Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

Interactively Visualizing Procedurally Encoded Scalar Fields

Yun Jang[1], Manfred Weiler[2], Matthias Hopf[2], Jingshu Huang[1], David S. Ebert[1], Kelly P. Gaither[3] and Thomas Ertl[2]

[1] Purdue University,{jangy, jhuang2, ebertd}@purdue.edu
[2] University of Stuttgart,{weiler, hopf, ertl}@vis.uni-stuttgart.de
[3] University of Texas,kelly@tacc.utexas.edu

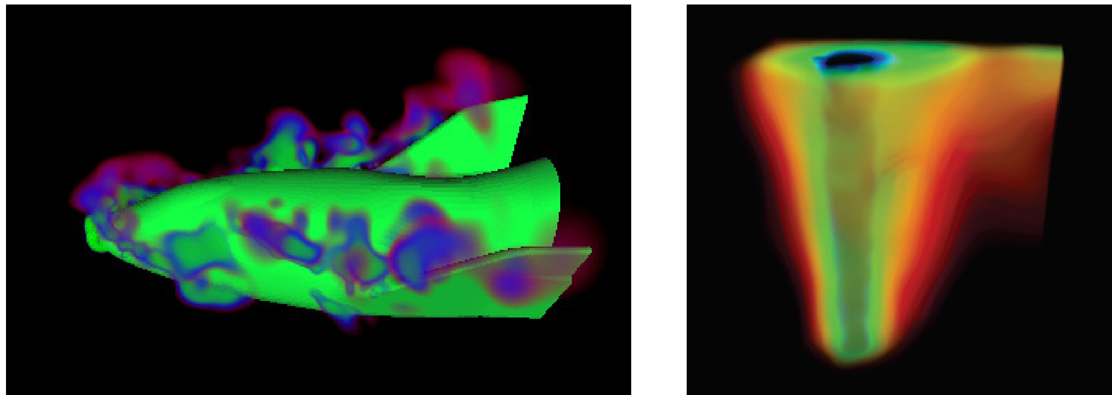

Figure 1: *RBF reconstruction of unstructured CFD data. (a) Volume rendering of 1,943,383 tetrahedral shock data set using 2,932 RBF functions. (b) Volume rendering of a 156,642 tetrahedral oil reservoir data set using 222 RBF functions organized in a hierarchy of 49 cells.*

Abstract

While interactive visualization of rectilinear gridded volume data sets can now be accomplished using texture mapping hardware on commodity PCs, interactive rendering and exploration of large scattered or unstructured data sets is still a challenging problem. We have developed a new approach that allows the interactive rendering and navigation of procedurally-encoded 3D scalar fields by reconstructing these fields on PC class graphics processing units. Since the radial basis functions (RBFs) we use for encoding can provide a compact representation of volumetric scalar fields, the large grid/mesh traditionally needed for rendering is no longer required and ceases to be a data transfer and computational bottleneck during rendering. Our new approach will interactively render RBF encoded data obtained from arbitrary volume data sets, including both structured volume models and unstructured scattered volume models. This procedural reconstruction of large data sets is flexible, extensible, and can take advantage of the Moore's Law cubed increase in performance of graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Interactive Rendering, Scientific Visualization, Radial Basis Functions

1. Introduction

Most visualization applications are faced with a *data deluge:* scanners, sensors, and scientific simulations are now generating enormous amounts of data that must be rendered, visualized, interactively manipulated and explored. These massive data sets can have arbitrary structure and organization, ranging from easily rendered, rectilinear 3D grids, to tetrahedral and hexahedral grids, to arbitrary scattered data with no topological connectivity. Generating accurate and effective

© The Eurographics Association 2004.

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

visual representations of the information contained within these large data sets is the primary focus during the visualization process, and should therefore drive the computational and rendering efforts.

For many applications, however, data transfer bandwidth has replaced computational power as the bottleneck. Fortunately, the flexibility and speed of commodity computer graphics hardware has also increased tremendously, providing a super fast, programmable, vectorized, multiprocessor for rendering. Taking advantage of this Graphics Processing Unit (GPU) creates a unique set of challenges when visualizing, interacting with, and rendering these enormous data sets. These challenges are:

1. Transferring data from the CPU and main memory to the GPU.
2. Adapting the structure of the data to the capabilities of the graphics hardware.

We have taken a new approach to solve these two problems for volumetric scalar data sets. To eliminate the data transfer bottleneck from the CPU's main memory to the GPU, we are procedurally encoding the volumetric data using a small set of Radial Basis Functions (RBFs). This allows us to provide a unified representation for arbitrary volumetric data sets independent of the underlying topology, thus eliminating the dependence on the computational grid. The RBF representation allows us to fully represent the original data within an acceptable error tolerance, and enables us to solve the second problem of adapting the structure of the underlying data to the capabilities of the graphics hardware. The capability of commodity PC graphics hardware to interactively reconstruct and render data from this functional representation provides a very powerful tool for visualizing procedurally encoded volumes. Additionally, the entire functional representation can reside in texture memory, local to the GPU, almost eliminating the need for data transfer between the CPU and the GPU.

We first review related work and describe the use of RBFs for encoding volume data. We then discuss our interactive rendering and reconstruction system in detail and, finally, present some results achievable by our system.

2. Related Work

RBFs are simply one choice for encoding data. Compared to other data compression techniques like wavelets [NS01, BDHJ00] or iterated function systems [BJM*], compact RBFs are advantageous because of their limited spatial extent, first and second derivative smoothing of noisy data, and ease of evaluation.

A significant amount of work on RBFs [FN91, Fra82, FH99, Har90, Har71] has been done to reconstruct surfaces by approximating scattered data sets. The multiquadric function [FH99, Har90, Har71] has been used with many applications, and knot selection [MF92] is typically used for the approximation function. This work has been shown to be successful for surface reconstruction. In particular, Carr et al. showed nice results in their paper for surface objects [CBC*01]. In addition, Co et al. [CHH*03] showed a hierarchical representation of volumetric datasets based on a clustering computed by PCA, but they resample their data to a full uniform grid for visualization.

Interactive rendering is a crucial feature for the visualization of volumetric data. The ability to interact with transfer functions and viewpoint orientation provides powerful visual cues that would be difficult to reproduce in off-line volume rendering. Most interactive volume visualization algorithms for uniform grids utilize the texture mapping hardware of general-purpose graphics adapters.

These latter algorithms fundamentally represent the volume data as a 3D texture and resample it by rendering textured planes orthogonal to the viewing direction [CCF94]. The same technique also allows for the rendering of non-polygonally represented shaded isosurfaces [WE98]. The advent of 3D texture mapping, multi-texturing, and programmable graphics pipelines on PC graphics cards allows interactive high-quality volume rendering of these voxel data sets [RSEB*00, EKE, KPHE02]. However, the limited amount of texture memory is a serious constraint for visualizing large data sets. When data size exceeds the limits of physical texture memory, texture paging is necessary, which severely hampers the interactivity of the rendering. Adaptive multi-resolution representations can alleviate this problem [LHJ99, WWH*00]. However, the requirement for artifact-free consistent interpolation leads to a large amount of topological information and a higher rendering and reconstruction overhead. Moreover, even with this multiresolution representation, the sampling of a single plane still requires accessing large amounts of data. As this data may be widely spread across the texture memory, the rasterizer can hardly benefit from texture caching.

Interactive rendering for unstructured volumes is almost entirely based on the Projected Tetrahedra (PT) algorithm [ST91]. This algorithm exploits hardware-accelerated triangle scan conversion by decomposing projected tetrahedra into triangles and rasterizing these triangles with the correct color and opacity computed at the triangle vertices by ray integration. Improvements of the basic PT algorithm include improved rendering quality [SBM94, RKE00] and exploiting today's programmable vertex and fragment units by mapping the tetrahedra decomposition to standard graphics hardware [WKFC02, WKE02], thus freeing CPU resources.

Unstructured grids can provide an adaptive representation of the volume data. However, the rendering performance for unstructured grids is still inferior to that of texture based volume rendering of structured grids. The main bottleneck is processing the tetrahedra in the correct visibility order [MHC90, Wil92].

© The Eurographics Association 2004.

The RBF approach provides a superior uniform solution for the visualization of structured and unstructured volume data, especially for large data sets. Compared to a multiresolution hierarchy of a structured volume, a radial basis function representation of the same data can achieve high compression ratios since no topological information is required. The relatively small number of basis functions required to reconstruct a single fragment leads to local memory access schemes that can benefit from texture caching.

By exploiting radial basis functions that are reconstructed via per-fragment operations during rasterization, we can combine the slice-based rendering approach with a compact volume representation and apply all rendering techniques that are well established for texture based volume rendering.

As our encoding approach uses radial basis functions, splatting [HSMC00] could be a valid alternative to per-fragment reconstruction. However, splatting with footprints of different sizes does not work well with hierarchy decomposition. Since the influence regions of different RBFs will overlap significantly, a global ordering of the RBFs is not possible, and, therefore, a slice based approach has to be taken.

3. Radial Basis Functions

Radial basis functions (RBFs) [SPOK95, TO99, MYR*01] are circularly-symmetric functions centered at a single point. Possible basis functions include thin-plate splines, multiquadrics, and Gaussians. RBFs are widely used in many fields (e.g., image processing and medical applications). Within computer graphics, RBFs are most commonly used for representing surface models and for mesh reduction [SPOK95, TO99, CBC*01, MYR*01, TO02]. RBFs have also been used for surface construction and rendering of large scattered data sets [CBC*01, Gos00]. The main advantages of RBFs include their compact description, ability to interpolate and approximate sparse, non-uniformly spaced data, and analytical gradient calculation.

With radial basis functions, the functional representation $f(x)$ can be expressed as linear combinations of the chosen basis functions as follows [GN01]:

$$f(x) = \sum_{i=1}^{N} w_i \phi_i (\|x - \mu_i\|) \tag{1}$$

N	Number of input
x	d-dimensional input vector
w_i	RBF weight
ϕ_i	Basis function
μ_i	RBF center
$\|x-\mu_i\|$	Vector norm of x to μ_i

In order to interpolate a function with N points, the simple form of an RBF places basis function centers at each of the N points and then solves for the weights of each RBF. For data compression and smoothing, a reduction of the number of basis functions is typically performed given some optimization criteria, thus providing a compact representation of the input data. Some reduction schemes introduce a constant (or linear) error term which has to be compensated for in the reconstruction equation.

4. RBF Encoding of Volume Data

As previously mentioned, there are many basis functions that may be used in RBF encoding. Biharmonic and triharmonic splines are well-suited for surface representation and can provide better results than compactly supported RBFs [CBC*01]. In this work, however, we use truncated Gaussians for the reconstruction. Their limited spatial support, in turn, translates to a small set of functions that must be evaluated at any given spatial location to produce a reconstructed scalar value. This small set of functions can be reconstructed at interactive frame rates.

Although any RBF with non-infinite influence will work with our real-time reconstruction method, we have chosen the truncated Gaussian function as our basis function because the functional value converges to zero exponentially, not polynomially like other basis functions. Moreover, by specifying the widths for each of the truncated Gaussian RBFs, we can make spatially isolated functions that accurately represent local features. With Gaussian basis functions, the RBF functional representation is defined as follows:

$$f(x) = w_0 + \sum_{i=1}^{M} w_i e^{-\frac{\|x-\mu_i\|^2}{2\sigma_i^2}} \tag{2}$$

M	Number of basis functions
w_i	RBF weight
μ_i	RBF center
$\|x-\mu_i\|$	3D-space distance of x to μ_i
σ_i^2	RBF width
w_0	Bias

Therefore, to effectively encode a scalar data set, we need to determine the center location, weight, and width of each basis function.

There are many different methods to choose RBF centers, including random subset selection, clustering algorithm, and mixture models[GN01]. In this work we use PCA analysis[Jol86] to cluster the data points and in each cluster, the RBF center is selected as either the value-weighted cluster average point or the maximum error point as chosen by the user. The RBF width is determined by a hybrid gradient-descent nonlinear optimization technique (Levenberg-Marquardt method) as stated in [PFTV92]. The cost function for optimization uses the mean square error over all data points The individual RBF weight and global bias are computed by minimizing the sum squared error for all data points (e.g., Pseudo-inverse method[Alb72]). Using this method, RBFs are repeatedly added in clusters with the

© The Eurographics Association 2004.

largest errors until the user specified error criteria is satisfied. Encoding errors are calculated as the difference between the original value and the evaluated RBF representation at each input data point.

Once the original volume data is encoded as a weighted sum of RBFs, the computational grid can be discarded. The series of RBFs will reproduce the original scalar field within the accuracy tolerance specified during encoding.

5. Spatial Data Structure

In an effort to reduce the number of basis functions that must be evaluated at a given location, we create an adaptive octree to store the list of relevant basis functions in a given region of space. We use non-uniform spatial subdivision based on basis function center locations. For each cell in the tree, the list of contributing basis functions is calculated by determining if their radius of influence r_i intersects the cell. For the Gaussian basis function, solving Equation (2) yields the following formula for r_i:

$$r_i = \sigma_i \cdot \sqrt{2 \cdot ln\left(\frac{|w_i|}{\varepsilon}\right)} \tag{3}$$

where ε is a user defined error tolerance.

Our subdivision terminates when the number of basis functions per cell is less than a threshold n (maximum RBFs per cell) or when further subdivision does not significantly reduce the number of basis functions for the eight children cells.

If very high accuracy is needed, the user may choose to still render all the basis functions at the cost of interactivity. To account for error while supporting more interactive rendering, we use the following approach. First, the error introduced by only evaluating the n most significant basis functions per fragment is calculated. We then store the error at each cell corner point in the cell data structure and these values are interpolated by the graphics processor during rasterization. From our initial experiments, this linear approximation provides good results. However, with very small values of n, linear artifacts may be introduced.

6. Interactive Reconstruction

For the visualization of the RBF encoded volume data we adopt the texture based volume slicing approach mentioned earlier. This method is well established for the visualization of volume data on regular grids. Slice polygons are computed by intersecting a plane with the bounding box of the desired volume domain and a set of these slices oriented orthogonal to the viewing direction is placed equidistantly within the volume domain, rendered with texture mapped volume data, and finally composited back to front.

Our approach, however, eliminates the need to store the volume data in a three-dimensional texture map. Only the RBF parameters reside in two-dimensional textures. Based on our compact RBF representation, we exploit the programmability of the GPU fragment processor to perform an on-the-fly reconstruction of the RBF encoded volumetric data during the rasterization of each slice as depicted in Figure 2.

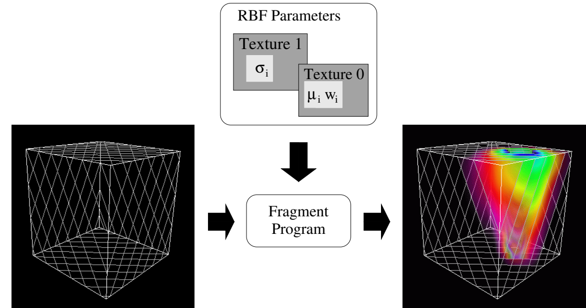

Figure 2: *Our interactive reconstruction is based on a volume slicing approach with a fragment program evaluating, for each rendered fragment, the RBF encoding stored in two-dimensional textures on the fly.*

Since the RBFs are evaluated by the GPU for each rendered fragment, the encoding of the data is hidden from the rendering and, therefore, our approach extends to a variety of visualization algorithms, such as arbitrarily oriented cutting planes, and volume-rendered non-polygonal isosurfaces as in [WE98]. This volume rendering is also achieved by rendering slices, but the reconstructed volume data is mapped to the alpha channel of the fragment color and the OpenGL alpha test is used to simulate the first-hit semantics of a volume ray caster. Further possibilities include the mapping of the reconstructed data onto the surface of a related geometry, e.g., color coded pressure on the body of an airplane.

6.1. High Level Rendering

As mentioned previously, we use a spatial decomposition of the data domain in order to reduce the number of RBF centers that have to be considered for a single fragment. Because each cell of the decomposition can have different sets of centers, separate rendering states per cell are required.

This situation is quite similar to the bricking approach taken in texture-based volume rendering when the size of the data set exceeds the physical texture memory [GHY98]. There, the data set is decomposed into a set of blocks or bricks, and each brick is rendered with a separate three-dimensional texture. The bricks are sliced independently in back-to-front order to minimize state changes.

In our approach, the costs for switching between cells are comparatively small—mostly not even a texture switch. We do, however, have to deal with multipass rendering, because a cell may contain more RBFs than can be handled

© The Eurographics Association 2004.

```
for (all slices)
{
  activeCellList      = createActiveCellList();
  intersectedCellList = copy(activeCellList);

  // Phase I
  while (cells in activeCellList)
  {
    setupRenderingPass();

    for (each cell c in activeCellList)
    {
      if (c contains only ONE chunk of unrendered RBF's)
        removeFromActiveCells(c);
      else
        renderIntersectionPolygon(c);
    }
  }

  // Phase II
  setupFinalRenderingPass();

  for (each cell c in intersectedCellList)
    renderIntersectionPolygon(c);
}
```

Figure 3: *Traversal algorithm for slice-based RBF-Rendering.*

by the fragment processor in one step. Therefore, we utilize a traversal order, which iterates over all slices in the outer loop and for each slice processes the intersected cells. We clip all slices at the boundaries of a given cell and render the resulting polygons (multiple times). Figure 3 shows the corresponding pseudocode.

Multipass rendering is performed in two phases using a set of two hardware accelerated floating point p-buffers: In the first phase the fragment program partially evaluates the RBF sum and writes the intermediate result into one of the buffers. This result is then used as an input for the next rendering pass by binding the p-buffer to a texture map. We need two p-buffers since the GeForce FX does not support simultaneous read and write operations on the same buffer. The final pass (Phase II) directly writes to the graphics context of the program window.

We utilize an active cell list in order to minimize the cell traversal costs. For each slice, the list is initialized with all cells intersected by the the current slice. For each cell we render all but one of the multiple passes during the first phase. The last rendering pass is performed in the second phase, guaranteeing that the intersected slice area for each cell is finally rasterized into the framebuffer. Therefore, we remove the cell from the active cell list as soon as only one additional pass would be required. A second list is maintained to store all cells which must be traversed during the last rendering pass. It is initialized with all intersected cells as well.

If our hierarchy contains only cells that can be rendered in a single pass, we switch to a cell-based traversal in order to minimize the traversal costs. In this case we process all slices for a given cell first, before switching to the next cell. Visibility sorting of the cells is then required to achieve proper semi-transparent volume rendering. We apply a recursive sorting algorithm at each level of our spatial decomposition that sorts the eight children using the distance of their centers from the viewer's position and descends in a depth-first manner based on the level-wise ordering of the children. The sorted cell list can also be used in multipass rendering for faster access to all cells that are intersected by a particular slice.

6.2. Texture Encoding

During rendering, the fragment processor has to be able to access the basis functions that are selected for function value reconstruction inside the current cell. We exploit the high memory bandwidth of the graphics adapter by storing the RBF data at full precision in a set of two floating point texture maps (see Figure 2). The required textures reside within the local memory of the graphics adapter, since the total amount of RBF data is small. Thus, the bottleneck of transferring data from the CPU to the GPU is avoided.

Our first texture is an RGBA map holding the positions μ_i of the RBF centers and the weights w_i of the RBF functions in the RGB and alpha components, respectively. The second map consists of only one color component storing the widths σ_i of the RBF functions. In order to reduce the number of fragment operations required for the reconstruction, we do not store the actual widths, but instead store $(2\sigma_i^2)^{-1}$.

We arrange all the parameters for a single cell consecutively in the texture maps allowing the fragment processor to access the required list of RBFs for each fragment by applying an increasing offset to the texture coordinates, which point to the first RBF of each cell. Texture wrapping is avoided since branching instructions introduce performance penalties in the fragment processing. Figure 4 demonstrates an example of the applied texture packing. Whenever a cell requires more basis functions than can be processed in a single rendering pass, we split the list of RBFs into chunks that can be handled in one pass. The multipass rendering allows for a tight packing of the cells' data, since we only have to guarantee that the RBF parameters needed for one pass are stored consecutively. Even several texture sets can be used if the RBF data exceeds the maximum size of one texture map.

The number of RBFs to be evaluated per cell may vary throughout the spatial decomposition. Therefore, since current graphics hardware does not support dynamic loops in fragment processing and branching instructions significantly degenerate the rendering performance, we provide specialized programs for different numbers of RBFs. However, we avoid extensive program switching and reduce performance penalties by restricting ourselves to a rather small set of different programs. Cells that require an intermediate number of RBFs, therefore, pad their RBF data in the texture maps with zero values up to the next available fragment program size.

© The Eurographics Association 2004.

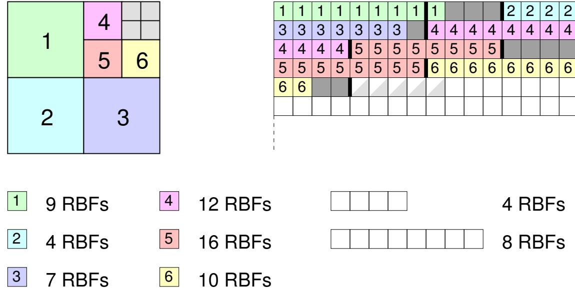

Figure 4: *The RBF data for all cells is tightly packed into a single set of texture maps. In this example, two different fragment programs for 4 and 8 RBF evaluations are available.*

6.3. Per-Fragment Reconstruction

We use a high-level shading language for programming the fragment processor since current graphics hardware, such as the GeForce FX, has the ability to run fragment programs with over 1000 operations in a single pass, and long assembler programs are hard to code and to debug. Our implementation is based on NVidia's Cg [nVi02], which supports both graphics APIs, DirectX and OpenGL, by providing different compiler profiles.

Our visualization system loads pre-compiled fragment programs instead of using Cg's on-the-fly compilation to reduce start-up overhead. Additionally, this approach easily allows for adding support for other basis function types and writing hand optimized assembler code.

Based on the RBF encoding in the texture maps, we apply the fragment program presented in Figure 5 for the single pass RBF reconstruction. We removed the multipass related parts, since we consider the presented version to be more readable and the multipass code is straightforward. The program uses the Gaussian radial basis function introduced in Section 4. We also implemented programs for inverse multiquadric RBFs and for different visualization modes, in particular isosurface rendering. The latter additionally performs lighting calculations based on the data gradient that is analytically evaluated in parallel to the data value as:

$$\nabla f(x) = -\sum_{i=1}^{M} \frac{x-\mu_i}{\sigma_i^2} w_i e^{-\frac{\|x-\mu_i\|^2}{2\sigma_i^2}} \tag{4}$$

The number of RBF functions to be evaluated is encoded as a fixed preprocessor constant, since current graphics hardware does not support dynamic loops in fragment processing.

The program straightforwardly accumulates the RBF functions in a local variable. The iteration over the number of RBFs include the lookup of the RBF center coordinates, the RBF weight and width from the texture maps, the computation of the center's distance to the current fragment position, and the evaluation of the RBF function. Due to performance issues we use the base-two exponential which is compensated with a correction factor multiplied to σ_i-entries in the texture maps. As mentioned previously, we require two lookups per-fragment since the five RBF parameters could not be stored in only one texel. By combining four RBF widths in one RGBA texel and performing the width lookup only every fourth RBFs, we would have been able to reduce the number of lookups to an average of 1.25. However, the more complicated data handling and the computation of the different texture coordinates resulted in very little performance increase.

```
// Maximum number of basis functions for loop unrolling
#define CONST_NUMFUNCS 36

float4 main  ( // Current world coordinates and RBF textures
             float4               inpos     : TEXCOORD0,
             float                error     : TEXCOORD1,
             uniform samplerRECT rbfcenter: TEXTUREUNIT1,
             uniform samplerRECT rbfwidth : TEXTUREUNIT2,
             // Texture  adressing: offset + increment
             uniform float4       texstart : C0,
             uniform float4       texinc    : C1,
             // Bias for RBF reconstruction
             uniform float        bias      : C2,
             // Color table, scale + bias, alpha scale
             uniform sampler1D    map       : TEXTUREUNIT0,
             uniform float4       mapSBA    : C20,
            ) : COLOR
{
  float  val    = 0.0;
  float4 texpos = texstart, output;

  for (float i = 0; i < CONST_NUMFUNCS; i++) {
    // texpos.z counts how many RBFs still have to be evaluated in this cell
    float4 tmp    = texRECT (rbfcenter, texpos.xy);
    float  w_inv  = texRECT (rbfwidth, texpos.xy);
    float3 vec    = tmp.rgb - inpos.xyz;
    float  expval = - dot (vec, vec) * w_inv;
    val    += tmp.a * ex2 (expval);
    texpos += texinc;
  }
  // Add bias and interpolated error
  val += bias + error;
  // Color table lookup after scale + bias
  output.rbga = tex1D (map, (val + mapSBA.r) * mapSBA.g);
  // Transparency correction for volmue slicing
  output.a   *= mapSBA.a;
  return output;
}
```

Figure 5: *The fragment program for reconstructing Gaussian radial basis functions.*

We use the interpolated texture coordinates `inpos` to provide the model space coordinates for each fragment. After evaluating all RBF functions, the constant bias is added. If the spatial data structure includes error values at the corner of each cell, we add the linearly interpolated error that is encoded in the secondary texture coordinates. After an additional scale and bias operation, which allows us to account for the relevant data range, the resulting scalar value is finally mapped to an output color by a 1D texture lookup.

7. Results

We have implemented our system on a Pentium 4 2800MHz processor with an Nvidia GeForce FX 5900 Ultra graphics processor and tested it on a variety of data sets. These data

© The Eurographics Association 2004.

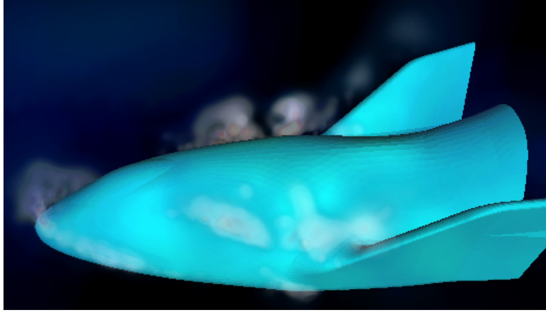

Figure 6: *Volume rendering of the RBF encoded X38 shock data set.*

sets include a computationally simulated X38 configuration, a natural convection simulation, a black oil reservoir simulation, the negative potential iron protein voxel data set, and the bluntfin data set. Unless stated differently we achieved our timings on a viewport of 400×400 pixels using a set of fragment programs for 20, 40, 60, 80, and 100 basis functions. In the following, we discuss the results obtained from encoding each of these data sets using the previously described radial basis functions.

7.1. X38 Crew Return Vehicle

The X38 data set that we used is based on a tetrahedral finite element viscous calculation computed on geometry configured to emulate the X38 Crew Return Vehicle. The geometry and the simulation were computed at the Engineering Research Center at Mississippi State University by the Simulation and Design Center. This data set represents a single time step in the reentry process into the atmosphere. The simulation was computed on an unstructured grid containing 1,943,483 tetrahedra at a 30 degree angle of attack.

We computed the normal Mach number and extracted data values greater than 0.6, as the shock is created due to the transition from sub-sonic speeds (< Mach 1) to super-sonic speeds (> Mach 1). The actual shock volume has a normal Mach number very close to 1.0. We clipped the values in the shock data at 0.6 to reduce the datasize for RBF encoding. This clipped data set was then encoded with 2,932 Gaussian RBFs. The images in Figure 1(a) and Figure 6 show volume renderings of a tight bound on the shock volume, with data between 0.9 and 1.1. Figure 7 shows an interactive cutting plane rendering of the shock values ranging from 0.6 (blue) to 1.7 (white). Figure 8 shows a comparison of encoding the shock data with both 855 and 1,147 RBFs for the more limited range of values 0.7 to 1.7. For the important narrow shock data range shown, 0.8 - 1.02, the overall structure of the shock is the same, with details of the bow shock missing in Figure 8(a). The additional RBFs used in Figure 8(b) are needed to capture the finer structures of the shock. These cutting plane images render at approximately 15 fps, while the volume rendering rate for the shock data is approximately

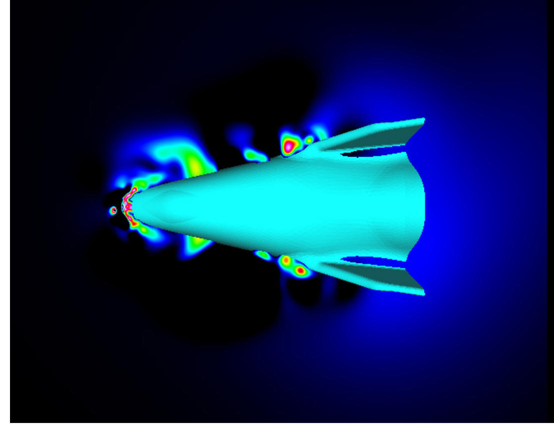

Figure 7: *RBF reconstruction of the X 38 shock data set.*

0.33 fps. Better spatial locality during the encoding of this dataset will increase the performance of the rendering. We have also encoded the density data set from this simulation where the most interesting values are density values less than 0.5. The data set was encoded using only 1,611 RBFs since the density variation doesn't have the sharp discontinuities of the shock data. The average encoding error was just under 2%. A volumetric isosurface rendering of the low density region of the data can be seen in Figure 9.

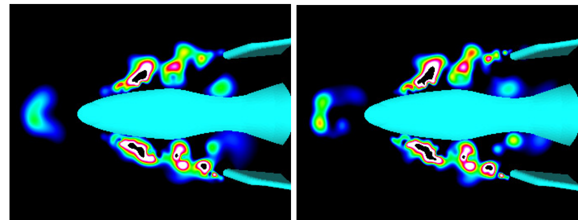

Figure 8: *RBF reconstruction of the X 38 shock data set. (a) 855 RBFs are used for reconstruction. (b) 1,147 RBFs are used for reconstruction.*

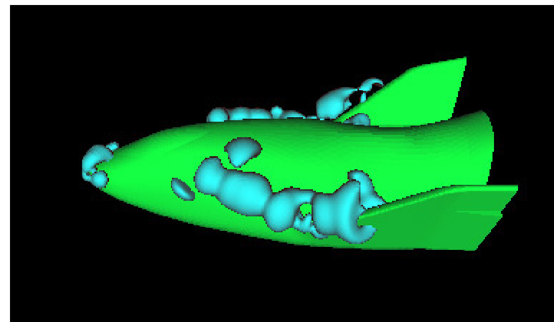

Figure 9: *Volume isosurface rendering of the X38 density data reconstructed with 1,611 RBFs.*

© The Eurographics Association 2004.

7.2. Natural Convection in a Box

Figure 10 shows a semi-transparent volume rendering and an isosurface rendering (isovalue 0.5) of the 80th time step of temperature generated from a natural convection simulation of a non-Newtonian fluid in a cube. The domain is heated from below, cooled from above, and has a fixed linear temperature profile imposed on the sidewalls. The simulation was developed by the Computational Fluid Dynamics Laboratory at The University of Texas at Austin and was run for 6000 time steps on a mesh consisting of 48000 tetrahedral elements. The semi-transparent volume rendering runs at approximately 1.8 fps using 32 slices. In isosurface mode the performance drops to 0.4 fps, since the isosurface fragment program is more expensive, and we could not use our most optimized program version here.

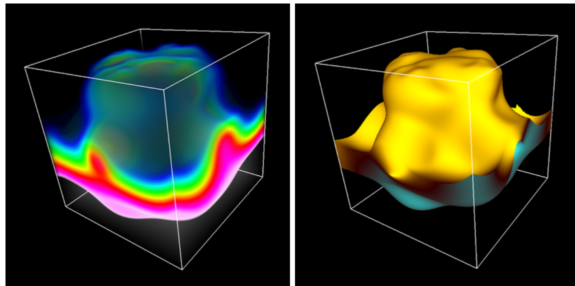

Figure 10: *Volume and isosurface rendering of temperature generated from a natural convection simulation.*

7.3. Black Oil Reservoir Simulation

Figure 1(b) and 11 show volume renderings of the reconstructed oil reservoir data set computed by the Center for Subsurface Modeling at The University of Texas at Austin. The data set is a simulation of a black-oil reservoir model used to predict placement of water injection wells to maximize oil from production wells. The data set has 156,642 tetrahedra containing water pressure values for the injection well. The data set renders at approximately 1.8 fps on a GeForceFX 5900 Ultra graphics adapter using 64 slices.

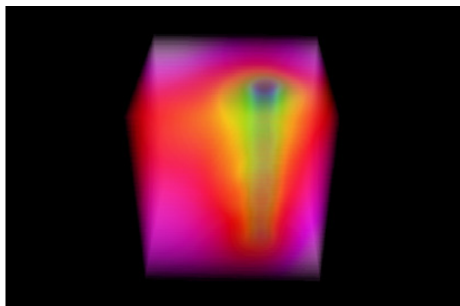

Figure 11: *Volume rendering of water pressure for an injection well. The 156,642 tetradra data set is encoded using 458 RBFs.*

7.4. Negative Potential Iron Protein Data

We also encoded the $32 \times 32 \times 32$ neghip data set from the University of Tübingen which shows the spatial probability distribution of the electrons in a negative potential protein molecule. Figure 12 shows a volumetric rendering of the RBF encoded data set using 812 basis functions on a four level spatial hierarchy consisting of 126 cells with a maximum of 100 basis functions per cell. The data set renders at approximately 2.6 fps on the test system using 32 slices.

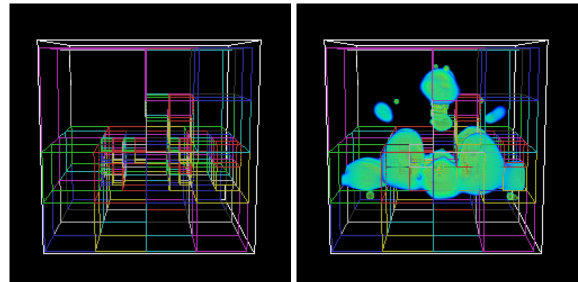

Figure 12: *Semi-transparent volume rendering of the neghip data set using 812 basis functions. The rendering is performed on a spatial decomposition comprised of four subdivision levels.*

7.5. Blunt Fin Data

The bluntfin data set displayed in Figure 13 has been encoded hierarchically with 695 RBFs, with 238 cells and a maximum of 60 RBFs per cell. Again, we used a set of fragment programs with up to 60 basis functions for rendering the data set at interactive rates. Using 64 slices the data set renders at approximately 3.7 fps on a GeForceFX 5900 Ultra graphics adapter.

All of the above Figures were generated with our hardware accelerated reconstruction program. For slice plane rendering, we achieve performance of 7 to 75 fps on a 400^2 viewport due to a comparatively high amount of RBFs per cell. Exploiting the half-float register type of Cg, led to a performance improvement between 30% and 300% depending on the rendering mode. However, we could not apply this program to all tested data sets, due to the limited 16 bit precision.

The 1024 fragment program instruction limit of the GeForce FX allowed us to evaluate 59 to 126 RBFs per pass, depending on data encoding and the rendering mode. When multipass rendering is needed, we experience increased performance for larger fragment programs, which need to write intermediate results to the frame buffer less often. For our adaptive octree encoding, the addition of cells in the hierarchy causes overhead for rendering. However, we have found that the reduction in wasted Gaussian RBF evaluation outweighs this overhead and significantly increases performance.

© The Eurographics Association 2004.

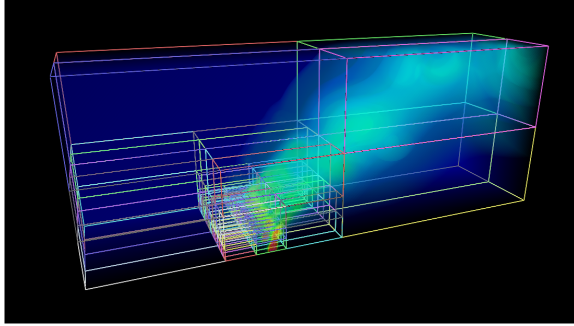

Figure 13: *Volume rendering of the bluntfin data set with its subdivision hierarchy.*

8. Conclusion and Future Direction

We have demonstrated a novel, unified approach for the interactive reconstruction and visualization of arbitrary 3D scalar fields, including voxel data and unstructured data. By combining a compact functional encoding, hardware-accelerated functional reconstruction, and domain knowledge of data importance, we have developed a system that avoids the traditional data transfer bottleneck of hardware accelerated rendering of large scalar fields. This approach can take advantage of the rapid performance increase of PC class graphics hardware. The flexibility and extensibility of functional encoding and interactive reconstruction allows the interactive exploration of very large data sets from a variety of sources. We can visualize data sets with a few million tetrahedra at interactive rates using slicing planes and preview-quality slice-based volume rendering. Our future work will include further optimization of our interactive reconstruction in terms of performance, e.g., removing bottlenecks of the current implementation by balancing the reconstruction efforts between vertex and fragment processing. Image quality could also be improved by incorporating pre-integrated volume rendering. We will also improve our RBF encoding techniques for volumetric scalar fields to provide more limited spatial support, significantly reducing the the number of RBFs required per fragment and, therefore, speeding up the performance.

Acknowledgements

We wish to thank the reviewers for their suggestions and helpful comments. This work has been supported by the US National Science Foundation under grants NSF ACI-0081581 and NSF ACI-0121288.

References

[Alb72] ALBERT A.: *Regression and the Moore-Penrose Pseudoinverse*. Academic Press, 1972.

[BDHJ00] BERTRAM M., DUCHAINEAU M., HAMANN B., JOY K. I.: Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. In *IEEE Visualization 2000* (October 2000), pp. 389–396.

[BJM*] BARNSLEY M., JACQUIN A., MALASSENET F., REUTER L., SLOAN A.: Harnessing chaos for image synthesis. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, pp. 131–140.

[CBC*01] CARR J., BEATSON R., CHERRIE J., MITCHELL T., FRIGHT W., MCCALLUM B., EVANS T.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001* (August 2001), Computer Graphics Proceedings, pp. 67–76.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. *1994 Symposium on Volume Visualization* (October 1994), 91–98.

[CHH*03] CO C. S., HECKEL B., HAGEN H., HAMANN B., JOY K. I.: Hierarchical clustering for unstructured volumetric scalar fields. In *Proceedings of IEEE Visualization 2003* (October 2003).

[EKE] ENGEL K., KRAUS M., ERTL T.: High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware '01*, pp. 9–16.

[FH99] FRANKE R., HAGEN H.: Least squares surface approximation using multiquadrics and parametric domain distortion. *Computer Aided Geometric Design 16*, 3 (March 1999), 177–196.

[FN91] FRANKE R., NIELSON G. M.: Scattered data interpolation and applications: A tutorial and survey. In *Geometric Modelling, Methods and Applications*, Hagen H., Roller D., (Eds.). Springer Verlag, 1991, pp. 131–160.

[Fra82] FRANKE R.: Scattered data interpolation: Tests of some method. *Mathematics of Computation 38*, 157 (1982), 181–200.

[GHY98] GRZESZCZUK R., HENN C., YAGEL R.: Advanced geometric techniques for ray casting volumes. *ACM SIGGRAPH '98* (July 1998). Course 4 Notes.

[GN01] GHOSH J., NAG A.: An overview of radial basis function networks. In *Radial Basis Function Networks 2*, Howlett R. J., Jain L. C., (Eds.). Physica-Verlag, 2001, pp. 1–36.

[Gos00] GOSHTASBY A. A.: Grouping and parameterizing irregularly spaced points for curve fitting. *ACM Transactions on Graphics (TOG) 19*, 3 (2000), 185–203.

[Har71] HARDY R. L.: Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research 76*, 8 (1971), 1905–1915.

[Har90] HARDY R. L.: Theory and applications of the

© The Eurographics Association 2004.

multiquadric-biharmonic method 20 years of discovery 1968-1988. *Computers and Mathematics with Applications 19*, 8-9 (1990), 163–208.

[HSMC00] HUANG J., SHAREEF N., MUELLER K., CRAWFIS R.: Fastsplats: Optimized splatting on rectilinear grids. In *IEEE Visualization 2000* (October 2000), pp. 219–226.

[Jol86] JOLLIFFE I. T.: *Principal Componenet Analysis*. Springer-Verlag, 1986.

[KPHE02] KNISS J., PREMOZE S., HANSEN C., EBERT D.: Interactive translucent volume rendering and procedural modeling. In *Proceedings of the conference on Visualization 2002* (2002), IEEE Press, pp. 109–112.

[LHJ99] LAMAR E., HAMANN B., JOY K.: Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization 1999* (1999), IEEE CS Press.

[MF92] MCMATHON J. R., FRANKE R.: Knot selection for least squares thin plate splines. *SIAM Journal on Scientific and Statistical Computing 13*, 2 (March 1992), 484–498.

[MHC90] MAX N., HANRAHAN P., CRAWFIS R.: Area and volume coherence for efficient visualization of 3d scalar functions. *ACM Computer Graphics (Proceedings of San Diego Workshop on Volume Visualization 1990) 24*, 5 (1990), 27–33.

[MYR*01] MORSE B. S., YOO T. S., RHEINGANS P., CHEN D. T., SUBRAMANIAN K. R.: Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings Shape Modeling International* (2001).

[NS01] NGUYEN K. G., SAUPE D.: Rapid high quality compression of volume data for visualization. *Computer Graphics Forum 20*, 3 (2001), 49–56.

[nVi02] NVIDIA: Cg language specification, 2002. Cg Language Specification, available at http://developer.nvidia.com/cg.

[PFTV92] PRESS W. H., FLANNERY B. P., TEUKOLSKY S. A., VETTERLING W. T.: *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, 1992.

[RKE00] ROETTGER S., KRAUS M., ERTL T.: Hardware-Accelerated Volume and Isosurface Rendering Based On Cell-Projection. In *Proceedings of IEEE Visualization 2000* (2000), IEEE, pp. 109–116.

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00* (2000), Addison-Wesley Publishing Company, Inc., pp. 109–118,147.

[SBM94] STEIN C. M., BECKER B. G., MAX N. L.: Sorting and hardware assisted rendering for volume visualization. In *Proceedings 1994 Symposium on Volume Visualization* (1994), Kaufman A., Krueger W., (Eds.), ACM Press, pp. 83–89.

[SPOK95] SAVCHENKO V. V., PASKO A. A., OKUNEV O. G., KUNII T. L.: Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14*, 4 (1995), 181–188.

[ST91] SHIRLEY P., TUCHMAN A.: A polygonal approximation to direct volume rendering. In *Proceedings of the San Diego Workshop on Volume Visualization* (November 1991), pp. 63–70.

[TO99] TURK G., O'BRIEN J.: Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 99* (August 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 335–342.

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics (TOG) 21*, 4 (2002), 855–873.

[WE98] WESTERMANN R., ERTL T.: Efficiently Using Graphics Hardware in Volume Rendering Applications. *Computer Graphics (SIGGRAPH '98) 32*, 4 (1998), 169–179.

[Wil92] WILLIAMS P. L.: Visibility ordering meshed polyhedra. *ACM Transactions on Graphics 11*, 2 (1992), 103–126.

[WKE02] WEILER M., KRAUS M., ERTL T.: Hardware-Based View-Independent Cell Projection. In *Procceedings of IEEE Symposium on Volume Visualization 2002* (2002), IEEE, pp. 13–22.

[WKFC02] WYLIE B., KENNETH M., FISK L. A., CROSSNO P.: Tetrahedral Projection using Vertex Shaders. In *Procceedings of IEEE Symposium on Volume Visualization 2002* (2002), IEEE, pp. 7–12.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMAN K., ERTL T.: Level-Of-Detail Volume Rendering via 3D Textures. In *Volume Visualization and Graphics Sympsium 2000* (2000), IEEE, pp. 7–13.

© The Eurographics Association 2004.