terminal level.

The intersection of a ray with the implicit surface lies within a straddling octree cell, and is found using a Lipschitz bound of the directional derivative of the implicit function with respect to the ray direction. This Lipschitz condition allows them to determine if the implicit surface intersects an interval along the ray zero, one or possibly several times. In the latter case, the interval is subdivided and the Lipschitz condition reapplied until the answer is zero or one. Once isolated in a monotonic region, a root refinement method such as Newton's method or *regula falsi* quickly finds the root.

2.2.2 Sphere Tracing

Hart [4] presents a similar algorithm requiring only the Lipschitz bound of the function defining the implicit surface. Equation (3) can be reformulated to state that the distance from a point $\mathbf{x}_0$ to the implicit surface of f is bound by

$$d(\mathbf{x}_0, f^{-1}(T)) > (f(\mathbf{x}_0) - T)/L. \qquad (4)$$

Ray intersection then progresses in a fashion similar to the distance transform volume rendering acceleration method of [16].

3 The Lipschitz-Octree Structure

The Lipschitz-octree structure combines the speed enhancements of distance coding with the threshold flexibility of min-max octrees. The Lipschitz-octree method is based on L-G surfaces and sphere tracing. These implicit surface rendering methods are adapted to volume data by redefining the discrete voxel data as a continous scalar field, over which a local Lipschitz bound can be computed. These local Lipschitz bounds are repeatedly merged to form an octree of Lipschitz bounds over variously-sized domains, ending at the top-level with a global Lipschitz bound on the entire volume.

3.1 Defining Discrete Volumes Continuously

Our volume rendering algorithm adapts the two implicit surface rendering algorithms described in Section 2 to discrete data. Given a lattice of sampled volume data $V : \mathbf{Z}^3 \rightarrow \mathbf{Z}$ one reconstructs a continuous signal by interpolating discrete data points. The reconstructed volume $v : \mathbf{R}^3 \rightarrow \mathbf{R}$ matches V at lattice points, and returns interpolated values otherwise. Hence, the problem of rendering an isovalued surface from volume data becomes the problem of rendering the implicit surface $v(x, y, z) = T$.

The choice of interpolation impacts the analytic properties of the resulting continuous reconstruction. Nearest neighbor interpolation is not continuous and hence is not Lispchitz. Trilinear interpolation is continuous but is not smooth across cell boundaries. These creases can cause problems for typical root finding algorithms, such as the one used in L-G surfaces [6] which requires first derivative continuity. However, algorithms using only the Lipschitz bound, such as sphere tracing [4], converge cleanly to surfaces in continuous but creased spaces.

3.2 A Lipschitz Bound for a Single Cell

For simplicity, consider a single cell C with voxels V_{000} and V_{111} at its corners. Using trilinear interpolation, the value within C is defined using the values of the eight corner voxels V_{ijk} as

$$v(x, y, z) = \sum_{i,j,k \in \{0,1\}} (1-x)^{1-i} x^i (1-y)^{1-j} y^j (1-z)^{1-k} z^k V_{ijk}. \qquad (5)$$

for real $x, y, z \in [0, 1]$.

The Lipschitz constant $\operatorname{lip}_C v$ over the domain of cell C is the maximum gradient magnitude within C, which is bound by the magnitude of the maxima of the individual gradient components

$$\operatorname*{lip}_C v \leq \sqrt{\max_C \left(\frac{\partial v}{\partial x}\right)^2 + \max_C \left(\frac{\partial v}{\partial y}\right)^2 + \max_C \left(\frac{\partial v}{\partial z}\right)^2}. \qquad (6)$$

The maximum of each of these partial derivatives is subsequently bound by

$$\begin{aligned} \max_C \left|\frac{\partial v}{\partial x}\right| &\leq \max_{j,k \in \{0,1\}} |V_{1jk} - V_{0jk}| \\ \max_C \left|\frac{\partial v}{\partial y}\right| &\leq \max_{i,k \in \{0,1\}} |V_{i1k} - V_{i0k}| \\ \max_C \left|\frac{\partial v}{\partial z}\right| &\leq \max_{i,j \in \{0,1\}} |V_{ij1} - V_{ij0}| \qquad (7) \end{aligned}$$

3.3 Lipschitz-Octree Construction

Large Lipschitz bounds result in smaller step sizes. Often most of an image is fairly regular, but a few areas of the volume have high gradients which cause large Lipschitz constants. Since a global Lipschitz bound must be guaranteed accurate everywhere in the volume, these few areas force a poor Lipschitz bound for the entire volume. Hence, we borrow the concept of local Lipschitz bounds from [6].

We construct a hierarchy of Lipschitz bounds $L_m : \mathbf{Z}^3 \to \mathbf{R}, (0 \le m \le M)$. Using equations (6) and (7), we compute the volume L_0 whose entries are the Lipschitz bounds of each cell of the original volume. The Lipschitz constant of any continuous collection of cells forming a convex shape is bound by the maximum of the Lipschitz constants of the individual cells[1]. Hence, we build the hierarchy from the bottom up, each level a cubic region consisting of eight smaller cubes, and whose Lipschitz bound is the maximum of the Lipschitz bounds of its eight components,

$$L_m(x,y,z) = \max_{i,j,k \in \{0,1\}} L_{m-1}(2x+i, 2y+j, 2z+k). \quad (8)$$

The top level of the hierarchy L_M contains one cell, and this cell's entry is a global Lipschitz bound.

Comparing (8) to (1), the essential difference between the octree and the Lispchitz-octree is that the latter is independent of the isosurface threshold.

Volumes are often padded with zeros to make them cubes with power-of-two edge sizes. If the voxels at the edge of the original volume are non-zero, then padding zeros will create artificially high Lipschitz constants in the hierarchy along this boundary. Instead, edges must be padded by continuing the values at the edge to the padded resolution to avoid Lipschitz artifacts. These extra voxels may be culled during ray intersection by clipping the rays to the boundaries of the original volume.

4 Ray Intersection

Ray intersection uses the Lipschitz condition to step along the ray over areas known not to intersect the iso-surface. A global Lipschitz bound penalizes the entire volume for the region with the largest gradient whereas hierarchical methods use more localized Lipschitz bounds, resulting in larger step sizes. The global algorithm is presented first.

[1]Proof: A straight line l connecting any two endpoints a, b in the convex shape is itself in the convex shape. Letting the rest follow by induction, consider the simple case of two cells where a is in one cell and b in the other. Let l_a be the segment of l from a to the point c on l where it intersects the boundary between the cells, and let $|l_a|$ denote its length. Define l_b likewise. Let L_a (L_b) be a constant satisfying the Lipschitz condition for the cell containing a (b), and let L be the maximum of the two. Then

$$\begin{array}{lll} |f(a)-f(c)| & \le L_a|l_a| & (*) \\ |f(b)-f(c)| & \le L_b|l_b| & (**) \\ |f(a)-f(b)| & \le |f(a)-f(c)| + |f(b)-f(c)| & (\text{by } \Delta \neq) \\ & \le L_a|l_a| + L_b|l_b| & (\text{by } (*),(**)) \\ & \le L|l|. & \square \end{array}$$

4.1 Global Ray Intersection

Let L be the Lipschitz bound of the entire continuously reconstructed volume $v : \mathbf{R}^3 \to \mathbf{R}$ and let T be the desired threshold. Given a scalar value $s = v(\mathbf{x})$ at a point $\mathbf{x}$, one knows with surety, as a consequence of (2), that one could move a distance of

$$d = \frac{s - T}{L} \quad (9)$$

in any direction before any scalar values could possibly reach the threshold T. Eq. (9) underestimates the distance from $\mathbf{x}$ to the iso-surface, which was shown to be useful for ray tracing originally in [5].

A ray is defined by an anchor $\mathbf{r}_0$ and a unit direction $\mathbf{r}_d$ parametrically as

$$\mathbf{r}(t) = \mathbf{r}_0 + t\mathbf{r}_d. \quad (10)$$

The sequence defined by the recurrence equation

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \frac{v(\mathbf{x}_{n-1}) - T}{L}\mathbf{r}_d \quad (11)$$

begining with $\mathbf{x}_0 = \mathbf{r}_0$, converges to the first intersection between the ray and the iso-surface if and only if it converges at all. Thus, for each point $\mathbf{x}_{n-1}$ along the ray, the next point $\mathbf{x}_n$ can be determined by calculating the scalar value at $\mathbf{x}_{n-1}$ and determining the safe stepping distance d based on equation (9). This process is repeated until the value at $\mathbf{x}_n$ is sufficiently close to the threshold, or until the sequence escapes the volume.

4.2 Hierarchical Ray Intersection

Given a point $\mathbf{x}$ on the ray, one can use the Lipschitz bound of the entire volume with (9) to compute d_M, the size of the next step along the ray. But if the global Lipschitz bound is too large, then perhaps one could do better at a more local level.

Let $L_m(\mathbf{x})$ be the local Lipschitz bound, valid only in the domain $D_m(\mathbf{x})$ that contains the current point, of the 8^{M-m} domains at level m of the octree. Let d_m indicate the Lipschitz bound on the distance at level m, from $\mathbf{x}$ to the isosurface, valid only within the domain $D_m(\mathbf{x})$.

Let l_m be the length (distance) along the ray in the positive direction from the current point to the boundary of the domain $D_m(\mathbf{x})$.

When the local Lipschitz distance bound meets or exceeds the distance to its domain of validity ($d_m \ge l_m$), then one can safely step forward along the ray only to the current domain's boundary. Figure 1 illustrates an example.

In the other case ($d_m < l_m$), perhaps a more localized Lipschitz bound will give a greater distance.

If the distance d_m steps beyond the domain $D_{m-1}(\mathbf{x})$, or symbolically $d_m \ge l_{m-1}$, then we need not find the local Lipschitz bound $L_{m-1}(\mathbf{x})$, since it is guaranteed

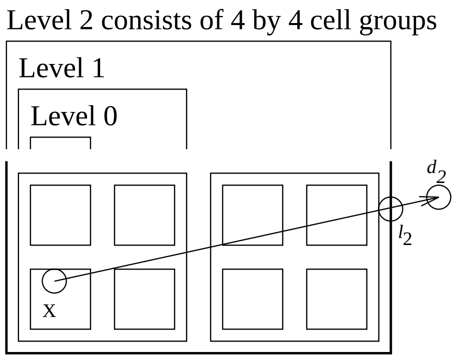

Figure 1: Case A: Local Lipschitz distance bound d_2 exceeds ray-boundary distance l_2. Step forward only to the level 2 boundary.

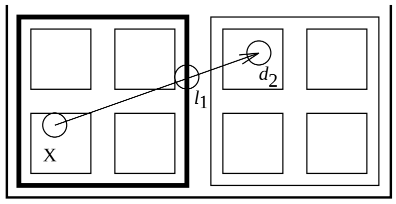

Figure 2: Case B: No need to try level 1 since l_1 is less than d_2. Step forward d_2.

only within $D_{m-1}(\mathbf{x})$. At this point, one cannot do better than d_m. Figure 2 illustrates an example.

On the other hand, if $d_m < l_{m-1}$, and stepping by d_m remains within the domain $D_{m-1}(\mathbf{x})$, then compute d_{m-1}. Figure 3 illustrates an example.

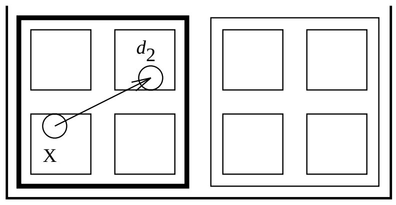

Figure 3: Case C: Since d_2 does not exceed l_1, try level 1 for a better Lipschitz distance bound.

This process repeats, subdividing the octree as necessary, searching for the largest possible safe step size. The subdivision terminates at the bottom of the octree $m = 0$. Figure 4 illustrates an example.

After stepping along the ray, determination of the next step size starts at the same level m used to determine the previous step, or one step higher $m+1$ if a level m domain boundary was crossed. Figure 5 summarizes the algorithm.

For ray tracing general implicit surfaces, the convergence test should be set to machine precision to avoid numerical problems in self-shadowing. However, since

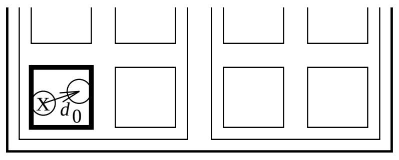

Figure 4: Case D: Step forward d_0, the distance provided by the cell-level Lipschitz bound.

we are dealing with linearly interpolated discrete volume data, and self-shadowing is not routinely performed in typical volume visualization environments, better convergence test are available.

The algorithm need only step along the ray until the distance puts the surface within the current cell in L_0. Then one can solve, in closed form for trilinear interpolation, for any intersections of the ray with the interpolated iso-surface. If there are no intersections, then one moves along the ray to the boundary of the current cell and continues stepping from there.

Another alternative is to sample a cell known to contain the iso-surface at its centroid. When volume resolution is much greater than image resolution, the artifacts of this approximation are hardly visible.

5 Results

Section 5.1 describes the implementation and the examples used for analysis of the algorithm. Section 5.2 analyzes the algorithm's performance, showing it spends most of its time on silhouette edges. Section 5.3 compares the algorithm to other acceleration techniques, arguing, solely on the basis of algorithm analysis, that the Lipschitz-octree combines the benefits of the min-max octree and the distance transform to beat both algorithms individually.

5.1 Implementation

We have implemented the Lipschitz-octree structure and analyzed its performance, but, at the time of this writing, we have not yet implemented the other acceleration techniques. We hope to have implemented the other ray acceleration techniques and to have found conclusive timing results at the time of this paper's presentation.

In the interest of fair comparison, the results presented here only count the steps needed to converge to the cell that contains the iso-surface. Our implementation continues the process to converge onto the intersection of the ray with the isosurface, though there are faster, constant-time techniques available for this.

The Lipschitz-octree algorithm was tested on a digitized lobster (Figure 6) and on four voxelized spheres implicitly defined by the integer function

$$V(x, y, z) = x^2 + y^2 + z^2. \qquad (12)$$

```
Initialize $\mathbf{x}$ to the point where ray $\mathbf{r}(t) = \mathbf{r}_0 + t\mathbf{r}_d$ enters the volume.
Initialize $m = M$.                                          The maximum (global) level.
Until $\mathbf{x}$ hits the iso-surface or exits the volume ...
    Initialize $d = 0$.                                      Current safe stepping distance.
    While $m >= 0$ ...                                       the minimum (cell) level (Case D)    .
        Let $l_m$ = length along the ray from $\mathbf{x}$ to the boundary of $D_m(\mathbf{x})$.
        If $d > l_m$ then break (while).                     Distance at maximum (Case B).
        Let $d_m = |v(\mathbf{x}) - T|/L_m(\mathbf{x})$.     Distance guaranteed not to intersect iso-surface.
        Let $d = \min(d_m, l_m)$.                            $d_m$ only valid up to $l_m$.
        If $d_m > l_m$ then break (while).                   Distance at maximum (Case A).
        Decrement $m$.                                       Try for better results one level finer (Case C).
    Let $\mathbf{x} = \mathbf{x} + d\mathbf{r}_d$.           Step along the ray by $d$.
    Let $m = \max(m + 1, M)$.                                Start one level higher next time.
Return intersection at $\mathbf{x}$.
```

Figure 5: The hierarchical Lipschitz-based ray traversal algorithm. Figures 1 through 4 illustrate cases A-D.

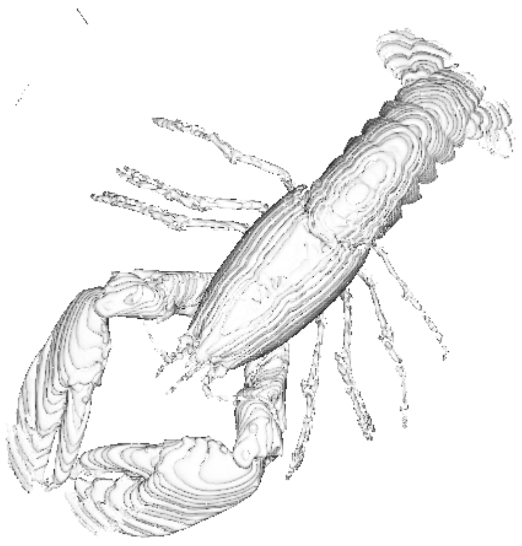

Figure 6: Rendered image of the lobster

5.2 Analysis

Table 1 lists the average distance between samples for both hit and miss rays.

Even for highly detailed volumes such as the lobster, the average step size between samples is well over cell size. For more uniform volumes such as the spheres, the average step size between samples can reach an order of magnitude higher than any uniform voxel stepping algorithm could achieve. The results of the spheres' rendering demonstrate that as the resolution of the volume increases, the Lipschitz algorithm's results improve.

Figure 7 portrays a histogram of the step sizes between samples for the images of Table 1. Because of the clipping to octree-cell boundaries, the step sizes tend to be at powers of two.

Volume	Cells	Ave. Step Size	
		Hit Rays	Miss Rays
Lobster	$319^2 \times 33$	1.52	2.36
Sphere 1	32^3	2.95	2.38
Sphere 2	64^3	4.10	4.40
Sphere 3	128^3	5.20	7.33
Sphere 4	200^3	6.36	10.22

Table 1: Step size statistics for rays that hit the iso-surface and rays that miss the iso-surface.

Figure 8 illustrates that silhouette edges require the most computation (compare to [8]). The lobster was suspended in a cylinder, perhaps a bucket, while being digitized, which caused a ring of voxels with large gradients, resulting in poor Lipschitz bounds and an increased number of samples required per ray. The ring is isolated by the octree structure, so its impact on the performance is decreased somewhat, though clipping such scanning artifacts would yield lower Lipschitz constants, improving performance.

5.3 Discussion

The step sizes in Table 1 show the average Lipschitz-octree algorithm step sizes beat the unit step sizes of ray marching. Moreover, the Lipschitz-octree structure combines the benefits of both the min-max octree and the distance transform, with the expectation that it can outperform both methods individually.

The min-max octree [13] is comparable to the Lipschitz octree. Given a point $\mathbf{x}$ in a level m octree cell, the

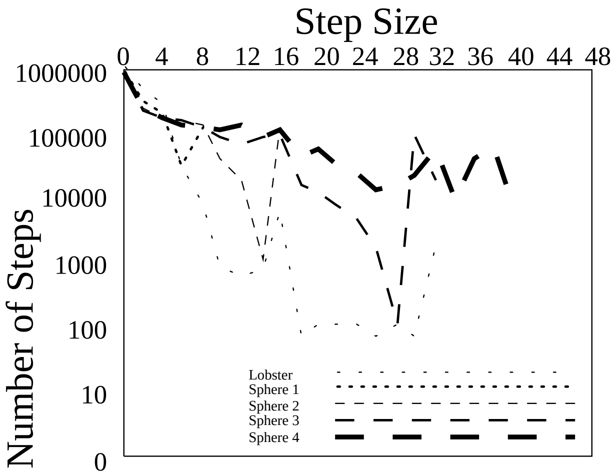

Figure 7: Step size histogram. The horizontal axis indicates various distances between steps. For each step size, the vertical axis indicates how many actual ray tracing steps were that size.

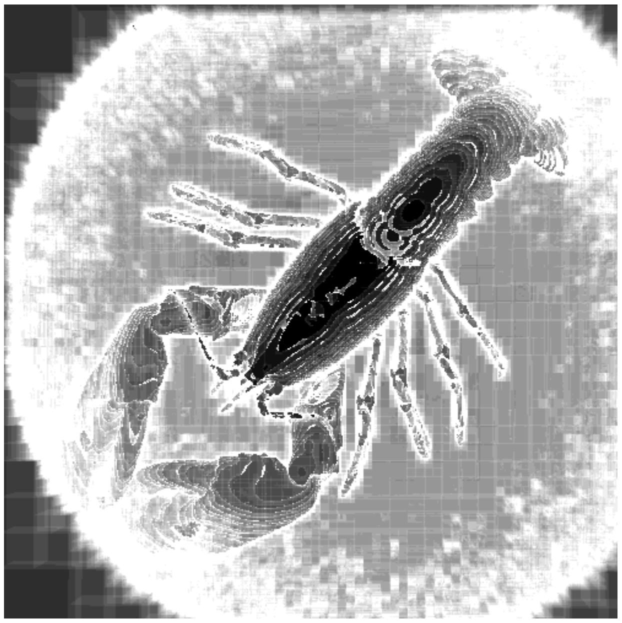

Figure 8: Work image where brightness indicates number of steps taken to traverse the ray, showing the algorithm concentrates on silhouette edges.

values in that cell are bounded by $B^+_m(\mathbf{x})$ and $B^-_m(\mathbf{x})$ in the min-max octree, where B^+_m and B^-_m functions return the minimum and maximum of the level m cell containing $\mathbf{x}$. In the Lipschitz octree, the level m cell values are bounded by $v(\mathbf{x}) + rL_m(\mathbf{x})$ and $v(\mathbf{x}) - rL_m(\mathbf{x})$ where r is the distance from $\mathbf{x}$ to the farthest point on the boundary of the level m cell. The comparability of these two structures shows the Lipschitz-octree method to operate similarly to the min-max octree method, through with the added benefit of the Lipschitz distance bound.

The step size histograms in Figure 7 are roughly shaped as the histograms in [16], implying that the step size distribution of both algorithms is skewed toward the smaller step sizes. This similarity implies that the Lipschitz bound steps perform comparably to the distance transform steps. The use of an octree can be seen in Figure 7 as variances in the general trend of the histogram. Although the Lipschitz distance bound and its associated octree generate more conservative distance estimates than distance coding, both algorithms converge similarly and Lipschitz octrees furthermore avoid the cost of reprocessing for each new isovalue.

6 Conclusion

The Lipschitz-octree structure, based on techniques from implicit surface research, merges the benefits of both the min-max octree and the distance transform to create a new faster direct volume isosurface renderer. Unlike any other direct volume ray-casting acceleration technique except min-max octrees, the same Lipschitz octree can be used to render isosurfaces at any desired threshold level.

6.1 Further Research

Since most volumes are not square nor powers of two (note the dimensions of the lobster) and since the interpolation function defines the volume over $\mathbf{R}^3$, it may be more efficient to build the Lipschitz hierarchy independent of the cell resolution of the volume, from the top down to some bottom level just below the cell resolution. This would eliminate any need for padding. In fact, this would also eliminate the requirement of a cubic lattice, extending the algorithm to applications involving the visualization of volumetric data defined on irregular grids.

The Lipschitz-octree method was developed for final presentation-quality rendering. Fast approximate volume rendering algorithms, such as those surveyed in [2], produce the best rendering in a given amount of time or within a given error bound, and are useful for interactive volume visualization. Mip-mapping average opacity and shading information into the octree, and using the Lipschitz bounds as an error bound, transforms this method into a more interactive approximate renderer.

We are pleased with the interplay of volume isosurfaces and implicit surfaces, and have integrated volume visualization into our research program in implicit surfaces with the expectation that their similarities will foster further advancement in both areas.

6.2 Acknowledgements

This research supported in part by the National Science Foundation under Research Initiation Award #CCR-9309210. The research was performed in the Imaging Research Laboratory, supported in part by the School of EECS and the NSF under grant #CDA-9121675.

References

[1] Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing 34* (1986), 334–371.

[2] Danskin, J., and Hanrahan, P. Fast algorithms for volume ray tracing. In Proc. of *1992 Workshop on Volume Visualization* (Oct. 1992), pp. 91–98.

[3] Drebin, R. A., Carpenter, L., and Hanrahan, P. Volume rendering. *Computer Graphics 22*, 4 (Aug. 1988), 65–74.

[4] Hart, J. C. Sphere tracing: Simple robust antialiased rendering of distance-based implicit surfaces. Tech. Rep. EECS-93-15, School of EECS, Washington State University, Jan. 1993. Appears in SIGGRAPH '93 Course Notes #25 "Modeling, Visualizing and Animating Implicit Surfaces".

[5] Hart, J. C., Sandin, D. J., and Kauffman, L. H. Ray tracing deterministic 3-D fractals. *Computer Graphics 23*, 3 (1989), 289–296.

[6] Kalra, D., and Barr, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics 23*, 3 (July 1989), 297–306.

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

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

[9] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics 21*, 4 (July 1987), 163–170.

[10] Verwer, B. J., Verbeek, P. W., and Dekker, S. T. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 4 (April 1989), 425–429.

[11] Von Herzen, B., and Barr, A. H. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics 21*, 4 (July 1987), 103–110.

[12] Von Herzen, B., Barr, A. H., and Zatz, H. R. Geometric collisions for time-dependent parameteric surfaces. *Computer Graphics 24*, 4 (Aug. 1990), 39–48.

[13] Wilhelms, J., and Van Gelder, A. Octrees for faster isosurface generation (extended abstract). *Computer Graphics 24*, 5 (Nov. 1990), 57–62.

[14] Wyvill, G., McPheeters, C., and Wyvill, B. Data structure for soft objects. *Visual Computer 2*, 4 (1986), 227–234.

[15] Yagel, R., and Shi, Z. Accelerating volume animation by space-leaping. In Proc. of *Visualization '93* (1993), G. M. Nielson and D. Bergeron, Eds., IEEE Computer Society Press, pp. 63–69.

[16] Zuiderveld, K. J., Koning, A. H. J., and Viergever, M. A. Acceleration of ray-casting using 3-D distance transforms. In Proc. of *Visualization in Biomedical Computing 1992* (Oct. 1992), vol. 1808, pp. 324–335.