On The Simplification Of Radial Basis Function Fields For Volume Rendering: Some Practical Insights

Neophytos Neophytou [1] Klaus Mueller [1] Kevin T. McDonnell [1,2]

[1] Center for Visual Computing, Computer Science, Stony Brook University
[2] Department of Mathematics and Computer Science, Dowling College

ABSTRACT

Volume rendering with splatting most commonly decomposes a volume dataset into a field of overlapping radial basis function (RBF) kernels, such as Gaussians. The rendering effort is directly related to the number of RBFs in the volume and the degree of RBF overlap in the rasterization phase. Much work has been done that seeks to lower the number of RBFs by replacing a neighborhood of kernels with a single, wider kernel of the same family. The most regular decomposition for this is the octree. We argue in this paper that this simplification can adversely change the characteristics of the volume, as this substitution results in a different reconstruction of the local volume function.

To cope, we propose a special kernel, called AG-splat (AGglutinative splat), which is designed to faithfully encode the local set of RBFs it seeks to replace. At the same time, AG-splats are amenable to the same expedient rendering mechanisms than the regular RBFs. Apart from the rendering primitive itself, we also give a local algorithm, based on statistical fitting, for its encoding of homogenous and ramp-like volume areas.

1 INTRODUCTION

There are two types of approaches to reduce the complexity of an RBF-encoded dataset: local optimization and global optimization. In global optimization a simultaneous equations system is solved that results in three kernel parameters: coefficient, radius, and location. The optimization procedure is geared to finding a set of RBFs that interpolate the known grid values at good accuracy, but requires fewer RBFs for this. A recent paper by [7] has described such an algorithm for scattered data, with good success. However, global RBF-fitting algorithms can be complex and also time-consuming to conduct, due the large number of equations (one per existing grid point) involved. This can make them infeasible when the number of original datapoints is large, as is usually the case in regular-gridded volumes. On the other hand, local algorithms are much more efficient in this regard, but they do not seek, by definition, a proper substitution in the optimal sense of global algorithms. A popular approach in conjunction with splatting on regular grids dates back to a paper by Laur and Hanrahan [10]. In this paper, they created an octree hierarchy, substituting 8 octant kernels by a single larger one, according to an error threshold specified at the given level. Later, Welsh and Mueller [28] used frequency-space Gabor function matting to derive a hierarchy based on local frequency content, where individual kernel sizes were related to the spatial frequency they could encode. There are also various approaches that use wavelet analysis [17][24] and Delauney triangulation schemes [6] to pursue similar goals. What is common to all of these approaches is their use of pre-shaded rendering, where kernel color and opacity is determined in object space, prior to rasterization. This type of rendering pipeline is quite forgiving of fitting artifacts, since the density mapping with transfer functions is conducted prior to rasterization. However, as was shown in [15] and others, pre-shaded rendering leads to significant blur and loss of detail on even modest zooms. Post-shaded rendering of kernels, on the other hand, can yield much improved results, but it requires the RBF-based reconstruction of the density field (instead of color and opacity) for gradient estimation and transfer function lookup, a process which reveals the shortcomings of local kernel fitting at much lesser mercy than the pre-shaded approaches.

To illustrate these effects, consider Fig. 1 where we show the results of a few experiments we have conducted, both in 1D and for 3D volume rendering with post-shaded splatting, using the image-aligned splatting approach described in [16]. The figure caption holds a detailed explanation for the effects observed. But what we learn from these experiments is that simply replacing a dense set of kernels in uniform regions by a wider one of the same basic shape yields a poor fit of the original function. Of course, these adverse effects could be overcome by running a global optimization procedure, which would likely modify not only the kernel radii but also their locations, resulting in an irregular-gridded dataset. For this work, however, our goal was to retain the original lattice structure and rather modify the kernel itself. By retaining the overall lattice structure, we also retain some of the associated regularity, which brings advantages for both storage and bandwidth. It also keeps a tighter control on the inter-lattice ripple artifacts that is a hallmark of radially symmetric functions, when placed on axis-orthogonal grids (see Fig. 1a). In fact, the new AG-splat kernel we are proposing here, overcomes even these imperfections.

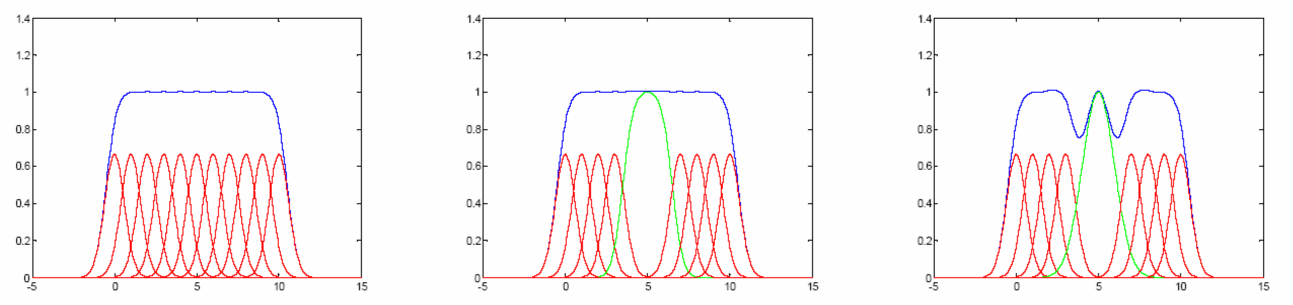
We should note that, although we focus in this paper on general RBF field simplification using irregular decompositions, the special case of constructing an octree via collapsing eight child RBFs into one larger parent RBF, as was done in the seminal paper by Laur and Hanrahan [10], suffers from similar problems. While the adverse effects and artifacts shown in Fig. 1c may be less pronounced, they potentially still exist.

In the work reported in this paper, we seek to compress uniform and ramp-like regions and represent them by a single rendering primitive, the AG-splat. This eliminates the excessive overdraw of overlapping kernels in the non-simplified configuration and is the major source of speedup, without sacrificing rendering quality by less-than-optimal wider kernel substitution.

Our paper is structured as follows. First, in Section 2, we describe some of the relevant previous work on the subject of RBF rendering, fitting, and encoding, while Section 3 describes our approach used for this. Section 3 also discusses our new AG-RBF kernels used as rendering primitive. Section 4 then outlines the rendering engine best suited to splat our rendering primitives. Finally, Sections 5 and 6 report on results and future prospects.

2 RELATED WORK

In the following, we shall concentrate on related work in volumetric RBF splatting. One of the first volume rendering algorithms in fact was splatting [27]. But while early splatting approaches suffered from blurring as well as popping artifacts during view transitions, the introduction of post-rasterization classification and shading [15], in conjunction with the image-aligned kernel-slicing approach [16], enabled volume rendering with splatting at high quality in both static and dynamic viewing modes. This approach interpolates the density function on a set of image-aligned slices, from which gradients, colors, and opacities can be either computed



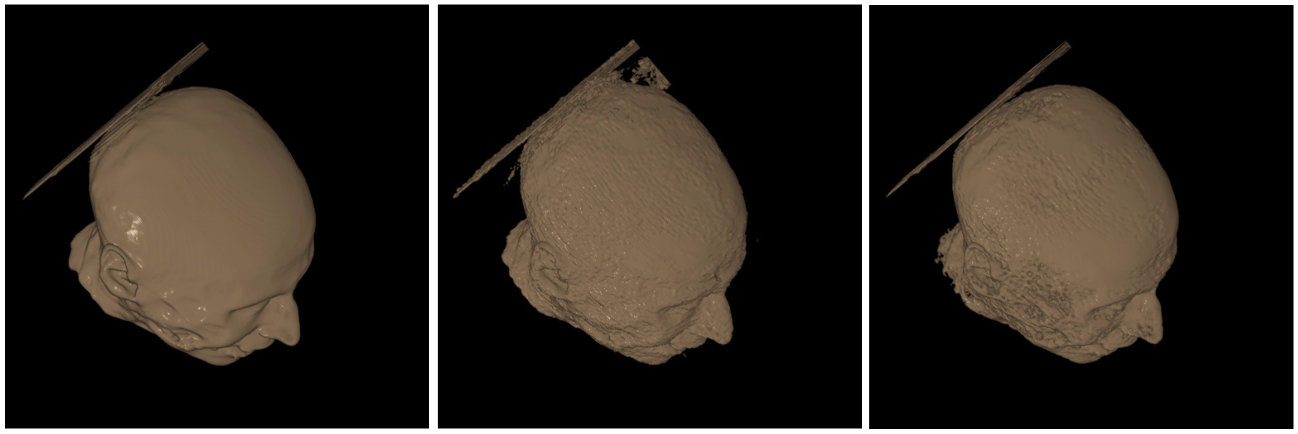


Figure 1: Top row: 1D experiments with regular kernel substitutions. (a) the equally-spaced field of uniform-valued Gaussian kernels (red) and their interpolated function (blue). We observe a small ripple on top of an otherwise constant-valued reconstruction. (b) replacing the center four Gaussians by a single Gaussian whose amplitude is determined by the constant function it needs to reconstruct. (c) the function reconstructed by the mixed set of Gaussians. We can easily observe the overwhelming lack of fit. Had we chosen the center Gaussian wider, the overshoot already slighly indicated at the far sides would increase in size

Bottom row: 3D volume rendering experiements with regular kernel substitutions. Similar criteria for simplication than in the 1D case were chosen, simplifying unofmr areas with wider kernels. The poor estimation of the volume function by the substitutions with regular kernels manifests itself as severe rendering artifacts in post-shaded rendering mode. (d) CT head rendered with a volume composed of kernels of equal size. (e) and (f) CT head rendered with the volume containg the simplified aggregation of kernels. The exessive density ripple effects caused by placing wider kernels into the inner more homogeneous skin layers affect the reconstructed density field at the surface. No image with smooth surface can be obtained -- the holes obtained with the theoretical correct setting (also compare (c)) are substitued by bumps when the kernel is made wider (analogous to the obvershoot observed at the side lobe in (c))

or derived from the transfer functions. In essence this algorithm is like tracing a set of simultaneous rays, where the skipping of empty volume regions is implied by the point-based representation A fast GPU-based implementation has also been described [18].

A hallmark of splatting is its object-centric (or better, voxel-centric) rendering paradigm, which enables the efficient and rendering of sparse and irregularly-shaped volumetric objects and datasets [14][4]. If all one desires are iso-surfaces, then a favorable approach is to convert a volume into a set of surface points on the fly and render these with surface splatting [11][25]. This eliminates the need for reconstructing the surface in volume space by interpolating the local neighborhood of 3D RBFs. On the other hand, if the goal is to create a composited or summed integration of the volume data, say, along the viewing direction, then it is preferable to retain the volumetric RBF representation during the rendering. Our paper focuses on this latter task. A recent trend has gone into representing regular-gridded volumes on more efficient lattices, such as the body-centered cartesian (BCC) grid. It allows the number of basis functions to be reduced by almost 30% in 3D [23] and 50% in 4D data [19], which can then be rendered via splatting (although raycasting is also a possibility, see [3][2]). Our approach constructs a more feature-oriented and only semi-regular decomposition, as opposed to the more regular grid decompositions created by hierarchical techniques proposed for splatting previously, such as octree representation by Laur and Hanrahan [10], the wavelet-based approaches by [17][24], and the frequency-space Gabor-function approach by [28]. Presently, our scheme does not offer a hierarchical representation, although such an extension would be possible. At the current stage, we seek a decomposition that, along with the AG-RBF, simplifies visually homogeneous features as good fidelity, while eliminating the risk of adverse visual effects that appear when rendering these with ordinary RBF-type primitives.

Aside from the regular grids, there have also been various algorithms that use the splatting paradigm for scattered data. Meredith and Ma [13] use spherical kernels that fit into a cube and are mapped to textured squares for projection. Jang et al. [8] fill a cell with one or more ellipsoidal kernels, which they render with elliptical splats. A similar approach that was also taken by Mao [12]. Hopf et al. [5] apply splatting to very large time-varying datasets and render the data as anti-aliased GL points. Jang et al. [7][26] run an optimization algorithm to encode their volumes into a minimal set of spherical RBFs, still irregularly spaced. They render them in post-shaded mode using a GPU-based slice-plane kernel-evaluation approach. In contrast, we [20] have given a significantly faster GPU-based slice-plane splatting approach to accomplish this task, and extended the rendering capabilities to elliptical basis functions.

Our approach is also somewhat related to that of Pajarola [21], who proposed a family of special types of surface splats for better surface function fitting. On one end of the spectrum the splats have a similar shape than a Gaussian, on the other, they look like a rounded box. These functions allow better control over the kernel blending behavior, such as sharpness, smoothness, and so on.

3 Kernel Fitting

The main goal of our approach is to create an encoding of the volume that both minimizes the overdraw which is inherent in any RBF encoding method, by creating larger homogenous splats, but also lessens the ripple artifacts that occur in the region between any two RBFs in the volume. Therefore, our kernel agglutination (merging) approach uses an encoding process which favours bigger composite kernels, and eliminates as many interface regions as possible inside homogenous volume segments.

In the remaining of this section we will provide more details on the detection of homogenous and ramp-like regions and discuss the agglutination process which is optimized to reduce overdraw. Then we introduce the rendering primitives that will be used during reconstruction and are geared towards reducing the ripple artifacts in the interface regions.

3.1 Least Squares Detection of Constant and Ramp-like areas

We employ a least squares approximation method to locate and encode constant regions of variable sizes, as well as ramp-like areas of varying sizes and slopes. In the context of this work, a volume segment is considered to be *homogenous* if the difference of every voxel in the region from the mean of the region is bound within a small error. This means that the density function of the volume in this region can be fitted to a constant function in the least square sense. More specifically,

$$\sum_{i=0}^{N-1} \frac{(x_i - \bar{x})^2}{N} \le \varepsilon \qquad (1)$$

for all pixels x_i *(i=0..N-1)* in the given volume region and a small error ε. Similarly, a region of the volume is classified as a *ramp* along any of the x, y, or z directions if it's gradient in the given direction is constant in the least square sense.

Further, our definition of a *homogenous* or *ramp* region is restricted to the shape of an axis aligned box of variable sizes along each direction. Therefore, the primitives used for reconstruction will have to fit in any box of variable side sizes. These are described in a later sub-section.

The task of computing homogenous regions in three dimensions requires an enormous computational effort which involves scanning through the entire volume for all possible N^3 box size combinations, and then storing this in an $N^3 \times N^3$ capacity data structure to be used in the agglutination process.

Instead, we take the shortcut of computing the homogenous regions of the given volume dataset only as a 1D function along each axis direction separately. This information will then be used to merge the 1-D "sausage"-type cells into 3-D agglutinated homogenous regions. Figure 2 shows the resulting sets for the ctHead dataset, using a chi-square tolerance value of 0.5.

The process of detecting ramp like regions is very similar. The same detection algorithm is used, but it is applied on the normalized gradient magnitude along each volume direction, which is pre-computed in the volumes G_X, G_Y and G_Z.

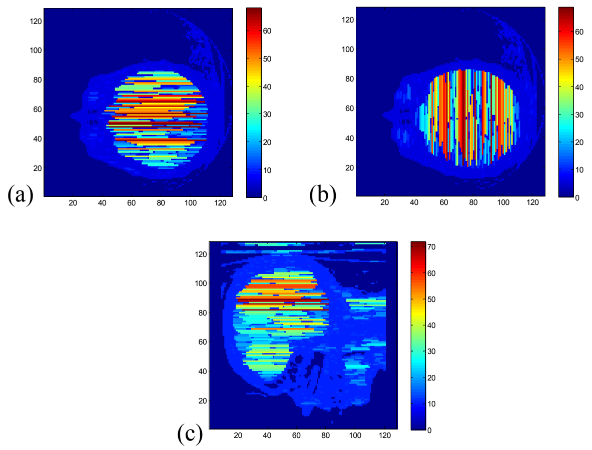


Figure 2: Constant areas inside the ctHead dataset detected in 1D along (a) X-direction, (b) Y-direction, (c) Z-direction. The color of each line depicts the size of a continuous homogenous region. The darkest blue color in the non-empty areas of the volume shows that these voxels do not belong in a homogenous region since the size of their associated region is 1.

3.2 The Agglutination Process

Our proposed method encodes each homogenous region as a composite kernel, which will then be modulated by a flat value for the entire region during rendering.

However, the task of computing the optimal set of 3D homogenous regions in any given volume is quite complex and presents interesting challenges. It is not adequate to just locate homogenous boxes of all possible sizes in the volume, by applying the least square fit test for different sizes throughout the entire dataset. In addition, one also has to find a fitting combination of these boxes that maximizes the gain metric, which in our case is the reduction of overdraw by eliminating similar overlapping regions. That is, we need to create an encoding for the volume that uses the least number of agglutinated 3D splats.

It is apparent that this is an NP-complete problem, and efforts to provide a rigorous complexity proof [1] would hint to 2D and 3D generalizations of the knapsack problem. Since an optimal solution of polynomial complexity cannot be found, we have to be restricted to approximate solutions that use greedy approaches along with some practical heuristics.

In this text, we present a relatively simplified solution to this problem, which has nevertheless provided us with satisfactory experimental results.

We first define our *gain function* so that the agglutination process will result in reduced overdraw and eliminates as many point-to-point interfaces as possible, by including them inside the composite splat. Thus, the desired gain function will not always favor the largest composite splats but, rather, the splats that are better spread over their 2-D and 3-D regions. Let us define

$$\begin{aligned} Gain(Sx, Sy, Sz) &= ((Sx-1) \times Sy \times Sz) \\ &\quad + (Sx \times (Sy-1) \times Sz) \\ &\quad + (Sx \times Sy \times (Sz-1)) \end{aligned} \qquad (2)$$

where *Gain* is a function of the sizes of the agglutinated homogenous region along the Z,Y and X direction. This is better illustrated in Figure 3, where the gains of variable sized boxes are compared. We can clearly see that well-spread composites are favored better than elongated ones.

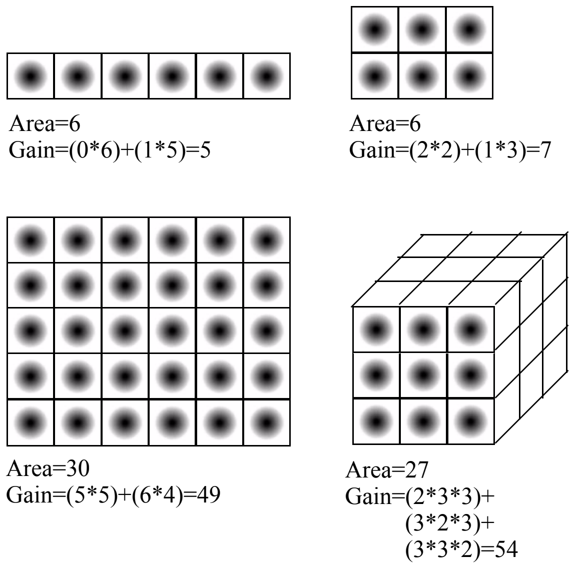


Figure 3: Comparison of different shapes of homogenous regions and the gains when they are substituted by an agglutinated splat. We can see that smaller boxes that are better spread along the 3 directions yield a better gain by eliminating more neighboring point boundaries.

The merging process itself uses a greedy approach. It utilizes heuristics based on the 1D homogenous region information to decide where to start building the next agglutinated kernel. The least-square detection process as described in the previous sub-section, results in a set of homogeneity volumes. The volumes H_X, H_Y, and H_Z store the length of the longest homogenous run that each voxel is part of, as illustrated in the slices shown in Figure 2a,b,c, as well as the beginning and ending position of that run to be used during the merge.

First, we place the seed at the voxel position V_i, which maximizes the product $H_X(V_i)$ * $H_Y(V_i)$ * $H_Z(V_i)$. Intuitively, there is a high likelihood that the volume data at this location will have a large homogenous region in all 3 directions, since it satisfies the maximum homogeneity metric along each of the three axes directions. In the next step the algorithm will greedily keep growing the seed in each of the three directions to maximize the gain metric defined in Equation (2). The resulting agglutinated kernel (AG-Splat) is then stored and the homogeneity volumes are updated to account for the deletion of the new kernel. The process repeats by placing the next seed at voxel position V_j which maximizes $H_X(V_j)$ * $H_Y(V_j)$ * $H_Z(V_j)$ after the first update. This process will terminate when only single voxels are left in the dataset, resulting in the $\mathrm{Max}(H_X(V)$ * $H_Y(V)$ * $H_Z(V))=1$. An overview of the above process is illustrated in Figure 4, which also shows a slice of the final agglutinated volume and a wire-frame slice of the final volume sent to the rendering engine.

3.3 The AG-Splat Reconstruction Primitives

As mentioned above, the purpose of the AG-Splat kernel is to exactly replicate all the effects of the group of kernels it is replacing in a constant or homogenous value region of the volume. Therefore, it cannot be any form of a simple Gaussian function, but instead, it always has to be a composite kernel which consists of a sum of equally spaced Gaussians functions. In 2D, which applies to the actual kernel values at a volume slice, the value of the kernel an any point will be:

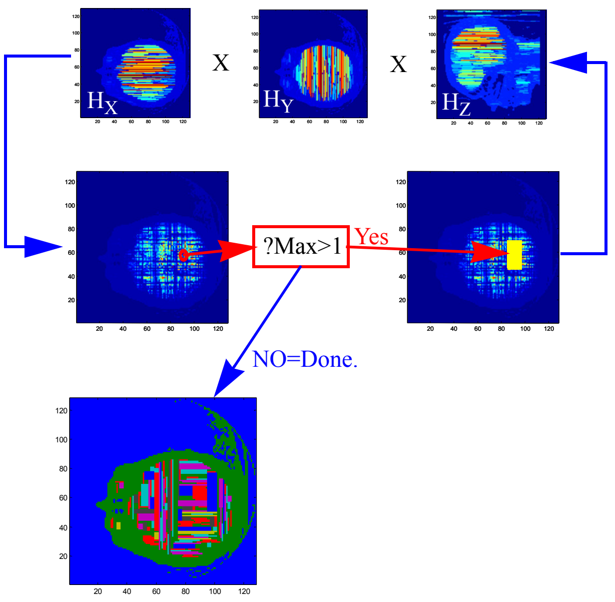


Figure 4: Overview of the agglutination process. At each step, the seed is placed at the position with maximum H_X* H_Y* H_Z. Next, the seed is grown to maximize the gain metric, using the information from the homogeneity volumes. This process repeats while more homogenous areas (with length>1) are available in the volume. When no more large areas are available, then the product becomes $\mathrm{Max}(H_X$* H_Y* $H_Z)<=1$, at which point the agglutination process terminates.

$$AG(x, y) = \sum_{ix = -(xlen)/2}^{(xlen)/2} \sum_{iy = -(ylen)/2}^{(ylen)/2} G(x-ix, y-iy) \qquad (3)$$

where *xlen* and *ylen* are the lengths of the box along the respective directions, and *G* is the RBF primitive that was used in the original dataset. Figure 5 illustrates how the AG-Splat kernel interacts with other primitives in the data. We can see that the blue and green AG kernels of different sizes integrate seamlessly with each other as well as with the single splat primitives shown in red.

The computation of the AG-Kernel as described in Equation (3) is certainly not practical, and almost defeats the purpose of our efforts. However, after taking a closer look at the kernel in Figure 6, it can be obvious that only 3 single splats on each side of

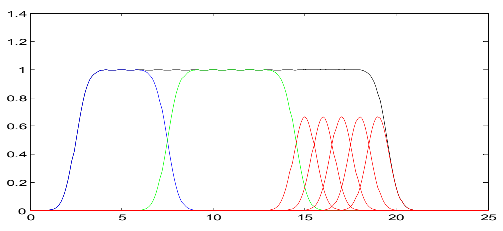


Figure 5: AG-Splat kernel interactions. Since AG-Splat emulates exactly the effects of the underlying grid aligned primitives it can interface seemlessly with both other AG-kernels (blue and green) of various lengths as well as single splat primitives (red) of the original dataset. The value of the reconstructed function is shown in black ink.

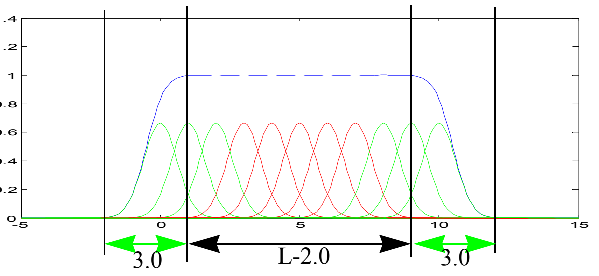


Figure 6: The AG-Splat Kernel in 1D. The length of the encoded constant region is L. There is a 3 unit fallout, which acts as the interface region with neighboring kernels.

the boundaries (shown in green) have to be computed. The constant region can be replaced by 1.0, and a very simple fragment program can be used to evaluate this in 1D and in 2D. In fact, since the AG-Kernel is separable, we can evaluate the 2D kernel by first evaluating the corresponding components of the kernel in 1D and then multiplying the results. This enables the computation of the composite kernel at any pixel using at most 6 gaussian evaluations, which involve one exponent for each. The benefits of this are more apparent as the size of the AG-Kernels increases.

An alternative way of doing this would be to precompute a generic AG-Kernel texture, and then use tex-coord arithmetic to direct the sampler to the correct section of the texture, since a large part of the kernel is a repeating pattern. This will require just one texture lookup per pixel, which is the standard cost of any splatting approach.

4 Rendering Engine

In the previous sections we have introduced a volumetric encoding scheme which utilizes AG-Splats. This composite primitives are used to encode flat homogenous regions that can be enclosed in any axis aligned variable sized box. The AG-Splat rendering primitive is constructed in a way that simulates the existence of all the initial RBF primitives at every voxel location enclosed in the AG-Splat. For the visualization of AG-Splat encoded volumes we employ an approach best suited to efficiently render this type of axis aligned primitives: Shear-Warp Splatting.

4.1 The Shear-Warp Splatting technique

The Shear-Warp algorithm, conceived by Lacroute and Levoy [9] in 1994 is still widely known as the fastest software rendering algorithm for volume visualization. In shear-warp, the volume is rendered by a simultaneous traversal of RLE-encoded voxel and pixel runs, where opaque pixels and transparent voxels are efficiently skipped during these traversals. Further speed comes from the fact that a set of interpolation weights is pre-computed per volume slice and stays constant for all voxels in that slice. The caveat is that the image must first be rendered from a sheared volume onto a so-called base-plane, aligned with the volume slice most parallel to the true image plane. After completing the base-plane rendering, the base plane image is warped onto the true image plane and the resulting image is displayed.

Our adaptation of the shear-warp algorithm to splatting draws from the basics of the original algorithm, but adds several extensions that are possible with a splatting approach, as well as the capabilities of a GPU accelerated framework.

The core functionality of the proposed rendering engine comes from previous work on our GPU accelerated image aligned splatting algorithm. The existing framework has been modified to take advantage of the shear-warp factorization of the viewing transform, which eliminates the need for the visibility sorting step at the expense of having three separate encodings of the volume resident in GPU memory. There is, however, a real gain in performance which spans from the reduced overdraw that is achieved by the AG encoding method. The additional costs for the storage of duplicated volumes are also offset from the dramatic data reduction achieved by the use of the agglutinated kernels.

An overview of the rendering process is given in Figure 7. After the agglutinated dataset is produced at a pre-processing stage, three encodings of the volume, one for each axes aligned viewing direction is loaded into the GPU in the form of vertex array buffers. For every AG-Splat primitive in the volume, we have stored its x,y,z position along with the associated density value, and three additional values which are passed as texture coordinates, the length, width and height of the agglutinated splat.

The rendering process consists of the following steps:

- First, the shear and warp factorization matrices are created from the given view transform matrix, and a viewing direction is decided.
- We use the OpenGL extensions for vertex and texture arrays for the efficient passing of all the parameters, while every slice of the volume is kept in a separate vertex array. During the rendering of each slice, all the associated vertex arrays that affect that given slice are called using glDrawArrays calls.
- The activated vertex program transforms the position of the AGL-Splat, using the shearing matrix, and creates the image aligned polygon which will hold the splat to be rasterized. It also computes a scaling coefficient which will modulate the density value based on the distance of the point from the currently rendering slice. Four vertices are used for each splat, and they will be transformed into the four corners of a texture-mapped polygon, using the shearing transform and the size information given by the agglutination process.
- The activated fragment program evaluates the AG kernel as described in Section 3.3 for every pixel of the current polygon.

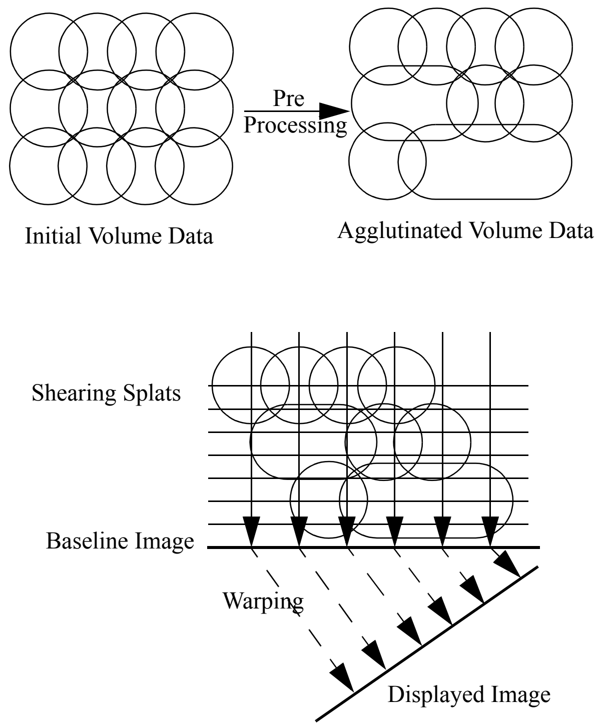


Figure 7: Overview of the Shear-Warp splatting algorithm for agglutinated volume datasets.

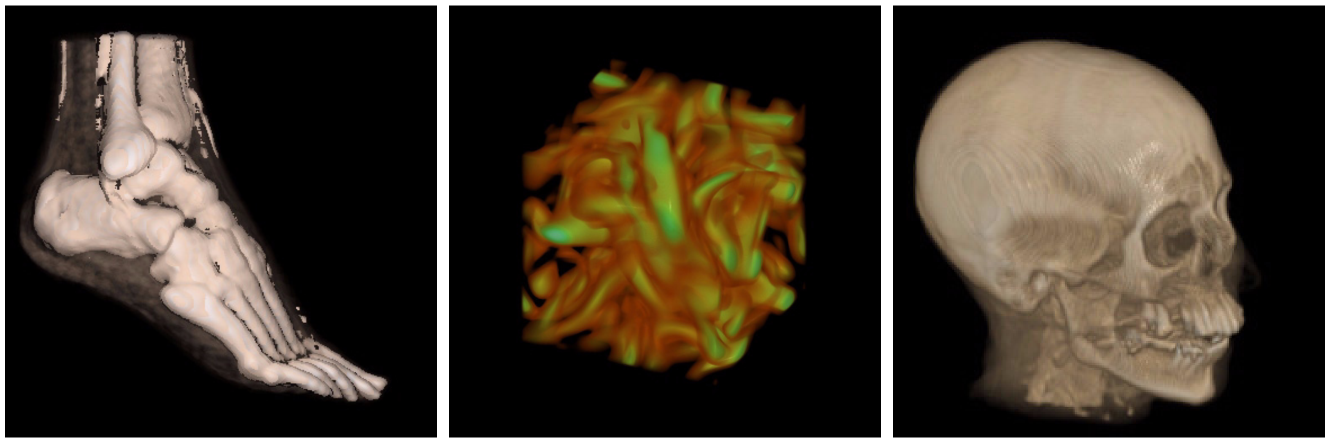


Figure 8: Some results obtained with our renderer

- Every finished slice is then shaded and composited using the same shading fragment shaders described in [18]. The shading process has been slightly modified to adjust the gradient computations using the shear-warp transformations, in order to produce consistent images across different volume encodings and avoid possible popping.
- After the intermediate slice is shaded, it is then warped and composited onto the final image plane.

4.2 AG-Splatting Specific Codifications

One of the main disadvantages of using the shear-warp factorization is the inconsistency of the sampling rate along the z-direction for varying angles, leading to undersampling for some angles, and eventually aliasing artifacts. The latest shear-warp approaches [22] resolve this problem by introducing resampled intermediate slices and effectively doubling the sampling rate along the z-direction.

Adaptive Sampling along Z direction. Slice-based splatting, on the other hand, is more flexible in terms of the sampling rate along the z-direction, because of the encoding of the volume as overlapping kernels, which are intersected by arbitrary slices along the axis direction. In our current implementation we can adoptively vary the sampling rate from 0.5 to 1.0, depending on the viewing angle to compensate for the possibility of aliasing when viewing under specific angles.

Simultaneous Multi-Channel Rendering. A very effective optimization that we have applied to our previous image aligned GPU splatter was the ability to render on the RGBA channels of the image buffer independently, having effectively 4 simultaneous rendering buffers for the cost of rendering to one. This resulted in speeding up the rendering process up to 3 times. We have migrated this facility to the current implementation and additionally enabled it to support the adaptive sampling facility that is necessary in the context of shear-warp splatting. This required some additional accounting which is done when preparing each slice for rendering.

Efficient Empty-space Skipping and Opacity Culling. These facilities were directly migrated from the existing image aligned GPU splatter [18] and they make full use of the early-z culling hardware optimization provided at both NVidia and ATI graphics boards. A special tagging of the current splats a depth values in the z-buffer allows for the isolation of the recently splatted fragments while empty spaces are culled before entering the shading process. Further use of the Depth Range extension allows for the elimination of opaque fragments from both the splatting and the rendering process.

5 Results

We have obtained the following results, on a Pentium 4 running at 3 GHz and 1 GB of RAM. The graphics board is an NVidia Quadro FX 3400 with 256MB RAM, which is equivalent to a GeForce 6800 GT board. We tested three datasets of representative origin and source: the vortex dataset, a CT foot and a CT head. The framerate for the vortex dataset with regular shearwarp splatting (without agglutinative splats) was 6.0 fps, while with agglutinative splats it reached 8.3 fps. For the semi-transparent head we got 5.3 fps for normal rendering and 9.3 fps for the new rendering method. Finally, for the semitransparent foot we reached 6.8 fps for the standard case and with AG-splats we reached 10.9 fps. So generally, the speedup was just below 2 and grew for semi-transparent rendering since it can take better advantage of the encoding. Figure 8 shows some of the images we obtained with our new renderer.

6 Conclusions and Future Work

We have described a variant to volumetric point-based rendering that is specifically targeted to reduce the overdraw associated with the need to blend adjacent kernels. For this capitalize on homogeneous and ramp-like regions and encode these by a special kernel called AG-splat. Our approach is the proper way to do this since just encoding these regions by a larger splat will modify the underlying function which can have adverse rendering effects. It is interesting that this has not been noticed before, and we believe that this is mainly due to the fact that preshaded splatting hides the inaccuracies well, while the higher-quality post-shaded rendering pipeline does not. We should note that, although we have focused in this paper on general RBF field simplification using irregular decompositions, the special case of constructing an octree via collapsing eight child RBFs into one larger parent RBF, as was done in the seminal paper by Laur and Hanrahan [10], potentially also reveals similar problems, although the artifacts may be less pronounced. Our paper is meant to make researchers aware of these effects, and provide some guidelines to cope with them.

For the future, we plan to extend the framework to a hierarchical approach, where nodes further up the tree have progressively higher error thresholds in the fitting stage. We also plan to experiment with non-axis aligned and more freeform primitives to capture more types of regularities for encoding.

ACKNOWLEDGMENTS

This research was supported, in part, by NSF CAREER grant ACI-0093157 and NIH grant 5R21EB004099-02.

REFERENCES

[1] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[2] B. Csébfalvi, "Prefiltered Gaussian Reconstruction for High-Quality Rendering of Volumetric Data Sampled on a Body-Centered Cubic Grid," *IEEE Visualization 2005*, pp. 311-318.

[3] A. Entezari, R. Dyer, T. Möller, "Linear and Cubic Box Splines for the Body Centered Cubic Lattice", *IEEE Visualization 2004*, pp. 11-18, 2004.

[4] F. Vega Higuera, P. Hastreiter, R. Fahlbusch, and G. Greiner, "High Performance Volume Splatting for Visualization of Neurovascular Data," *Proc. IEEE Visualization 2005.*

[5] M. Hopf, M. Luttenberger, T. Ertl, "Hierarchical splatting of scattered 4D data," *IEEE Computer Graphics & Applications*, 24(4): 64-72, 2004.

[6] W. Hong, N. Neophytou, K. Mueller, and A. Kaufman "Constructing 3D Elliptical Gaussians for Irregular Data", *Mathematical Foundations of Scientific Visualization, Comp. Graphics, and Massive Data Exploration*, Springer-Verlag, Germany 2006.

[7] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. Ebert, K. Gaither, T. Ertl, "Interactively visualizing procedurally encoded scalar fields," *IEEE/EG Visualization Symp. 2004*.

[8] J. Jang, W. Ribarsky, C. D. Shaw, N. Faust, "View-dependent multiresolution splatting of non-uniform data," *Proc. IEEE/EG Visualization Symp. 2002*.

[9] P. Lacroute, M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *SIGGRAPH 1994*, pp. 451-458.

[10] D. Laur, P. Hanrahan, "Hierarchical splatting: a progressive refinement algorithm for volume rendering," *SIGGRAPH 1991*, pp. 285-288.

[11] Y. Livnat, X. Tricoche, "Interactive point-based isosurface extraction," *IEEE Visualization 2004.*

[12] X. Mao, "Splatting of curvilinear volumes," *IEEE Trans. Visualization and Computer Graphics*, 2(2): 156-170, 1996.

[13] J. Meredith, K.-L. Ma, "Multiresolution view-dependend splat based volume rendering of large irregular data," *Symp. Parallel and Large-Data Visualization and Graphics 2001*.

[14] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, "A practical comparison of popular volume rendering algorithms," *2000 Symp. on Volume Rendering*.

[15] K. Mueller, T. Möller, and R. Crawfis, "Splatting without the blur," *Proc. IEEE Visualization*, 1999.

[16] K. Mueller, N. Shareef, J. Huang, R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Trans. on Visualization and Computer Graphics*, 5(2): 116-134, 1999.

[17] Muraki, S., "Multiscale 3D edge representation of volume data by a DOG wavelet," *1994 Symp. on Volume Visualization*, pp. 35-42.

[18] N. Neophytou and K. Mueller, "GPU accelerated image aligned splatting," *IEEE-TCVG/EG Workshop on Volume Graphics 2005*.

[19] N. Neophytou and K. Mueller, "Space-time points: 4D Splatting on effcient grids," *Symposium on Volume Visualization and Graphics '02*, pp. 97-106, 2002.

[20] N. Neophytou, K. Mueller, K. McDonnell, W. Hong, X. Guan, H. Qin, and A. Kaufman, "GPU-accelerated volume splatting with elliptical RBFs," (to be presented), J*oint Eurographics - IEEE TCVG Symposium on Visualization 2006* (EuroVis'06), Lisbon, Portugal, May 2006

[21] R. Pajarola, M. Sainz, P. Guidotti, "Confetti: Object-Space Point Blending and Splatting," *IEEE Trans. Vis. Comput. Graph*. 10(5), pp. 598-608, 2004.

[22] J. Sweeney and K. Mueller, "Shear-Warp Deluxe: The Shear-Warp algorithm revisited," *Joint Eurographics - IEEE TCVG Symposium on Visualization 2002*, pp. 95-104, May 2002.

[23] T. Theussl, T. Möller, and E. Gröller, "Optimal regular volume sampling,"*Proc. Visualization'01*, pp. 91-98, 2001.

[24] M. Westenberg and J. Roerdink, "X-Ray Volume Rendering by Hierarchical Wavelet Splatting," *Proc. 15th Internl' Conference on PatternRecognition,* pp. 163-166, 2000.

[25] B. von Rymon-Lipinski, N. Hanssen, T. Jansen, L. Ritter, E. Keeve, "Efficient point-based isosurface exploration using the span-triangle," *Proc. IEEE Visualization* 2004.

[26] M. Weiler, R. P. Botchen, J. Huang and Y. Jang, "Hardware-assisted feature analysis and visualization of procedurally encoded multifield volumetric data," *IEEE Computer Graphics & Applications,* 25(5): 72-81, 2005.

[27] L. Westover, "Footprint evaluation for volume rendering," *Proc. SIGGRAPH* 1990.

[28] Welsh, K. Mueller, "A Frequency-Sensitive Point Hierarchy for Images and Volumes," *IEEE Visualization 2003*, pp. 425-432