Iso-splatting:
A Point-based Alternative to Isosurface Visualization

Christopher S. Co Bernd Hamann Kenneth I. Joy
Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science
University of California, Davis
{co,hamann,joy}@cs.ucdavis.edu

Abstract

We present a new approach to isosurface visualization that we call "iso-splatting." We use point primitives for representing and rendering isosurfaces. The method consists of two steps. In the first step, point samples are generated throughout the volumetric domain of a scalar function. In the second step, these points are projected onto the isosurface of interest. We render the resulting point set using a surface splatting algorithm. The method can be extended to out-of-core or parallel environments. Our results show that this method can offer much greater time and space efficiency when compared with standard triangle-based methods, thereby supporting higher levels of interactivity. Parts of the algorithm can be accelerated using graphics hardware. One key advantage of this approach is that, since extraction computations are divided into two smaller phases, work can be distributed to exploit all available resources.

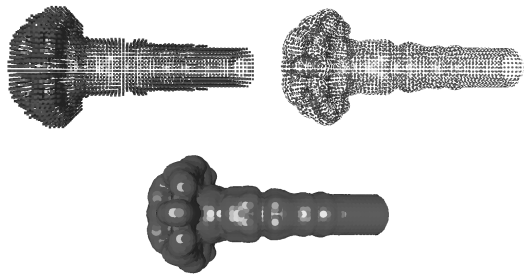


Figure 1. Iso-splatting applied to a fuel injection simulation data set. Top-left: point samples generated near the isosurface; top-right: points projected onto the isosurface; bottom: resulting point set rendered using surface splatting.

1 Introduction

Isosurface visualization is an extremely valuable method for the exploration of scalar fields. A large number of publications propose various methods for optimizing the extraction and rendering phases of isosurface visualization. Isosurfaces are typically stored and rendered as polygonal meshes, usually triangle meshes. Recent developments in the rendering of densely sampled models using point-based primitives have made point-based rendering of surfaces a viable and popular alternative [24, 25, 27, 1, 10, 22]. We propose an approach for more efficient isosurface visualization based on recent advances in point-based rendering.

A large fraction of isosurface visualization research has followed one model: a user specifies an isovalue of interest, portions of the domain intersected by the isosurface are determined, and geometric primitives are computed. These geometric primitives are then rendered. The process is performed for each new isovalue specified by the user. For each isovalue, previous isosurface geometry is typically discarded, and a completely different set of geometry is computed.

This basic methodology is quite straightforward, but it is also wasteful. A relatively large amount of computation must be performed to produce geometry specific to each isovalue. One set of geometric primitives created by this method can only be used to represent one specific isosurface. The situation is made worse when considering the cost involved in storing polygonal meshes.

Iso-splatting provides an alternative to this paradigm by splitting the work of extraction into two independent steps. In the first step, discussed in Section 4, point samples are generated. In the second step, these point samples are projected onto the isosurface of interest, which we discuss in Section 3. The resulting point set is rendered using a surface

splatting technique. Figure 1 illustrates the iso-splatting technique applied to a fuel injection data set. The crucial difference between our approach and "standard" polygon-based methods is that the point geometry can be used to represent a range of isosurfaces. Geometry must be adjusted only to a relatively small degree to represent a slightly different isosurface. By virtue of this fact, we greatly decrease computation time. The use of points, in many cases, can also decrease storage requirements. Furthermore, this new isosurface visualization paradigm can be tailored to meet application-specific needs, which we discuss in Section 5.

2 Related Work

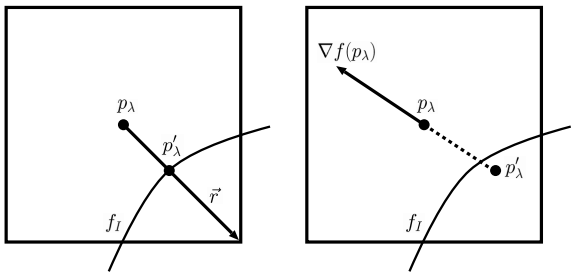
Isosurfacing algorithms can be classified as either view-dependent or view-independent. View-dependent approaches focus on the resulting image and therefore attempt to perform computation mainly in regions that contribute substantially to the final image. At approximately the same time when Lorenson and Cline published their well-known Marching Cubes (MC) paper [15], they developed a lesser known method called Dividing Cubes (DC) [5]. Based on viewing parameters, the DC algorithm renders grid cells as points after iteratively or recursively subdividing cells to a pixel or subpixel level in screen space. Later, ray tracing was employed to render isosurfaces of large data sets interactively [17]. This method is attractive since it is relatively insensitive to input data size and thus scales well. View-dependent approaches are attractive in general as no intermediate form of the isosurface needs to be stored explicitly, which greatly decreases storage requirements. One drawback of view-dependent approaches is that each time a new view is specified the isosurface extraction process must be repeated. For interactive applications, where viewing parameters are being changed frequently, such methods perform a relatively large number of computations. View-dependent approaches often offer excellent image quality, but frequently no geometric representation of the isosurface is generated, making them undesirable for use in geometric modeling applications, for example.

View-independent approaches in general generate geometry near the isosurface. Most methods are based on the MC method and use triangles to approximate an isosurface [8, 14, 4, 2, 3, 11, 9]. The use of *interval trees* [4] and the *span space* [14] domain decomposition can greatly decrease the amount of work necessary to identify cells intersected by an isosurface (also called *active-cells*), a major bottleneck in the extraction process. One advantage of generating geometry is that extraction need not be performed for each view. However, storing geometry becomes a burden as data resolution increases. *Dual contouring* methods were introduced to preserve sharp features and to alleviate storage requirements by reducing triangle count [11, 9]. Such methods produce high-quality polygonal models, but are not ideal for interactive visualization of large data due to the added computation and intermediate data storage requirements.

Researchers have investigated the use of point primitives as a rendering and representation alternative, and this topic has received much attention in recent years. The use of points as a rendering primitive dates back to at least 1985, when Levoy and Whitted [12] described its advantages and limitations. Grossman and Dally [7] revisited the notion of using points more than ten years later. The use of points to represent a surface was also promoted by Rusinkiewicz and Levoy [24, 25], who proposed a hierarchical surface splatting technique for rendering surfaces of great complexity. Pfister et al. [19] proposed the use of *surfels* for the representation and rendering of surfaces. The surface splatting algorithm was later formalized and improved using *elliptical weighted averaging* (EWA) to support texturing, hidden surface removal, edge anti-aliasing, and transparency [27]. Ren et al. [22] developed a hardware-accelerated approach based on an object space formulation of the EWA surface splatting algorithm. (We refer the reader to [21] for an overview of splatting theory and implementation issues.) Besides surface splatting, many other techniques were developed using points to render surfaces. Kalaiah and Varshney [10] developed an approach to rendering a surface through the use of *differential points*. Alexa et al. [1] used a point set and *moving least-squares* techniques to define a surface that can be down-sampled and up-sampled as needed to meet rendering and modeling requirements. Fleishman et al. [6] extended this point-set-surface approach using multiresolution methods.

These point-based rendering techniques deal primarily with static surface models. Isosurfaces can exhibit vast geometric and topological changes when changing the isovalue of interest. Due to this "dynamic" nature, the idea of using points to represent isosurfaces seems daunting. However, points provide a large degree of flexibility and therefore are often more suitable than triangles when dealing with such changing surfaces.

The method we present focuses on improving interactivity through storage and computational efficiency via the use of point samples. Iso-splatting is a view-independent approach, as it generates the isosurface geometry once per isovalue and does not require repeated extraction when viewing parameters change. We show how large storage and computational savings can be achieved through the use of points as a basis for surface representation and rendering. We employ surface splatting techniques, although other point-based rendering techniques can be used without altering core aspects of our algorithm.



(a) (b)

Figure 2. Projection of sample point p_λ onto isosurface f_I, producing point p'_λ using (a) exact projection and (b) approximate projection.

3 Projection

Given the trilinear function

$$f(x,y,z) \quad = \quad \sum_{i,j,k=0}^{1} \rho_{ijk} x^i y^j z^k \ ,$$

where ρ_{ijk} denote the eight polynomial coefficients associated with one trilinearly interpolated cell, we are interested in the isosurface defined by the isovalue

$$f_I \quad = \quad \sum_{i,j,k=0}^{1} \rho_{ijk} x^i y^j z^k. \qquad (1)$$

We project a sample point p_λ, which is inside an active-cell, to obtain p'_λ, a point on or near the isosurface.

3.1 Exact Projection

Given a ray

$$\vec{r}(t) = p + t\vec{d},$$

where p is a point on the ray and $\vec{d}$ is an associated direction vector, the exact intersection between $\vec{r}$ and the isosurface defined by Equation (1) can be determined by solving

$$\begin{aligned} f_I \quad &= \quad f(p + t\vec{d}) \\ &= \sum_{i,j,k=0}^{1} \rho_{ijk} (p_x + t d_x)^i (p_y + t d_y)^j (p_z + t d_z)^k \end{aligned} \qquad (2)$$

for the unknown ray parameter t. Once t is known, the sample on the exact isosurface is $p'_\lambda = p + t\vec{d}$. For an arbitrary ray direction, Equation (2) leads to a cubic equation in t of the general form

$$At^3 + Bt^2 + Ct + D \quad = \quad 0 \, . \qquad (3)$$

Cardan's solution [16] can be used to determine the roots of Equation (3). (For details on implementing ray-isosurface intersections, we refer the reader to [17].)

The key to computing this intersection is defining the ray $\vec{r}$. From the sample point p_λ, rays can be shot to a subset of the eight corners of the cell, using the MC case table to determine which corners should be considered. One can also use the gradient of the scalar function computed at p_λ as a direction vector for $\vec{r}$. However, a ray defined in such a way may not intersect the isosurface inside the cell. Figure 2 (a) illustrates exact projection.

This approach produces points that lie on the isosurface but at a high computational cost. Roots of a cubic polynomial must be determined in order to obtain ray intersections, thereby slowing down the projection phase of this method. An approximation can be performed to alleviate this computational burden.

3.2 Approximate Projection

We use one iteration of the Newton-Raphson [20] root-finding method to compute an approximation of p'_λ. First, we describe how this formula is used to find an approximate isopoint of a univariate scalar function and then describe an extension to find a point approximately near the isosurface of a trivariate scalar function.

The Newton-Raphson method essentially uses the first two terms of the Taylor series expansion of a function $h(x)$ near a root. Given a univariate function $f(x)$ and an isovalue f_I, we seek the roots of

$$h(x) = f(x) - f_I \, .$$

The Taylor series of $h(x)$ at a point $x_0 + \epsilon$ is defined as

$$h(x_0 + \epsilon) = h(x_0) + h'(x_0)\epsilon + \frac{h''(x_0)}{2}\epsilon^2 + \ldots \, .$$

The first iteration of the Newton-Raphson method considers only the first-order terms and solves

$$h(x_0 + \epsilon) = h(x_0) + h'(x_0)\epsilon = 0 \, . \qquad (4)$$

If we consider $x_0 + \epsilon$ to be a line parameterized by t, where $\epsilon = h'(x_0)t$, we obtain

$$h\big(x_0 + h'(x_0)t\big) = h(x_0) + h'(x_0)h'(x_0)t = 0 \, ,$$

which, when we solve for t, results in

$$t = \frac{-h(x_0)}{h'(x_0)h'(x_0)} \, .$$

Since $h(x_0) = f(x_0) - f_I$ and $h'(x) = f'(x)$ we obtain

$$t = \frac{f_I - f(x_0)}{f'(x_0)f'(x_0)} \, .$$

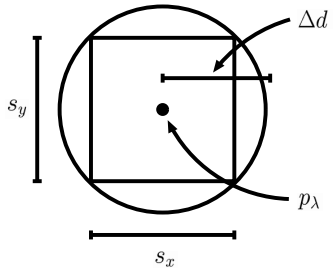


Figure 3. Displacement criterion. The square symbolizes the cell, the black dot the sample location p_λ. The sample point p_λ may be projected onto an arbitrary isosurface that intersects the cell as long as the projection does not exceed the displacement boundary, defined by the sphere of radius Δd, which is determined by cell dimensions s_x and s_y.

We now consider Equation (4) for a trivariate function $h(\mathbf{x})$ at a point $\mathbf{x}_0 + \vec{\epsilon}$, i.e.,

$$h(\mathbf{x}_0 + \vec{\epsilon}) = h(\mathbf{x}_0) + \nabla h(\mathbf{x}_0) \cdot \vec{\epsilon} = 0 \, .$$

If we consider $\mathbf{x}_0 + \vec{\epsilon}$ to be a line parameterized by t, where $\vec{\epsilon} = \nabla h(\mathbf{x}_0)t$, we obtain

$$h\big(\mathbf{x}_0 + \nabla h(\mathbf{x}_0)t\big) = h(\mathbf{x}_0) + \nabla h(\mathbf{x}_0) \cdot \nabla h(\mathbf{x}_0)t = 0 \, ,$$

which leads to

$$t = \frac{-h(\mathbf{x}_0)}{\nabla h(\mathbf{x}_0) \cdot \nabla h(\mathbf{x}_0)} \, .$$

By substituting $h(\mathbf{x}_0) = f(\mathbf{x}_0) - f_I$ and $\nabla h(\mathbf{x}_0) = \nabla f(\mathbf{x}_0)$, one obtains

$$t = \frac{f_I - f(\mathbf{x}_0)}{\nabla f(\mathbf{x}_0) \cdot \nabla f(\mathbf{x}_0)} \, . \qquad (5)$$

By setting $\mathbf{x}_0 = p_\lambda$, we can evaluate $f(p_\lambda)$ and $\nabla f(p_\lambda)$ efficiently at the same time using trilinear interpolation. By computing t according to Equation (5), the approximation of p'_λ is given as

$$p'_\lambda = p_\lambda + \nabla f(p_\lambda)t \, . \qquad (6)$$

Figure 2 (b) illustrates this approximate projection.

The developers of *Kizamu* [18] used a similar approach to project arbitrary point samples onto the zero-set of an adaptive distance field. Their technique was meant as a method to preview a surface defined by a volumetric distance field, while iso-splatting is a method designed to visualize isosurfaces of arbitrary scalar field data.

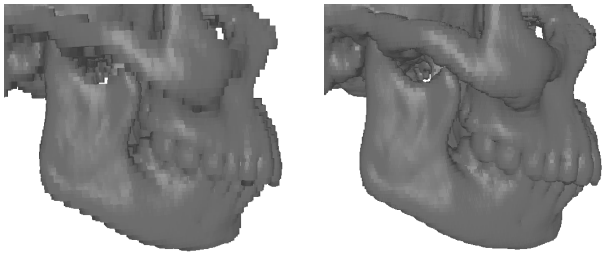


Figure 4. Impact of point sample projection. Shown is an isosurface of a skull rendered with quadrilateral surface splats. (a) Samples are grid-aligned when unprojected. (b) Same iso-splatted surface using projection.

The Newton-Raphson method converges quadratically, provided that the initial guess $\mathbf{x}_0$ is sufficiently close to a root. This method may result in roots far away from the "desired" root when $\mathbf{x}_0$ is near a local maximum or minimum. For this reason, we discard "far-away" solutions when encountered using a displacement criterion. After a single iteration of the Newton-Raphson procedure, if the point is "too far" from the initial guess, we do not consider the point. A displacement threshold Δd is computed based on the dimensions of the cell. Given cell dimensions s_x, s_y, and s_z, we define the maximum displacement threshold as $\Delta d = \frac{1}{2}\sqrt{{s_x}^2 + {s_y}^2 + {s_z}^2}$. Geometrically speaking, when the sample point p_λ is the center of the cell, this displacement criterion is the same as restricting the location of p'_λ to reside inside the sphere of radius Δd with center p_λ, see Figure 3. Figure 4 illustrates the difference between unprojected and projected point samples.

4 Sampling

Often, the contribution to the isosurface of a given cell can be approximated cheaply without sacrificing overall rendering quality. In a standard triangle-based isosurfacing method, grid cells that are intersected by an isosurface are determined. For these cells, edge intersection points are computed, triangulated and added to a polygonal model. In iso-splatting, instead of generating a set of triangles inside a cell, we generate a point sample. Although a given cell may contain several disjoint components of the same isosurface, we have found that many high-resolution data sets only rarely exhibit this phenomenon. In fact, restricting the amount of geometry to one point per cell is a common technique used in large model simplification [23, 13].

A surface rendering algorithm requires certain information necessary to synthesize an image. For surface splatting,

a fairly regular sampling of the surface is required in order to produce a proper rendering. Each sample should be characterized by location, normal, and a local variance matrix, which indicates roughly the average distance to neighboring samples. Uniform rectilinear volume data implicitly provides locations of volume samples. For this type of data, gradients are often used for computing local illumination properties. The variance matrix can be derived by knowing the grid spacing. We can therefore derive all the elements required to define a surface splatting primitive.

Point samples may not lie on the desired isosurface and must be projected onto that surface. If an exact projection (as discussed in Section 3.1) is used, the cell corner values are necessary to compute the projection of this sample point onto the isosurface. If our approximation (as discussed in Section 3.2) is used, a function value and gradient must be computed for this sample point. The gradient can be reused for shading purposes in the rendering phase.

The original data may be sampled more sparsely in one dimension than in the other dimensions. In this case, it may be beneficial to generate more samples in the sparse dimension via cell subdivision, similar to the approach chosen in DC [5]. One possible criterion for cell subdivision is function value interval width. Cells that span a large range of values can be subdivided until the interval width of child cells are below a threshold. Another solution is to alter the variance matrix such that the rendering algorithm produces splats with an appropriate shape and size to compensate for uneven sample spacing.

In general, iso-splatting requires that each point sample consist of sample location, function value, gradient, and a local variance matrix. For gridded data sets, several methods exist to obtain location, function value, and gradient. The local variance matrix can be derived by analyzing the spatial distribution of the volume samples or from the size and shape of grid cells. Thus, iso-splatting is applicable, in principle, to any type of grid.

5 Implementation Issues

The use of the point as a representation and rendering primitive for isosurfaces results in many desirable features. One primary advantage is its flexibility at the application level. Computations necessary for iso-splatting can be performed in different stages of the visualization pipeline, providing more flexibility than many triangle-based approaches.

There are three stages of any geometry-based isosurfacing pipeline:

1. **Preprocessing**
   Data reorganization or data structure initialization is performed to improve the extraction and/or rendering stages.
2. **Extraction**
   Geometry approximating an isosurface is computed.
3. **Rendering**
   The isosurface geometry is rendered.

Two phases characterize the iso-splatting algorithm:

1. **Point generation**
   Sample points are selected, and key information for the projection and rendering phases are computed.
2. **Point projection**
   Sample points are projected onto the isosurface.

Stage	(A)	(B)	(C)	(D)
Preprocessing	G	G		
Extraction	P		G,P	G
Rendering		P		P

G = point generation P = point projection

Table 1. Iso-splatting paradigms.

In iso-splatting, point sample generation and projection can be performed in different stages of the isosurface visualization pipeline to satisfy application-specific needs. We recognize four basic application paradigms, shown in the columns of Table 1.

Paradigms (A) and (B) are useful when a high level of interactivity is desired and storage is not a major issue. These two paradigms can be storage-intensive, since point samples are pre-generated and must be stored for later use. Using the approximate projection scheme, location, function value, and gradient must be stored for each point sample. In addition, in order to perform extraction, the function value interval of the cell in which a point sample resides must be known. Under such conditions, out-of-core techniques should be more readily considered. When the input data size is already too large to fit in-core, this approach may not be a problem. Under paradigms (A) and (B), extraction consists primarily of collecting point samples in active-cells. Several isosurface extraction techniques exist to perform active-cell lookups efficiently out-of-core [8, 2, 3]. Furthermore, since point samples are not highly coupled (i.e., point samples do not require information about other point samples), parallel extraction of the point samples from disk is possible. Performing point sample generation in the preprocessing stage can be thought of as a “partial evaluation” of the isosurface.

Paradigms (C) and (D) require more computation but provide better storage efficiency than paradigms (A) and (B). These two paradigms require only the original data set, from which we can compute the information needed for projection and rendering. In many cases, iso-splatting offers

improved extraction efficiency over standard triangle-based schemes. A standard MC implementation performs several edge intersections and perhaps computes a gradient for each point for smooth shading. In the method of Ju et al. [9], a quadratic error function must be minimized in addition to computing these edge intersections in order to place a single point inside the cell. In iso-splatting, a single function value and gradient computation, which can be computed simultaneously, is followed by one iteration of the Newton-Raphson procedure.

Programmable hardware on modern graphics cards makes the efficient computation of point projections feasible in the rendering phase, as in paradigms (B) and (D). The approximate projection procedure consists of a single floating-point division and a vector dot product, both operations supported by existing graphics hardware. Point location and gradient will already be sent down to the hardware for rendering the surface splat. The only additional piece of information that must be sent is the function value approximated for the point and the isovalue, which can be sent once per frame. The displacement criterion can be implemented with the use of texture lookups. However, the next generation of graphics processing units (GPUs) will also support branches (i.e., conditional statements), thereby allowing a more straightforward implementation of the displacement criterion. Discarded points can be rendered outside of the viewing frustum.

We believe that iso-splatting leads to computational efficiency, while using a geometric primitive that is highly storage-efficient. Consider n cells that contribute to an isosurface. Let us assume that each vertex has an associated normal vector used for shading purposes. Suppose a triangle-based isosurfacing algorithm generates one triangle per cell, and, moreover, outputs the entire mesh as one triangle strip, offering the most compact storage representation. Such a surface requires $n + 2$ vertices of storage. Iso-splatting generates one point per cell, producing at most n vertices. While the benefit is small, one must consider that many current triangle-based isosurfacing methods generate "triangle-soup" representations efficiently, which requires $3n$ vertices assuming one triangle per cell. (Standard MC implementations generate on average more than one triangle per cell.) Producing more memory-efficient triangle representations–such as a triangle strip in the ideal case–would most likely require more computation and thus slow down extraction time.

6 Results

We have performed tests on a Pentium4 PC with 2 GB of main memory and an nVidia GeForce4 Ti video card with 128 MB of memory on a motherboard supporting a $4\times$ accelerated graphics port (AGP). We implemented paradigms (A) and (C), see Table 1. QSplat-style surface splatting [24, 25] was chosen due to its rendering efficiency. However, EWA surface splatting [27, 22] could easily be incorporated into the system without affecting our algorithm.

We have used quadrilateral and elliptical surface splatting kernels, which offer different degrees of quality and efficiency. Bandwidth to the graphics card is currently a bottleneck in geometry-intensive applications. Quadrilateral splats can be more efficient, since they can often be implemented by passing a single vertex to the graphics hardware, which usually rasterizes this vertex as a quadrilateral. Hardware extensions exist to automatically scale the size of this rasterized quadrilateral based on viewing parameters. Elliptical splats offer better image quality, but, since they are often rendered using alpha-textured polygons, more geometry must be sent to the graphics hardware, thereby reducing rendering efficiency. When better support for surface splatting in graphics hardware exists, this need to balance the trade-off between quality and efficiency may disappear.

To quantitatively compare a triangle-based approach with iso-splatting, we extracted 16 isosurfaces 10 times for four data sets, measuring average extraction time and memory usage for a standard MC implementation and a paradigm-(C) iso-splatting implementation. In addition, we measured average framerate considering 36 frames for each isosurface, each time rotating the model by 10 degrees about the y-axis. For both implementations, interval trees were used to accelerate active-cell lookup. We chose a standard MC implementation over a dual-contouring implementation, as it offers the most competitive extraction times. Figure 6 summarizes extraction time, memory usage, and framerate information. For our framerate calculations, we used quadrilateral kernels due to their rendering efficiency. In the graphs, two curves of the same color represent measurements collected for the same data set. Of these two curves, the dotted curve corresponds to measurements made for the MC implementation, and the solid curve corresponds to measurements made for the iso-splatting implementation. The left-most graph in Figure 6 indicates that iso-splatting generally performs extraction in approximately half the amount of time used by a standard MC implementation, even when both point generation and projection are performed in the extraction stage. The fact that fewer primitives are generated and that these primitives are more storage-efficient is shown in the middle graph. This storage efficiency accounts for framerates that can exceed those achieved by rendering an isosurface represented by triangles, as seen in the right-most graph of Figure 6. This graph indicates that iso-splatting generally provides quadruple the framerates using quadrilateral splats when compared with a standard MC implementation. Figure 5 shows images of isosurfaces for each data set used in the experiment.

Figure 7 provides side-by-side comparisons of an im-

Data set	Dimensions
Fuel injection	$64 \times 64 \times 64$
Skull	$68 \times 256 \times 256$
Bucky ball	$128 \times 128 \times 128$
Aneurysm	$256 \times 256 \times 256$
Primate lung	$266 \times 512 \times 512$
Stanford bunny CT	$361 \times 512 \times 512$
Lobster	$80 \times 324 \times 301$
Engine	$128 \times 256 \times 256$
Boston teapot	$178 \times 256 \times 256$
Argon bubble	$640 \times 256 \times 256$

Table 2. Data set sizes.

plementation of MC and iso-splatting. Some detail in the aneurysm data set is lost due to the fine structure of the arteries, but the overall essence of the isosurface is not lost. Details could be recovered by using an adaptive sampling scheme, as described in Section 4. Figure 8 shows images from an out-of-core implementation of iso-splatting using paradigm (A). We used a Morton-order indexing scheme to perform lookups efficiently on disk. The dimensions of all data sets used in our experiments are provided in Table 2. All data sets consist of values in the range $[0, 255]$.

7 Conclusions

We have presented a new algorithm called iso-splatting for isosurface visualization. Improved extraction, storage, and rendering efficiency can be obtained through the use of points. The point samples are, in general, more cheaply stored than triangles, and they require little connectivity information, opening up possibilities for parallelism. Our results indicate that iso-splatting performs well in out-of-core settings. Since computation can be divided in various ways, implementations can be tailored relatively easily to meet resource limitations or to take advantage of emerging GPU technology. These characteristics make iso-splatting suitable for large, high-resolution data sets, when a geometry-based isosurface visualization algorithm is desirable. We point out that the point set generated by our method is useful not only for rendering purposes. Many techniques, such as the point-set-surface method [1, 6], have been developed to support geometric modeling using point sets. An additional strength of iso-splatting is its simplicity and ease of implementation.

We believe that iso-splatting opens up several avenues for future work. Iso-splatting can be extended to non-rectilinear and unstructured grids. The technique would extend well to data represented in a multiresolution format, such as adaptive mesh refinement (AMR) data [26]. Iso-splatting may be combined with standard isosurface rendering techniques to produce higher-quality images. The two methods can be combined in another way. When a user is actively changing the isovalue, iso-splatting may be used to provide greater detail while maintaining the same level of interactivity that down-sampling supports. When the user has identified an interesting isosurface, higher-quality images can be rendered using triangle-based methods in conjunction with higher-order approximations. Adaptive sampling could be performed to capture more detail in certain regions, possibly determined by interval width or gradient behavior.

The projection of points onto an isosurface is currently performed using one iteration of the Newton-Raphson method. The effectiveness of higher-order methods to perform this projection will be a topic for future research. Since iso-splatting is based on point-based rendering techniques, it faces many of the same problems that other point-based rendering methods face. As further advances in point-based rendering emerge, adjustments to the iso-splatting algorithm may be made to manage these problems elegantly.

Acknowledgments

This work was supported by the National Science Foundation under contracts ACI 9982251 and ACI 0222909, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institutes of Health under contract P20 MH60975-06A2; the Lawrence Livermore National Laboratory under contract B523818; and the Lawrence Berkeley National Laboratory. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis. The aneurysm, Boston teapot, engine, fuel injection, and lobster data sets were obtained from http://www.volvis.org/. The Stanford bunny CT data set was obtained from http://visual.nlm.nih.gov/. The primate lung data set is courtesy of E. R. Wisner, Department of Surgical and Radiological Sciences, University of California, Davis. The time-varying argon bubble data set was provided by the Center for Computational Sciences and Engineering at the Lawrence Berkeley National Laboratory. We thank Christof Nuber for his support with the out-of-core implementation.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization '01 (VIS '01)*, pages 21–28, Washington - Brussels - Tokyo, Oct. 2001. IEEE.

[2] Y. J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 293–300. IEEE, 1997.

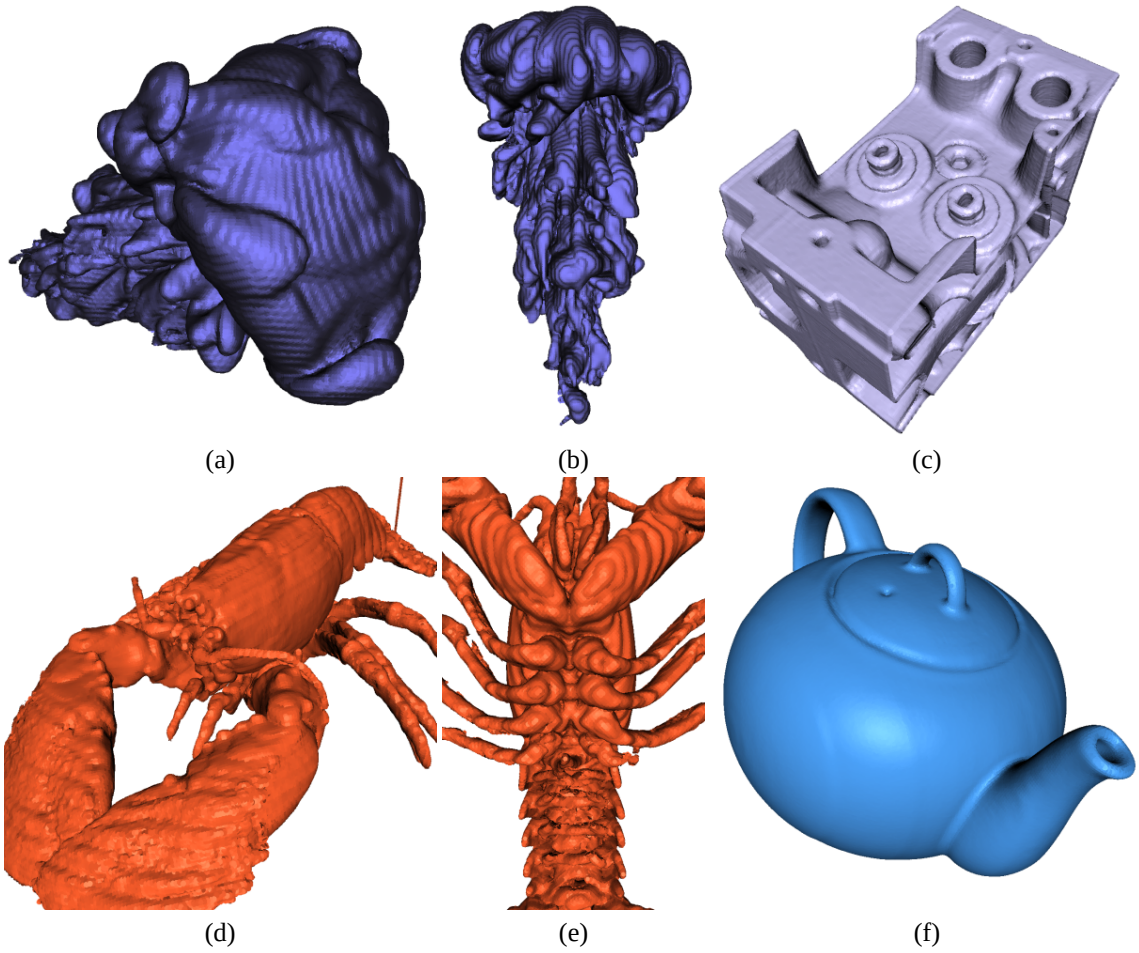


Figure 5. Images of argon bubble (a) and (b), engine (c), lobster (d) and (e), and Boston teapot (f) data sets. (See also Figure 6.)

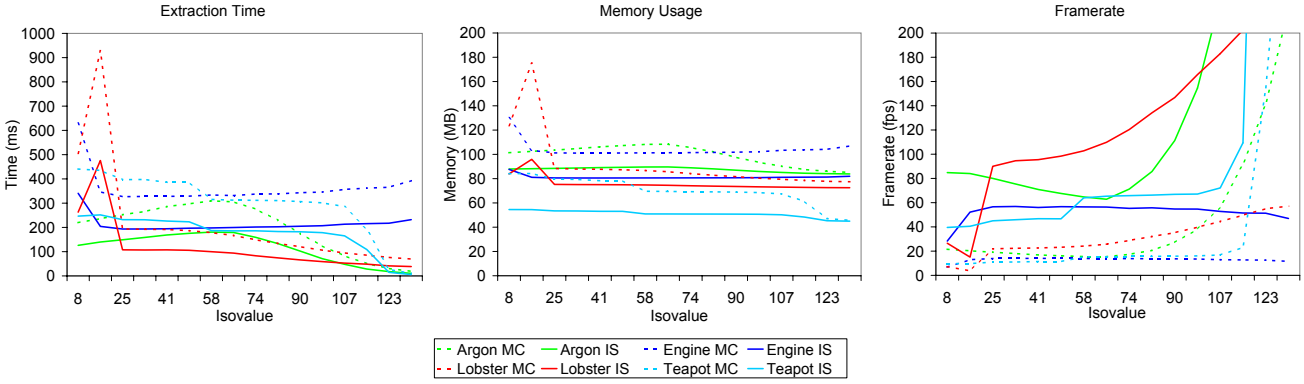
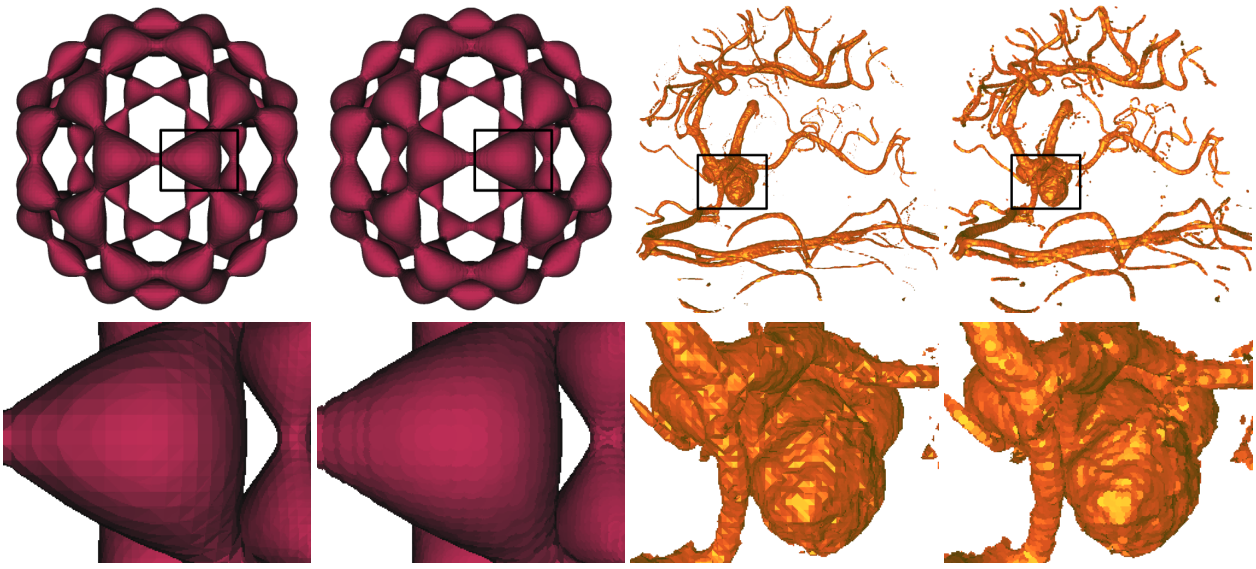
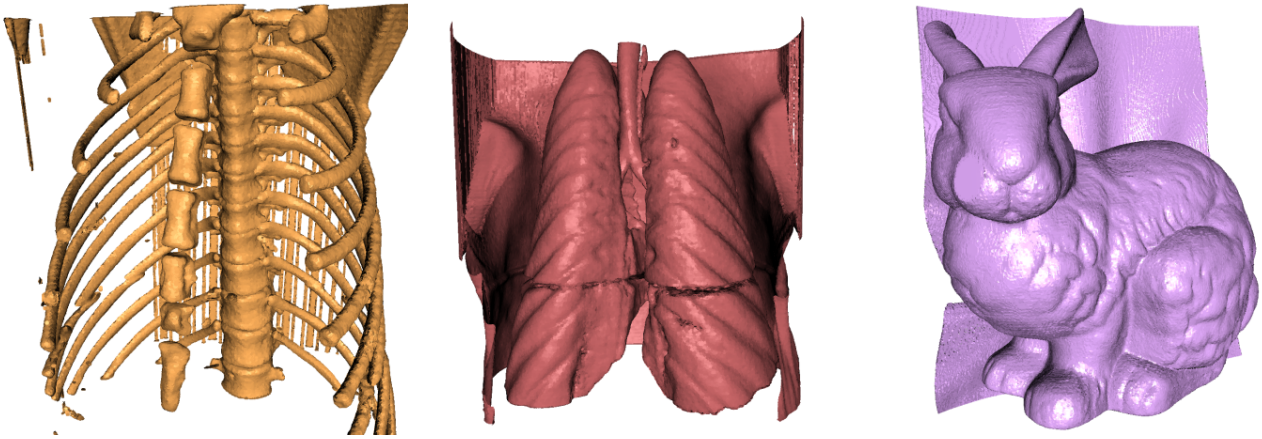


Figure 6. Extraction time (left), memory usage (middle), and framerate (right) comparisons. A standard Marching Cubes (MC) implementation and a paradigm-(C) iso-splatting (IS) implementation are compared. (See also Section 5 and Table 1.)



Marching Cubes Iso-splatting Marching Cubes Iso-splatting

Figure 7. Comparisons of Marching Cubes and iso-splatting rendering results of a Buckminsterfullerene ("bucky ball") data set and an aneurysm data set.



804,611 splats extracted in 0.38 s 2,251,374 splats extracted in 0.52 s 2,053,942 splats extracted in 0.52 s

Figure 8. Images generated with an out-of-core implementation of iso-splatting applied to a computed tomography (CT) scan of a primate lung (left and middle images) and a CT scan of the Stanford bunny (right image).

[3] Y. J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *IEEE Visualization '98*, pages 167–174. IEEE, 1998.

[4] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, Apr. 1997.

[5] H. E. Cline, W. E. Lorenson, S. Ludke, C. R. Crawford, and B. C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. In *Medical Physics*, volume 15, pages 320–327, June 1988.

[6] S. Fleishman, M. Alexa, D. Cohen-Or, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Computer Graphics*, 2003. In press.

[7] J. P. Grossman and W. J. Dally. Point sample rendering. In G. Drettakis and N. Max, editors, *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998.

[8] C. D. Hansen and P. Hinker. Massively parallel isosurface extraction. In *Proceedings Visualization '92*, pages 77–83. IEEE, Oct. 1992. LANL.

[9] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In S. Spencer, editor, *Proceedings of the 29th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-02)*, volume 21, 3 of *ACM Transactions on Graphics*, pages 339–346, New York, July 21–25 2002. ACM Press.

[10] A. Kalaiah and A. Varshney. Differential point rendering. In S. J. Gortler and K. Myszkowski, editors, *Rendering Techniques '01*, pages 139–150. Springer-Verlag, August 2001.

[11] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature-Sensitive surface extraction from volume data. In S. Spencer, editor, *Proceedings of the Annual Computer Graphics Conference (SIGGRAPH-01)*, pages 57–66, New York, Aug. 12–17 2001. ACM Press.

[12] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report 85-022, University of Carolina at Chapel Hill, 1985.

[13] P. Lindstrom. Out-of-Core simplification of large polygonal models. In S. Hoffmeyer, editor, *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)*, pages 259–262, New York, July 23–28 2000. ACMPress.

[14] Y. Livnat, H. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[15] W. E. Lorenson and H. E. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. In M. C. Stone, editor, *Siggraph 1987, Computer Graphics Proceedings*, volume 21, pages 163–169. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 1987.

[16] R. W. D. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. In *Mathematical Gazette*, volume 77, pages 354–359. 1993.

[17] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98 (VIS '98)*, pages 233–238, Washington - Brussels - Tokyo, Oct. 1998. IEEE.

[18] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 47–56, Los Angeles, CA, Aug. 12–17 2001. ACM Press / ACM SIGGRAPH.

[19] H. Pfister, J. van Baar, M. Zwicker, and M. Gross. Surfels: Surface elements as rendering primitives. In S. Hoffmeyer, editor, *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)*, pages 335–342, New York, July 23–28 2000. ACMPress.

[20] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, second edition, 1992.

[21] J. Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Helsinki University of Technology, May 2002. URL:http://www.hybrid.fi/research/splat/thesis01.pdf.

[22] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Eurographics 2002*, 2002.

[23] J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy.

[24] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[25] S. Rusinkiewicz and M. Levoy. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 63–68, New York, NY, 2001. ACM Press.

[26] G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In D. S. Ebert, J. M. Favre, and R. Peikert, editors, *Data Visualization 2001 (Proceedings of "VisSym '01")*, pages 25–34, Vienna, Austria, 2001. Springer-Verlag.

[27] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In E. Fiume, editor, *Siggraph 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press / ACM SIGGRAPH, 2001.