Combined Volume Registration and Visualization

Arlie G. Capps, Robert J. Zawadzki, John S. Werner and Bernd Hamann

Abstract We describe a method for combining and visualizing a set of overlapping volumetric data sets with high resolution but limited spatial extent. Our system combines the calculation of a registration metric with ray casting for direct volume rendering on the graphics processing unit (GPU). We use the simulated annealing algorithm to find a registration close to optimal and allow the user to closely monitor the optimization progress. The combined calculation reduces memory traffic, increases rendering frame rate, and makes possible interactive-speed, user-supervised, semi-automatic combination of many component volumetric data sets.

1 Introduction

Volumetric imaging modalities have become centrally important in biological and medical applications. These modalities can produce a volumetric data set ("volume image") composed of samples or voxels commonly arranged in a 3D Cartesian grid

A.G. Capps
Physical and Life Sciences, Lawrence Livermore National Laboratory, Livermore, CA, 94550 USA
Vision Science and Advanced Retinal Imaging Laboratory (VSRI), Department of Ophthalmology and Vision Science, University of California, Davis, Sacramento, CA 95817 USA
Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, Davis, CA 95616 USA
e-mail: capps2@llnl.gov

R.J. Zawadzki, J.S. Werner
Vision Science and Advanced Retinal Imaging Laboratory (VSRI), Department of Ophthalmology and Vision Science, University of California, Davis, Sacramento, CA, 95817 USA
e-mail: rjzawadzki@ucdavis.edu; werner@ucdavis.edu

B. Hamann
Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, Davis, CA 95616 USA
e-mail: hamann@cs.ucdavis.edu

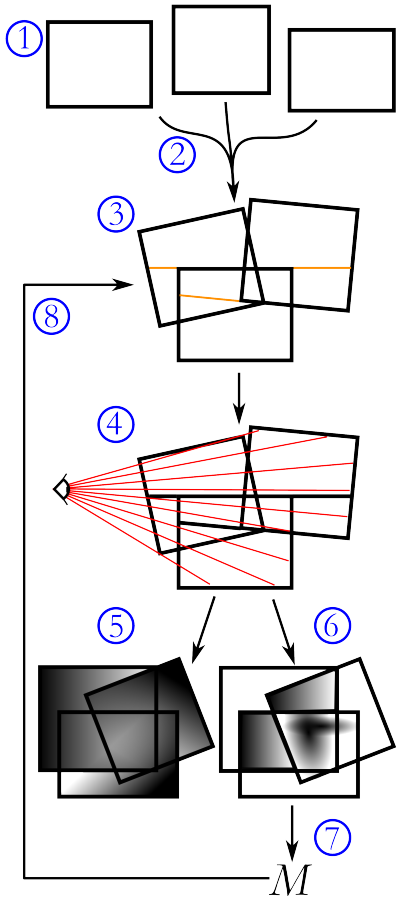

Fig. 1 Overview of the algorithm. (1) Input volumes. (2) User adjusts position. (3) Build a binary space partition (BSP) tree, cutting volumes into non-overlapping convex polyhedra. (4) Perform ray casting. (5) Each ray produces a color value for its pixel. The resulting image is displayed on the screen. (6) Each ray that pierces the intersection of multiple input volumes also produces an intermediate metric result. The resulting metric image is stored in a texture. (7) The intermediate metric results are reduced to a single value M. (8) An optimization step adjusts volume positions. Note that the image texture and the metric intermediate values shown at (5) and (6) represent the view from the eye point, a quarter turn from the representation at (4).

of resolution $I \times J \times K$. Advances in technology and engineering have resulted in rapidly increasing image resolution, data acquisition speed, and data set size and dimensionality. However, with some clinical *in vivo* imaging techniques such as OCT the coverage provided by individual high resolution volume images is smaller than the extent of the imaged structures. Multiple volume images that partially overlap must be captured, registered and combined ("stitched") to allow a single larger region to be analyzed as a single data set. Gaps or holes, areas of the larger region not covered by any volume image, must be easily identifiable as such by the user. Overlaps, areas covered by more than one volume image, must contain enough information for the stitching operation to unambiguously determine the volume images' correct relative position and are combined through an averaging operation to reduce noise and improve contrast.

We have developed methods and a prototype system for stitching multiple high-resolution volume data sets which combines on-screen visualization with registration metric computation. After the user places individual volume images into rough

alignment, our system automatically refines the registration of the component images (see Fig. 1) using rigid-body transformations. Our system also supports combination of scanned data sets in which correction for motion artifacts has changed a deformed, regular image into a correct image with sampling irregularities and gaps. Successive motion-corrected scans of the same region contain different motion and hence different sampling gaps, which are filled in when the scans are properly registered and combined.

The procedures for visualizing multiple overlapping data sets and calculating a metric for registration are closely linked. Both tasks require sampling throughout the regions of image space where multiple volumes overlap followed by combination of the sample values to a summary, either to an image of the combined data or to a metric for goodness-of-fit. Since both tasks require intensive and wide-ranging data sampling, an efficient data access pattern directly benefits the speed of the system. We combine computation of the registration metric with the volume ray casting algorithm running on a workstation GPU.

We developed our system for use with non-invasive *in vivo* cellular resolution human retinal imaging modalities. Specifically, we applied our system to two varieties of optical coherence tomography, which acquires a series of consecutive 2D cross-sectional tomograms that when stacked define a 3D image. The first system [12] scans an area of 3mm$\times$3mm on the retina in 4 seconds to produce an image of 375×360 steps with an axial resolution of 10μm. The second system [14, 15] uses adaptive optics to achieve much finer resolution and scans a retinal patch of 300μm$\times$250μm in 512×100 steps over about 4 seconds, with an axial resolution of 3μm. This fine resolution provides the potential for morphological analysis of microscopic structure in the human eye. The value of this analysis, already great, increases with the coverage of the image and provides a powerful motivation for volume image stitching. However, without some kind of correction, involuntary eye motion that is a significant nuisance in images from the first system becomes prohibitive to stitching ultra-high-resolution data sets acquired using adaptive optics with the second system. Our system should also be applicable to images from other volumetric modalities such as computed tomography (CT), magnetic resonance imaging (MRI), and confocal scanning optical microscopy.

In Section 2, we discuss work related to visualization and registration of volume data sets. We discuss volume partitioning, ray casting to produce images and calculate the registration metric, and optimization using simulated annealing in Section 3. In Section 4 we present examples demonstrating our method and the results of its application to OCT data sets, and we conclude with Section 5.

The key contributions made in this paper are (1) an algorithm combining visualization and computation of a registration metric for overlapping volumes and (2) a system using that algorithm to assist users in stitching multiple retinal OCT volumes to extend high-resolution coverage.

2 Related work

Early work on multiple volume rendering includes that of Jacq and Roux [5], whose method casts rays through combined volumes and accumulated a derived value (the minimum, maximum, or average). Cai and Sakas [2] addressed the use of more general functions for data intermixing and established a useful taxonomy for the stage in the pipeline where data from the two volumes are combined. *Image*-level combines a 2D rendering of each individual volume, *accumulation*-level fusion combines colors from individual volumes' transfer functions during ray casting, and *illumination*-level fusion combines raw data from each volume at each step into one value which is transformed to color. The choice of data fusion level has consequences for the design of the rendering system, requiring different levels of flexibility. Recent work on multi-volume rendering has focused on facilitating arbitrary alignment and increasing the number of overlapping volumes supported. Shader programs are kept manageable by splitting up rays into sections that do not cross volume boundaries. The approach of Rößler et al. [13] treats overlapping volumes as a scene graph and auto-generates the shader programs necessary to render homogeneous regions of space. Kainz et al. [6] introduced a system that transforms the boundary polygons of each input volume to screen space and sorts the resulting fragments in depth to produce homogeneous ray segments. These techniques, while powerful, require expensive depth-sorting techniques. To avoid this cost, solutions suggested by Lindholm et al. [9] and Lux and Fröhlich [10] use a binary space partitioning (BSP) tree to produce homogeneous volume fragments which are rendered in depth order provided by traversal of the BSP tree. We base our system on the approach of Lindholm et al., which is simpler in its treatment of large volumes. The systems discussed so far and several other recent works [4, 8] perform multi-volume visualization, requiring registered volumes as input.

Many capable systems have been introduced that accomplish automatic or semi-automatic registration of volume data. Bria et al. [1] described a system for stitching 3D confocal ultramicroscopy (CU) data sets, with no function for visualization. Unlike OCT, the CU data sets are sparse; like OCT, the initial position of each volume are approximately known and are used as starting points for registration. Dalvi et al. [3] introduced a multi-step process starting with recorded sensor position, performing feature extraction using a wavelet transform and finishing with intensity-based volume registration. This system performs non-interactive stitching of ultrasound data sets, with no visualization component. Ultrasound, like retinal OCT, is a coherent detection process and ultrasound data sets contain pervasive speckle noise. However, the examples in [3] show features that are much larger and less complicated than the range of feature scales present in retinal OCT images. Specific to OCT data sets, Zawadzki et al. [16, 15] presented a system that allows the user stitch multiple high-resolution AO-OCT data sets using axis-aligned translation in whole-voxel increments. Image combination is accomplished by choosing the maximum intensity of overlapping voxels.

3 Methods

Our system is summarized in Fig. 1. The user supplies several input volumes (1) and adjusts their positions (2). The overlapping input volumes are divided using a BSP tree (3) on the CPU, using the boundaries of the input volumes as partition planes. Interior nodes of the BSP tree represent cut planes that divide the overlapping input volumes; leaf nodes represent the subvolumes or BSP "cells" (4) defined by the cut planes. We produce a volume rendering by casting rays through the leaf node cells in order of increasing distance from the eye point. In addition to calculating an image pixel, each ray also calculates a partial metric value. Image pixels are displayed on the screen (5), and intermediate metric results (6) are stored in a graphics texture. The partial metric values are reduced to a final metric value M (7); an optimizer process uses the final metric to adjust the input volume positions (8). The optimizer repeatedly runs steps (3) through (8), using the simulated annealing algorithm [7] to choose successive relative positions.

BSP tree construction (3) and ray casting through the resulting volume fragments (4) to produce a display image (5) are adopted from the approach of Lindholm et al. [9] In our system, the ray casting algorithm produces not only an image for display but also calculates a registration metric. We also integrated the simulated annealing algorithm as the optimizer for this application.

The oriented plane in 3D space is a concept we use throughout our system. The plane pl with equation $ax+by+cz=d$ divides 3D space into its positive side, the open half-space pl^+ with equation $ax+by+cz>d$, and its negative side, pl^-, with equation $ax+by+cz<d$. The vector $\mathbf{n}=(a,b,c)$ is normal to the plane and points into pl^+.

3.1 Binary space partition

After the user selects and places input volumes, our system partitions the region occupied by the input volumes into a BSP tree (Fig. 1, step (3)). Algorithm 1 accomplishes the partition. The input to the algorithm is the list of the input volumes, which must all be convex polyhedra. The algorithm uses the oriented planes of the polyhedral faces to recursively divide the polyhedra until all the face planes have been *used*. Hence, all face planes of all input polyhedra are initially marked *unused*. The output is a binary tree whose nodes have the following fields:

- *pl*, the oriented cut plane (set on interior nodes, unset on leaf nodes),
- *polys*, a list of polyhedra (empty on interior nodes, populated on leaf nodes),
- *left*, the left child node, representing pl^+, the positive side of the cut plane
- *right*, the right child node, representing pl^-, the negative side of the cut plane.

Fig. 2 illustrates the first iterations of applying Algorithm 1 to a 2D scene. Although here we show the application in two dimensions, BSP tree generation is generalizable to any dimensionality, which for our system is 3D. Implementers of

Algorithm 1 Generate BSP tree

Input: Convex polyhedra $v_1 \ldots v_n$.
Output: BSP tree rooted at node r.

```
 1: r.vols ← input volumes
 2: Node n ← r
 3: Recursive section on n:
 4: n.pl ← unused face plane from n.vols
 5: Mark n.pl as used
 6: for all p within p.polys do
 7:     if p lies entirely in pl+ then
 8:         Move p to n.left.polys
 9:     else if p lies entirely in pl− then
10:         Move p to n.right.polys
11:     else
12:         Remove p from n.polys
13:         Cut p with n.pl into p+ and p−
14:         Add p+ to n.left.polys
15:         Add p− to n.right.polys
16:     end if
17: end for
18: if n.left.polys contains anything then
19:     Recur on n.left
20: end if
21: if n.right.polys contains anything then
22:     Recur on n.right
23: end if
24: return Node r, root of a new BSP tree
```

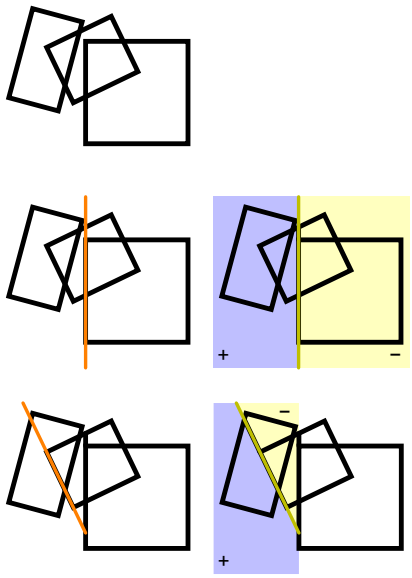

Fig. 2 First iterations of Algorithm 1 applied to a 2D scene. Top: initial state. Middle: first cut and resulting partition. Bottom: second cut and resulting partition.

the BSP tree algorithm must be careful to avoid errors in the geometric operations at lines 7, 9, and 13, which can result from floating-point inaccuracies.

The binary tree resulting from Algorithm 1 has properties useful for speedy ray casting. Each leaf node l contains a list of one or more polyhedra p. All p in the same leaf node have the same boundaries, and each p refers to one of the original input volumes. This means that we iterate over the list of p to get the list of input volumes that l intersects, and also that ray casting only needs to refer to the first p in each leaf node to determine ray entry and exit points. Since all leaf nodes have been constructed using cut planes chosen from the face planes of the input volumes, the interior of no p intersects any input volume boundary. This method of construction also assures that any ray cast through p must start precisely as it enters and stop precisely as it exits, and will never re-enter p. Finally, even with arbitrary viewpoints the Z-ordering of leaf volumes can be obtained without rebuilding the tree.

3.2 Drawing the BSP tree

After constructing the BSP tree, the system executes the ray casting algorithm on the volume cells in order of increasing distance from the eye point i (Fig. 1, step (4)). To determine the depth ordering, the program traverses the BSP tree starting from its root. At each internal node n, if i lies on the positive side of cut plane $n.pl$, the program traverses first the left then the right child node. Otherwise it traverses right then left child. The side of each cut plane that contains i is always visited first, so that leaf nodes (which contain the cells) are visited in increasing depth order.

The program performs ray casting as it visits each leaf node l. This is done in several passes, using code on the CPU and several GPU shaders, as described below. Textures are used to pass intermediate results between rendering steps.

1. Sort faces of l's first polyhedron into two lists: *fronts*, with normals pointing toward i, and *backs*, with normals pointing away from i.
2. Draw each face in *backs* using shader **RecordDepth**, which returns the fragment's distance from i. We store this in a buffer d. We render computed distance to a buffer rather than use the depth buffer commonly provided by rendering toolkits because we need the Euclidean distance from i to the fragment lying on the back-face, not the Z-buffer distance resulting from the perspective transform.
3. Draw each face in *fronts* using shader **RayCast** to perform the ray casting algorithm:
   a. Record the fragment world coordinates as ray start point s.
   b. Stop point t is calculated from the value in buffer d, the distance from i to the ray exit point from l.
   c. For each input volume $v_1 \ldots v_k$ that l overlaps, calculate texture-coordinate step vectors.
   d. Step through all input volumes in parallel from s to t. At each step p,
      i. Accumulate color from the average of $v_1[p] \ldots v_k[p]$
      ii. Accumulate metric from the sum of the squares of all pairwise differences
   e. On ray termination, output three values: the color accumulated along each ray, stored in the buffer lc, the ray's partial metric value, stored in lm, and the number of sample steps p, stored in lp.
4. Draw each face in *fronts* using shader **Finalize** to accumulate the local color, metric, and sample count buffers into the overall buffers C, $Mbuf$, and P.

The heart of the rendering process is line 3d. Shader **RayCast** runs in parallel for each pixel of the final image that portrays the polyhedron l, stepping through l along that pixel's ray from its entry point s to exit point t. Here we explain line 3(d)i in more detail. At each step p, the shader finds the average of all k input volumes overlapping l, then looks up the corresponding color (with r, g, b and a components) in the transfer function table tf:

$$raw = \frac{1}{k}\sum_{i=1}^{k} v_i[p]$$
$$t.rgba = tf[raw]$$

The shader accumulates the color into the pixel color lc using the ray casting formula:

$$lc.rgb = lc.rgb + lc.a \cdot t.rgb$$
$$lc.a = lc.a \cdot t.a$$

To compute the metric over volume l, we compute the sum of the square of image differences over all sample points p within P, the set of ray casting sites lying within l (here $v_n[p]$ means the value located at world space point p in volume n):

$$M = \frac{1}{|P|}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}(v_i[p] - v_j[p])^2. \quad (1)$$

When a ray traverses a volume l that intersects more than one input volume, at each ray cast step p the shader **RayCast** calculates the contribution at p to the registration metric (line 3(d)ii) and accumulates the partial metric value along the ray. The subtotal for all p in the ray are stored in that ray's pixel in metric texture lm.

After completing **RayCast**, shader **Finalize** accumulates color and metric values into the final buffers (line 4):

$$C.rgb = C.rgb + C.a \cdot lc.rgb$$
$$C.a = C.a \cdot lc.a$$
$$Mbuf = Mbuf + lm$$
$$P = P + lp$$

After visiting all leaf nodes in order of increasing distance from i, the program displays color buffer C on screen (Fig. 1, (5)). Then the system divides the sum of the partial metrics stored in *Mbuf* (Fig. 1, (6)) by the sum of the ray sample counts stored in P to produce a final registration metric value M (Fig. 1, (7)).

3.3 Optimization

The final metric value (Equation 1) has a minimum when pixels from one volume overlap pixels in other volumes having equal value. In other words, the metric has a global minimum when volumes are perfectly registered. The position of one volume relative to another can be expressed by six values, three orthogonal translations and three orthogonal rotations. Given k input volumes, we use the $6(k-1)$-tuple θ to represent the rigid transformations of volumes $2 \ldots k$ with respect to volume 1.

The metric function can be written as $M = f(\theta)$ to emphasize that the value of M depends on the rigid transformation (rotation and translation) of the input volumes. In Equation 1, θ was implicit in the statement that overlapping volumes are sampled at points in world space.

3.3.1 Simulated Annealing

To search for the optimum arrangement of input volumes, we implement an iterative optimization routine. The optimizer calculates the metric M for a series of θ, choosing successive θ to converge to a global minimum for M. Optimizers exist which choose new θ based on predicted function behavior [11]. *Newton* and *gradient descent* optimization methods use the function value, first and/or second derivative to predict $f(\theta_{i+1})$ from θ_i based on the assumption that the derivatives exist and the function can be predicted over the interval from θ_i to θ_{i+1}. However, the input volume images are full of high-frequency signal as well as speckle noise, making the metric function unpredictable even if the derivatives were available. The optimizer for the metric function must not require derivative information, and should not rely on function predictions to choose successive values for θ.

To satisfy these requirements, we implemented the simulated annealing (SA) optimizer process which executes a random walk in the parameter space, successively choosing values for θ_{i+1} that are "close" to θ_i. The step is "accepted" when it results in an improvement, and the random walk continues from θ_{i+1}. The step is "rejected" with gradually increasing probability when it results in a worse metric, and the random walk continues from θ_i. SA is well-suited to a noisy registration metric which must be calculated for each new value of θ: the possibility to accept suboptimal steps lets the algorithm escape from local minima, and in contrast to some other methods SA does not need an estimate of the function first or second derivative to choose θ_{i+1}. SA is modeled on the physical process of annealing, where a piece of hot metal is cooled slowly. When hot, the chance of random internal structure change is high. As the temperature drops, the chance of a random change in microstructure drops as well. When the temperature drops too quickly, regions of internal stress and irregular structure remain because the metal is too cold to allow the shifts that would relieve the weak spot. But in cases where the temperature drops slowly enough, the internal structure of the metal becomes stronger because regions of stress have a chance to relax in one of the random changes that occurs as part of the cooling process.

3.3.2 Details of Simulated Annealing

The SA algorithm tracks state with several variables: θ_i is the current rigid transformation of all input volumes, M_i is the metric calculated for θ_i using Equation 1, and temperature T controls the likelihood of accepting a step with a worse metric than M_i. SA uses several parameters: T_0, the initial value of T; dT, the factor by which T

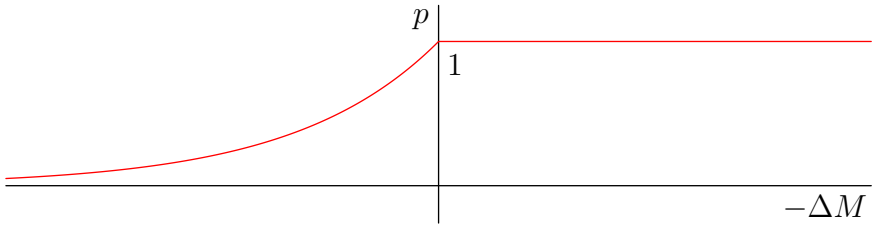

Fig. 3 Graph of Equation 2. The probability of accepting a random step is based on the change in metric.

periodically diminishes; and *freeze*, the value of T which determines the algorithm stopping point. SA has five steps:

1. $\theta_{i+1} = \textbf{randstep}(\theta_i, T)$
2. $M_{i+1} = \textbf{eval}(\theta_{i+1})$
3. If $\textbf{accept}(T, M_i, M_{i+1})$, then let $i = i+1$
4. Let $T = T \cdot dT$
5. If $T >$ *freeze*, go to line 1.

The **randstep** function chooses θ_{i+1} to be "not far" from θ. This keeps the SA random walk from jumping far away from the user's initial placement. The function randomly chooses to translate a volume transversely or axially or to rotate the volume around one of its axes, then randomly chooses an amount by which to move the volume. We rely on the user to place the input volumes close to their desired final registration, within the diameter of the dominant features in the image, and we want the optimizer to refine the user's placement, not to discard it. Thus, we constrain the translation steps to be on the order of ten pixels and rotation steps to be less than 2° at the beginning of the annealing process. We multiply the random translation or rotation by $(T_0 + T)/2T_0$ to decrease the random step along with T.

The **eval** function renders the superimposed input volumes as described in Subsections 3.1 and 3.2 to find the metric M_{i+1} that results from θ_{i+1}.

The **accept** function implements the annealing behavior. It accepts all random steps that give an improved metric (where $M_i > M_{i+1}$), and also probabilistically allows random steps that don't give an improvement (where $M_i \leq M_{i+1}$). The probability of accepting a step from θ_i to θ_{i+1} is:

$$p(\text{step } \theta_i \rightarrow \theta_{i+1}) = \min\left(1, \exp\left(-\frac{M_{i+1} - M_i}{T}\right)\right). \tag{2}$$

Fig. 3 shows a graph of Equation 2. When $M_i > M_{i+1}$ (to the right of the origin), the proposed step will improve the registration so **accept** always returns true. The further to the left of origin, the worse the effect on registration, so the lower the probability of accepting the step. As T decreases, the exponential curve becomes sharper and the probability of accepting a bad step decreases. Near freezing, the curve approaches a step from 0 to 1 at the origin, making the behavior of SA approximate a gradient-descent random walk.

3.3.3 Parameters and Starting Conditions

Successful optimization with SA requires good starting parameters. Together, the parameters T_0, dT and *freeze* determine the *annealing schedule*. The temperature T is closely linked to the random walk. When T_0 is too high, the algorithm will make many random steps with a very low penalty for a bad step and will tend to wander away from the user's specified initial placement, failing to find a registration. So, prior to starting SA proper, we determine a value for T_0 by perturbing the position of each input volume in turn for several tens of steps by random translation of 10 pixels or less and rotation by 2° or less. For the set of all bad steps ($M_i < M_{i+1}$) we find the average $\Delta M = M_{i+1} - M_i$ and take this to be the expected value. We then start SA with $T_0 = E(\Delta M)/\ln 2$. This way, the expected probability of accepting a step that worsens (increases) the metric is initially 0.5, giving the algorithm a chance to settle down around the correct registration as T decreases.

For some initial volume arrangements, SA will fail to settle down around the true registration. We have observed this to happen when the initial placement was not close to the true registration. Since SA is a stochastic process, we cannot guarantee convergence. However, we have observed that initial registration of OCT data sets cut into overlapping subsets and given an initial registration within 10 pixels of displacement tend to succeed. Rotation is much more sensitive; initial misrotation of 3° or more tends to cause SA to fail, while correct rotation of less than 1° tends to lead to success.

To allow the use of an easy-to-work-with value of $dT = 0.95$, we repeat line 1, 2 and 3 several (hundred) times for each input volume in turn before line 4, rather than decreasing T by an infinitesimal amount each time through the loop. Finally, we set $freeze = T \cdot 10^{-5}$. This results in an annealing schedule which in our experience results in successful registration after an acceptable computation time.

Finally, we save the lowest M_i and its θ_i, and apply that arrangement to the input volumes at the end of the SA process.

3.4 Effect of View Point

The ray casting algorithm can be implemented with either perspective or orthogonal projection of the scene to a viewing plane. Orthogonal projection casts parallel rays normal to the viewing plane through the volume data sets; perspective projection casts rays that diverge from the eye point, pass through the viewing plane and volume data. We implemented perspective projection in this project. Since the rays diverge, ray casting sample points p used for metric calculation are dense close to the eye point and become sparser in regions farther away. Thus, a change in eye point will move the region where p is most densely distributed, likely resulting in a different metric value even with no change in input volume position. This means that only metric values computed from the same eye point are comparable. Practically speaking, the simulated annealing registration must restart when a user moves

the global point of view. We have found that repeated experiments with the same initial placement of the input volumes but differing eye points converge to the same correct final registration.

Because our system calculates the registration metric by ray casting through volume overlap regions, we require these regions to be within view, or they will not contribute to the metric. Likewise, "island" volumes that do not overlap others cannot affect the metric, and must by registered by means other than our system.

The common technique of early ray termination cannot be applied to our combined metric and visualization method. Even though a ray may accumulate enough opacity that its color will not change over further sample steps, metric calculation must still be done at those sample steps; so all rays must run to volume exit. However, our algorithm skips all empty space, since it casts rays through BSP leaf nodes, which by construction always intersect at least one input volume.

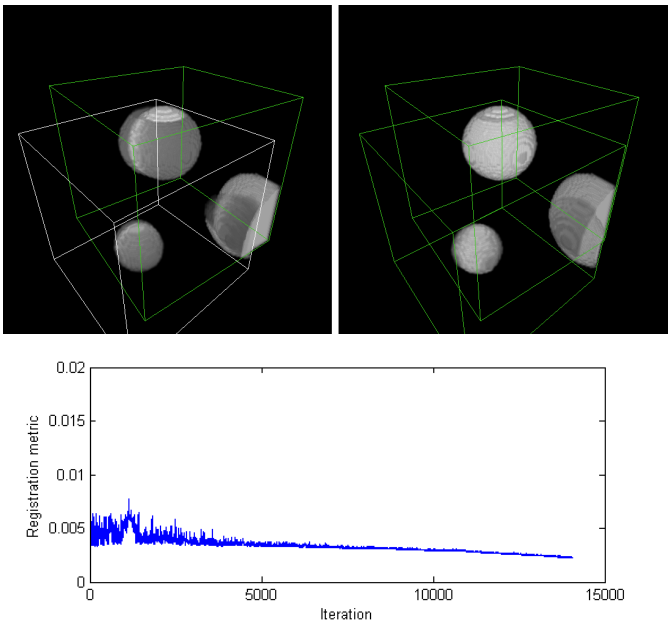

Fig. 4 Synthetic example with three blobs. Upper-left: initial placement, before registration. Upper-right: result of registration. Bottom: registration metric over duration of simulated annealing. The registration metric value depends on the overlapping volume images and the ray casting eye point. A metric value is only significant relative to other values in the same optimization run.

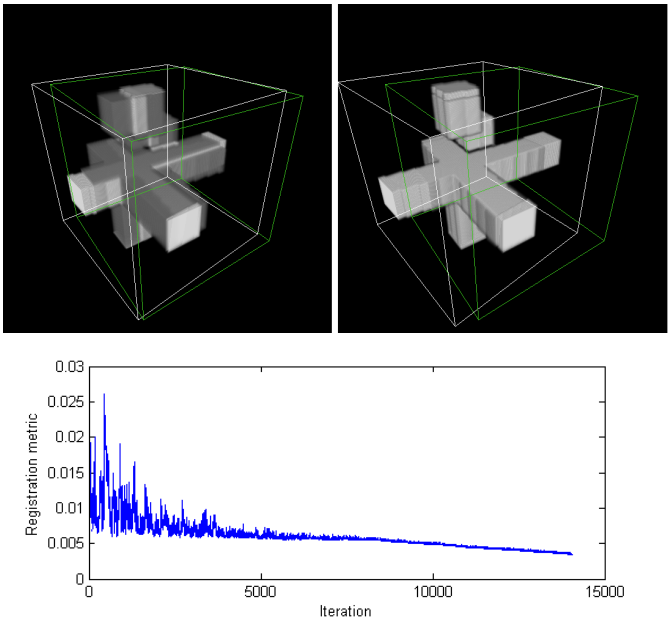

Fig. 5 Synthetic example with axis-aligned bars. Upper-left: initial placement, before registration. Upper-right: result of registration. Bottom: registration metric over duration of simulated annealing.

4 Examples and Results

We have applied our registration system on several classes of input. To demonstrate basic functionality, we show samples from synthetic data sets. Overlapping subsets taken from the *aneurism* data set (courtesy Viatronix, Inc., USA; available at http://www.volvis.org) show successful registration of real-world data. For each example, we show the evolution of the registration metric throughout the optimization run. For test data sets, a metric value of zero indicates perfect registration. A metric value of zero is not possible with real OCT data sets because distinct images of the same area contain random noise. Finally, we show two examples of successfully stitching OCT data sets. Lacking a ground truth for the *in vivo* OCT data sets, we show several cross-sectional slices through the combined volume to demonstrate the improvement brought to the combined image by our method.

4.1 Test Data Sets

The first test case is two volumes sampled from a data set containing three spherical solids, with centers at $(0.2, 0.4, 0.1)$, $(1.1, 0.1, 0.1)$, and $(0.6, 0.7, 0.6)$ and radius

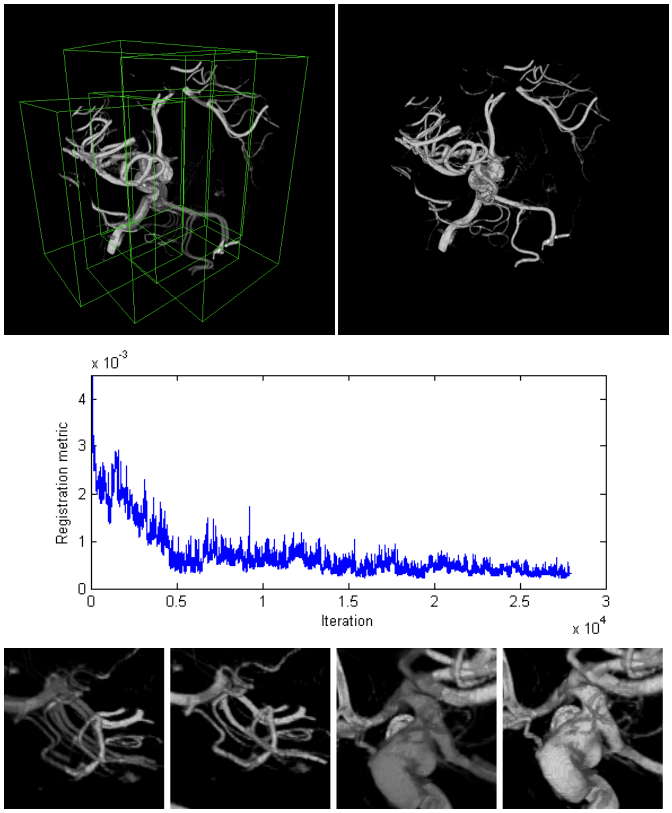

Fig. 6 The *aneurism* volume image. Original image was divided into four overlapping subvolumes, which were then re-registered. Upper-left: initial placement, before registration. Upper-right: result of registration. Middle: registration metric over duration of simulated annealing. Lower row: before-and-after detail views

0.15, 0.40, 0.25 (arbitrary units), as shown by Fig. 4. Starting from a close but inexact manual registration, the system was able to produce an accurately-registered result. The second test case is two volumes sampled from a data set containing three bars, parallel to the $X-$, $Y-$, and $Z-$axes, with a square cross-section 0.2 units on a side and several other features to help quick visual verification of correct orientation. See Fig. 5. The second data set was sampled at a rotation of x radians about the $Z-$axis. The registration was successful.

The third example shows the ability of our system to simultaneously register several data sets. From the *aneurism* data set, we extracted four overlapping subimages. We manually positioned the subimages in rough alignment, then ran our system to refine the initial placement. See Fig. 6 for overall images before and after registration as well as details of vessels interpenetrating several input volumes.

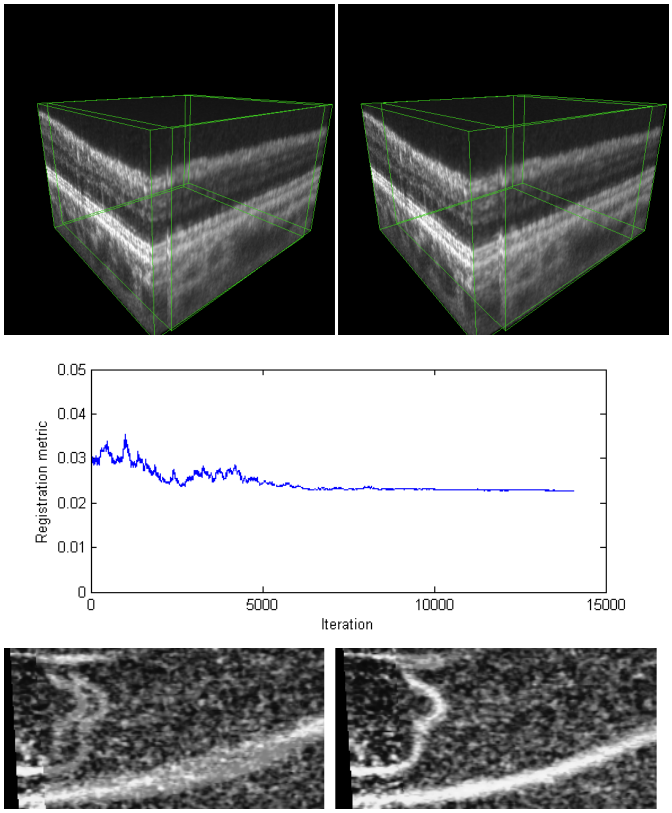

Fig. 7 Two OCT volumes taken around 6° superior retina. Upper-left: initial placement, before registration. Upper-right: result of registration. Middle: registration metric over duration of simulated annealing. Lower row: before-and-after detail views showing improvement in vasculature registration.

4.2 Application to OCT Volumes

We used our technique to stitch OCT volumes of overlapping retinal regions. A retinal region's location is described by visual angle off the line of sight, in degrees, and direction, in terms of temporal or nasal (toward the temple or the nose, respectively) and superior or inferior (physically above or below the fovea, or center of gaze). Fig. 7 shows two overlapping volumes acquired about 6° superior retina, before and after registration. The entire volume is shown, as well as views of various layers within the retina, to illustrate the effect of registration. Likewise, Fig. 8 shows the registration of two data sets taken at 6° temporal, 6° superior retina. In the detail views of both data sets the effects on vessels are clearly seen. While it is possible to register OCT data sets "by hand," features such as vasculature that guide registration generally occur in the middle of the dense OCT data set and are accessible to the user only with some effort. As shown by the detail views of Figs. 7 and 8,

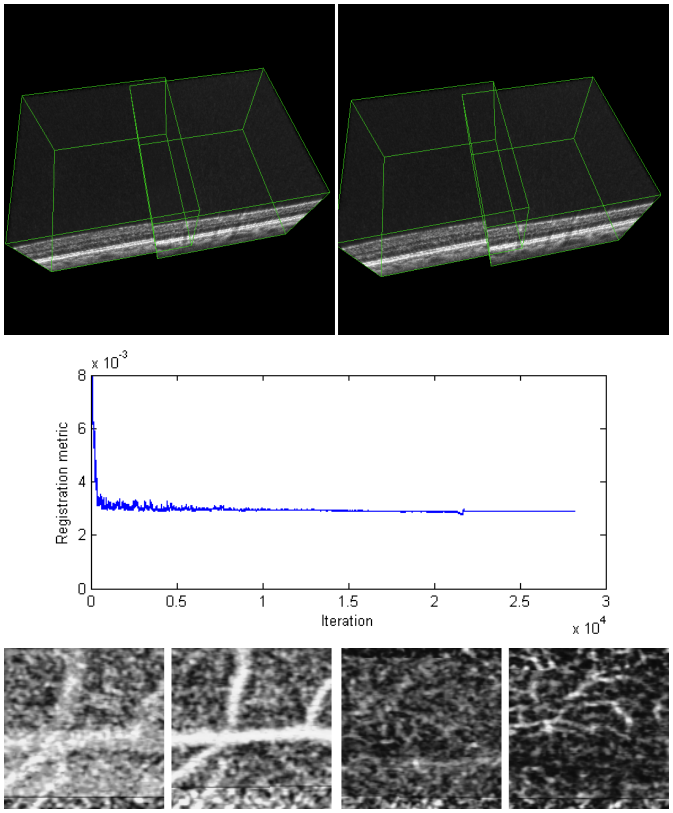

Fig. 8 Two OCT volumes taken around 6° superior, 6° temporal retina. Upper-left: initial placement, before registration. Upper-right: result of registration. Middle: registration metric over duration of simulated annealing. Lower row: before-and-after detail views.

the optimizer gives a major improvement in registration and lets us see and measure detail that would otherwise be lost in noise.

The main difficulty of registering OCT data sets is pervasive eye motion in imaging subjects. Any eye motion will result in distortion in the volume image, and since different motion is present over the course of successive OCT scans the images will not match and will be unable to register without correction for this motion.

5 Conclusions and Further Work

We have presented a system for combined visualization and registration of volume data sets. We perform ray casting through multiple volume data sets. In overlapping regions, we compute the sum of squared difference between volumes at each ray casting sample site. Thus, a metric for the quality of the registration within overlap

regions is calculated in real time for each frame displayed by the ray casting engine. We added an optimizer to incrementally perturb the position of each displayed volume, using the simulated annealing algorithm to find an optimum registration. After showing examples using synthetic and example data sets, we successfully applied our system to two pairs of OCT data sets of the human retina. This registration produced a volume image of high resolution and wide coverage.

We have found, however, that our technique is impractical if one or more of the input volumes contain motion distortion in the overlap region. Motion distortion is present to varying degrees in most OCT volumes. Techniques to mitigate motion distortion will increase the applicability of our system to OCT and other scanning image capture modalities.

Acknowledgements The authors gratefully acknowledge the help of VSRI laboratory members Susan Garcia and Raju Poddar. Support was provided by the National Eye Institute (R01 EY014743) and Research to Prevent Blindness (NY). This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-BOOK-644901.

References

[1] Bria, A., Silvestri, L., Sacconi, L., Pavone, F., Iannello, G.: Stitching terabyte-sized 3D images acquired in confocal ultramicroscopy. In: Biomedical Imaging (ISBI), 2012 9th IEEE International Symposium on, pp. 1659–1662 (2012). DOI 10.1109/ISBI.2012.6235896

[2] Cai, W., Sakas, G.: Data intermixing and multi-volume rendering. Computer Graphics Forum **18**(3), 359–368 (1999). DOI 10.1111/1467-8659.00356

[3] Dalvi, R., Hacihaliloglu, I., Abugharbieh, R.: 3D ultrasound volume stitching using phase symmetry and Harris corner detection for orthopaedic applications. Proc. SPIE **7623**, 762,330.1–762,330.8 (2010). DOI 10.1117/12.844608

[4] Hadwiger, M., Beyer, J., Jeong, W.K., Pfister, H.: Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. Visualization and Computer Graphics, IEEE Transactions on **18**(12), 2285–2294 (2012). DOI 10.1109/TVCG.2012.240

[5] Jacq, J.J., Roux, C.: A direct multi-volume rendering method aiming at comparisons of 3-D images and models. Information Technology and Biomedicine, IEEE Transactions on **1**(1), 30–43 (1997)

[6] Kainz, B., Grabner, M., Bornik, A., Hauswiesner, S., Muehl, J., Schmalstieg, D.: Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs. ACM Trans. Graph. **28**(5), 152:1–152:9 (2009). DOI 10.1145/1618452.1618498

[7] Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)

[8] Kirmizibayrak, C., Yim, Y., Wakid, M., Hahn, J.: Interactive visualization and analysis of multimodal datasets for surgical applications. Journal of Digital Imaging **25**, 792–801 (2012). DOI 10.1007/s10278-012-9461-y

[9] Lindholm, S., Ljung, P., Hadwiger, M., Ynnerman, A.: Fused multi-volume DVR using binary space partitioning. Computer Graphics Forum **28**(3), 847–854 (2009). DOI 10.1111/j.1467-8659.2009.01465.x

[10] Lux, C., Fröhlich, B.: GPU-based ray casting of multiple multi-resolution volume datasets. In: Advances in Visual Computing, *Lecture Notes in Computer Science*, vol. 5876, pp. 104–116. Springer Berlin Heidelberg (2009). DOI 10.1007/978-3-642-10520-3_10

[11] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Optimization Research. Springer-Verlag (2000)

[12] Poddar, R., Cortés, D.E., Werner, J.S., Mannis, M.J., Zawadzki, R.J.: Three-dimensional anterior segment imaging in patients with type I Boston keratoprosthesis with switchable full depth range swept source optical coherence tomography. Journal of Biomedical Optics **18**(086002), 1–7 (2013). Although this citation describes imaging of the anterior segment, we use retinal images produced by the system.

[13] Rößler, F., Botchen, R., Ertl, T.: Dynamic shader generation for GPU-based multi-volume ray casting. Computer Graphics and Applications, IEEE **28**(5), 66–77 (2008). DOI 10.1109/MCG.2008.96

[14] Wojtkowski, M., Leitgeb, R., Kowalczyk, A., Bajraszewski, T., Fercher, A.F.: *In vivo* human retinal imaging by Fourier domain optical coherence tomography. J. Biomed. Opt. **7**, 457–463 (2002). DOI 10.1117/1.1482379

[15] Zawadzki, R.J., Choi, S.S., Fuller, A.R., Evans, J.W., Hamann, B., Werner, J.S.: Cellular resolution volumetric *in vivo* retinal imaging with adaptive optics–optical coherence tomography. Opt. Express **17**(5), 4084–4094 (2009). DOI 10.1364/OE.17.004084

[16] Zawadzki, R.J., Fuller, A.R., Choi, S.S., Wiley, D.F., Hamann, B., Werner, J.S.: Improved representation of retinal data acquired with volumetric Fd-OCT: co-registration, visualization, and reconstruction of a large field of view. In: Proceedings of SPIE, the International Society for Optical Engineering, pp. 68,440C–1. Society of Photo-Optical Instrumentation Engineers (2008)