EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2006)
Thomas Stahovich and Mario Costa Sousa (Editors)

Sketch-based Volumetric Seeded Region Growing

H. L. J. Chen[1] F. F. Samavati[1] M. C. Sousa[1] J. R. Mitchell[1,2] †

[1]Department of Computer Science, University of Calgary, Canada
[2]Seaman Family MR Research Centre, Foothills Medical Centre, Calgary, Canada ‡

Abstract

Interactive volume segmentation is an essential and important step in medical image processing. Conventional interactive methods typically demand significant amounts of time and do not lend to a natural interaction scheme with the 3D volume. In this paper we present a sketch-based interface for seeded region growing volume segmentation. In our approach, the user freely sketches regions of interest (ROI) directly over the 3D volume. Parts of the volume outside the ROIs are then automatically cut out in real-time. The user repeats this process as many times as necessary until he/she decides to specify the seed point 3D location directly at the ROI. To prevent unexpected segmentations, the region growing is restricted to the specified ROI. Our sketch-based system utilizes GPU programming to achieve real-time processing for both rendering and volumetric cutting independent from the size and shape of the sketched strokes.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Segmentation, partitioning

1. Introduction

Medical imaging systems, such as computerized tomography (CT), magnetic resonance imaging (MRI) and ultrasound, are becoming increasingly ubiquitous. Clinicians and surgeons often use computer-based segmentation to identify and analyze anatomical structures of interest in medical image datasets. For example, neuroradiologists often segment and examine the internal carotid artery to determine its degree of stenosis in patients suffering from transient ischemic attacks (TIAs - "mini" strokes). The degree of carotid stenosis is a critical factor to determine if TIA patients should have surgery to open up this vital vessel. Other measurements (such as the shape, topology, and cubic volume) could also be obtained during the segmentation process [ONI05]. Therefore, volume segmentation is an essential and important step in medical image processing.

Segmentation is often broken down into "edge based" or "region based" methods. Each of these in turn may be "manual" or "computer assisted" (including completely automatic). Along the edge-based category, a typical manual segmentation process requires a trained specialist to draw contours around the region of interest (ROI) on cross-sectional images. These contour lines are then linked and reconstructed into a 3D representation for further analysis (Figure 1, top). This procedure can become a challenging task if the target is, for example, blood vessels in the brain, which by nature involves complex shape and unpredicted turning directions. Automatic methods currently focus on low-level features such as edge detection and texture analysis. An example of an edge detection algorithm exists in the use of histograms by considering the relationship between three quantities: the data value and its first and second directional derivatives along the gradient direction [KD98]. A number of contributions and efforts were made in the research direction for obtaining automatic segmentation results. However, the difficulty for a complete automatic approach is limited in one sense or another. Kirbas and Quek [KQ03] pointed out that all such attempts for developing automatic segmentation algorithms are limited to some global parameters or can fail with certain data.

The region growing [RK82] algorithm is one of the well-

† http://www.ImagingInformatics.ca
‡ http://www.mrcentre.ca

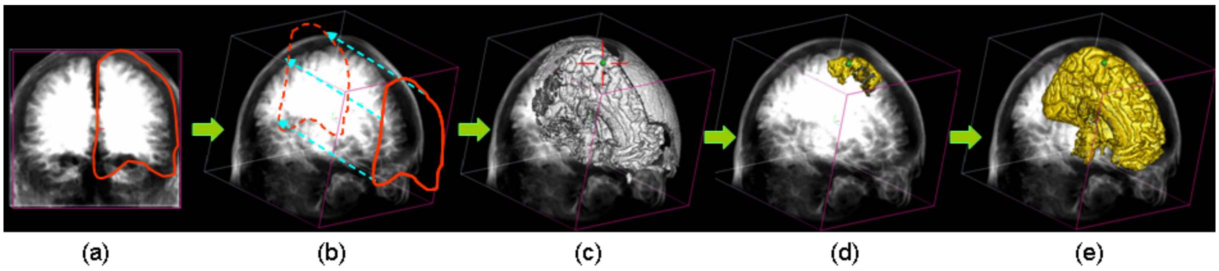

Figure 2: *Our sketch-based volume segmentation method: user sketches a ROI directly over the data (a), the ROI is extruded (b), volume outside is cut out and user plants the seed point (c), region grows (d) and segments volume portions within the extruded ROI (e).*

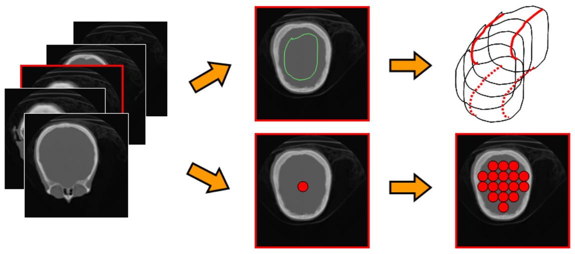

Figure 1: *Conventional segmentation methods. Top row: edge-based method. Bottom row: region-based method.*

known region-based segmentation methods that is simple to compute and applicable to a wide range of data types. Seeded region growing was first introduced by Rolf Adams and Leanne Bischof [AB94]. Their algorithm requires the planting of an initial seed point in the 3D volume dataset. However, the challenge of specifying a 3D coordinate from a 2D device, such as the mouse, is associated with providing an intuitive interface in assisting with the mapping process. Existing methods for specifying the seed point [SHN03] can be outlined as follows: the user navigates from a stack of 2D image slices; a desired slice is selected (i.e. equivalent to selecting one of the axis as a first step); and then the user places the seed point from the cross-sectional view of the data (Figure 1, bottom). As a result the seed point is propagated to the entire volume based on certain criteria the user defines.

The key limitations with the conventional seeded growing region process are the large amount of cross-sectional images a user has to go through. The user is also required to have a priori knowledge of the data in order to quickly identity the correct slice number and the appropriate seed location on the 2D grey-scaled image. This procedure demands a significant amount of time and does not lend to a natural interaction scheme with the 3D volume (i.e. direct manipulation of the 3D data).

In this paper, we propose a sketch-based interface for volumetric seeded region segmentation. Figure 2 illustrates the key stages of our method applied over a raw MRI super-brain dataset (152x154x181). At first, the user loads the volumetric data and defines an intensity range from the histogram. And then the user directly sketches a ROI over the displayed volume (Fig. 2, a). The system extrudes the ROI along the viewing direction within the entire volume (Fig. 2, b - dotted lines). The volume outside the extruded ROI is cut out and the user places the seed at the red cross (Fig. 2, c). The region starts to grow (Fig. 2, d) and finally the complete segmentation inside the extruded ROI is obtained (Fig. 2, e). In addition, the user could place multiple sketches from different views to form arbitrary-shaped ROI. Our system uses GPU programming for real-time rendering and interactive sketching. Furthermore, we utilize the stencil buffer to achieve a processing rate that is independent of the sketch complexity.

The rest of the paper is organized as follows. In Section 2, we review related work and current sketch-based interfaces for volume segmentation. In Section 3, we outline our system framework. In Sections 4, 5, and 6, we provide details of our sketch-based system for volume segmentation. Results are discussed in Section 7, and conclusions are presented in Section 8.

2. Related Work

Interactive seeded region growing. Many segmentation approaches have been proposed for the 2D image segmentation task. The set of well-known techniques include thresholding, k-means clustering, watershed segmentation, and level-set methods (see the survey conducted by Pham et. al. [PXP99]). For segmenting 3D medical datasets, these techniques could also be applied and adapted easily by re-using the 2D image techniques. Sherbondy et. al. [SHN03] developed a fast volume segmentation system using GPU. Their work was based on seeded region growing. The seed selection step allows the user to paint seeds by drawing on the sectional views of the volume. Their segmentation merging criteria are based on non-linear diffusion metric. They also incorporated image smoothing algorithms for noise conditions. More recently, Schenke et. al. [SWD05] analyzed the GPGPU paradigm and implemented the seeded region growing method with fragment shaders and VTK. In order to fully take advantage of the GPU parallelism, the user was encouraged to specify as many seed points as possible.

Sketch-based interfaces for volume segmentation. For general sketch-based modeling of volumetric data, Owada et. al. [ONNI03] presented a system that captures hand-drawn sketches and creates volumetric objects with internal structures. Owada et. al. [ONOI04] further extended the interface for users to define internal volumetric textures of a model. The system allowed interactive design and browsing for volumetric illustrations. Recent work for segmenting volumetric data have also focused on incorporating user intervention and developing interactive segmentation systems. Tzeng et. al. [TLM03] developed a novel interface for volume data classification. They allowed the user to draw strokes on the cross-section of volume data that roughly indicate foreground and background regions. The stroke information was used to train a classifier that is designed for segmenting voxels. Yuan et. al. [YZNC05] presented a novel method to cut out volumetric structures by drawing simple strokes directly on volume rendered images. Owada et. al. [ONI05] proposed an intuitive user interface for volume segmentation. The user traces the contour of the target region using a 2D free-form stroke on the screen. The volume catcher system then returns a plausible 3D region inside the stroke.

Similar to Owada's approach [ONI05], the concept of our system extends the stroke and sweeps through the volume. We use histograms as a first classification step whereas they applied opacity transfer functions. In contrast, we adapt closed strokes that include free-form and other variations. Most importantly, our approach allows the user to interact with a simple sketch-based interface for navigating to the ROI instead of browsing through hundreds of cross-sectional slices ([SHN03]; [SWD05]). For seed planting, our technique is fundamentally 3D and the user no longer needs to look at texture-mapped 2D planes. In addition, we enable the user to define a sub-volume of arbitrary shape with few sketches to constrain the region grow and provide rapid segmentation feedback.

3. Particle System Framework

In our sketch-based system, we utilize a particle system framework. Because of the generality and the fundamental design of the framework, the system can be easily extended to work with irregular datasets. Other potential applications include general point-based systems and polygonal meshes (which were converted to a point-cloud).

At the first stage of our system, a desirable range of intensities is selected by using the intensity histogram to define target voxels from the volumetric dataset. Since only a subset of the entire volume is rendered to the scene, we represent the target voxels by a particle system. We use lists of particles for rendering and processing. This avoids the need to traverse the 3D array containing the original dataset every time we access these target voxels.

In order to maintain the lists of particles, we organize them with a central particle system scheme. The particle system contains a list of particle objects. Each particle object can be organized and displayed by using the display list or vertex buffer objects (VBOs). When the display list option is used, each particle object contains an object color if particles do not possess color information. The particle object also maintains a list of particles and each particle contains information such as: position (x,y,z), color (r,g,b,a), and reference to voxel (which contains intensity and gradient). Position is used during the sketch-based volume cutting (Section 4). Reference to voxel is required to locate neighboring voxels during segmentation (Section 5). For rendering (Section 6), position, color, and voxel gradient (normal) are needed.

Alternatively, particle objects can utilize the various VBOs stored in a collection of *particle buffersets*. Each *particle bufferset* contains a vertex buffer (i.e. voxel position), normal buffer (i.e. voxel gradient), and color buffer (i.e. voxel intensity). Each of these buffers is stored on the GPU texture memory using VBO. The required voxels only travel across the system bus once whenever the histogram is defined. Each particle object then maintains only index information into the corresponding *particle bufferset*. Particles are rendered in either X-ray mode or surface mode (Section 6). Each particle object contains an attribute for its assigned rendering mode.

4. Sketch-based Volume Cutting

To place the seed for the region growing, we use a novel sketch-based interface. In the first stage, the user specifies a ROI by a closed free-form sketch on the screen. The extrusion of this sketch forms the ROI and likely contains the target area (organ). This approach has several benefits: it increases the performance of seed-growing after extrusion and cutting, the user is able to navigate and place the seed more easily, and finally it is very intuitive.

The main challenge here is to cut the extrusion from the volume at an interactive rate. With the defined histogram intensity range, a collection of particles is composed from the 3D volume array. The set of particle attributes is packaged into vertex buffer, normal buffer, and color buffer using VBO. These buffers are sent only once and stored on the GPU texture memory for successive rendering and processing. The sketched area is extruded along the view direction and pierces into the entire volume (Figure 3). The computed sub-volume is rendered in the surface mode and the background volume is rendered in the X-ray mode (Figure 9, right). Subsequent sketches affect only the 'visible' sub-volume currently rendered in the surface mode. Then the remaining task is to distinguish the particles that fall 'inside' the extrusion from the ones that are 'outside'.

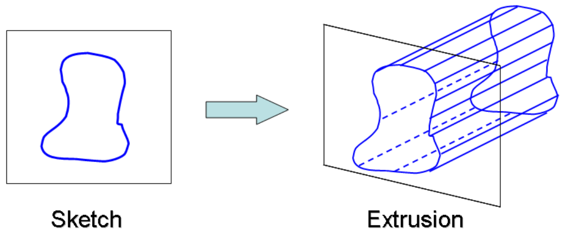

Figure 3: *Sketch extrusion.*

In order to find the list of selected particles that fall inside the sketched extrusion, one possible strategy is to use the standard polygon fill or crossing test algorithms [Hai94] [Fra]. For this, we can project the particle to the screen and check whether it is inside of the sketched stroke. The speed of this method depends on the number of points on the stroke (as a polygon). Unfortunately, this method suffers from a slow speed when the stroke (polygon) has a good quality. Although we could implement the crossing test in GPU, the speed is still dependent on the complexity of strokes and the level of interactivity can vary depending on the user input. Instead, we adapt a novel GPU-based technique that is independent of the sketch complexity.

4.1. Computational Mask

The fundamental concept of our sketch-based system is a real-time filtering process employing a computational mask (Figure 4). We move the mask to traverse the entire volume in a front-to-back order and pick up the particles (or voxels) that are inside the sketched area. The particles which are not visited by this process are labeled as being 'outside'. As depicted in figure 4, the volumetric dataset can be in any orientation with respect to the computational mask.

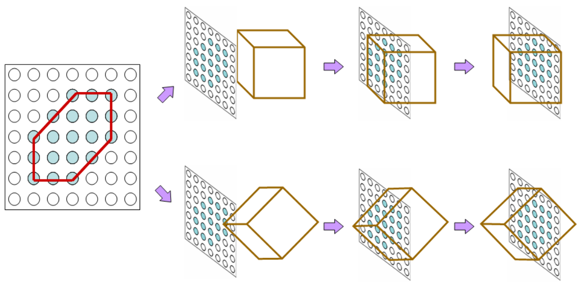

Figure 4: *Computational mask.*

The computational mask is composed of 1s and 0s, where 1 indicates that the pixel is covered by the sketched area and 0 means that the pixel is outside the area. Figure 5 illustrates the process for generating the mask. At first, the user places strokes on the screen and a closed curve is obtained. Next, we fill the enclosing area using the stencil buffer with a 1-bit color [WNDS99]. Then we save the content of the stencil buffer as a texture as demonstrated in Figure 5.

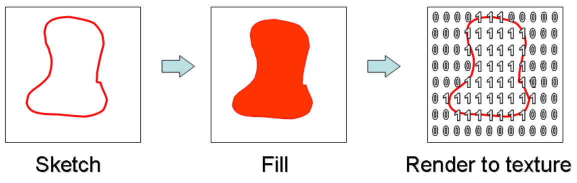

Figure 5: *Generating the computational mask using the stencil buffer.*

4.2. GPU-based Sketch System

In order to quickly filter the entire volume with the generated computational mask, we perform all of our computation in GPU and send back the result as a single texture to the CPU. We leverage the workload to both vertex and fragment shaders and optimize the speed by minimizing the program complexity. Notice that we use the fragments (pixels) and the framebuffer somehow different from their regular functions. Instead of sequential processing of the particles, we map many particles to the fragments at a time. This helps us to use parallel architecture of GPU for processing of the particles. Therefore, the fragments' "position" in our method is an index to the particles instead of being a position of visible pixels. We also use a binary "value" for the fragments to show whether the particle is inside of the extrusion. To map the index of particles, which has a linear order, to the position of fragments, that has two components, we use a 2D texture coordinate buffer. In addition, not all particles can be uniquely mapped to the screen. Consequently, to process all of the particles, we need to do the process in several passes of saving the current screen and mapping a new set of particles. For saving the current screen, which contains binary values, we use a one-bit plane of the framebuffer (off-screen). For example, with a given 100x100 sized screen, we are able to process 10,000 particles for each pass through the graphics pipeline. Figure 6 gives an overview of the processing pipeline.

4.2.1. Preparing Data Buffers for Pipeline Processing

The vertex buffer (1) contains all particles collected from the histogram pre-classification phase. It is not deleted unless the intensity range has been redefined. This enables the system to quickly fetch the target particles whenever a sketched region shall be resolved. This mechanism prevents unnecessary traffic of particles traveling across the system bus for every processing cycle. The texture coordinate buffer (2) stores a 2D array of screen coordinates (0, 0), (0, 1), ..., (s, t), ..., (height - 1, width - 1). Each particle is mapped to a screen coordinate using the associated texture coordinate. During the execution of the processing pipeline, we redirect every particle to its designated screen location.

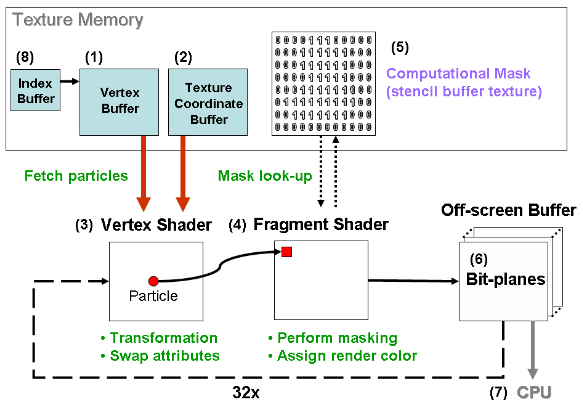

Figure 6: *Sketch system implemented in GPU.*

4.2.2. The Vertex and Fragment Programs

The vertex shader (3) is used to perform the particle coordinate transformation. In order to rasterize the current vertex (e.g. rendered as a 3D point) to the designated fragment location, we swap the incoming attributes as follows. The vertex (input) is multiplied by the model-view matrix, and the result is assigned to the texture coordinate (output). The accompanied texture coordinate (input) is multiplied by the projection matrix, and the result is assigned to the position (output). After the vertex shader has finished processing, both the resulting texture coordinate and position are rasterized and passed onto the fragment shader.

The fragment shader (4) performs the masking operation and assigns a pre-defined render color if the mask value is valid. The input texture coordinate (i.e. the particle's position assigned by the vertex shader) is adjusted with respect to the projection parameters and the value is looked up from the computational mask stored as a stencil buffer texture (5). If the texture look-up results a value of 1, then the particle processed by the current fragment program is inside the sketched region; otherwise, it is outside.

4.2.3. Parameter Calculations

Note that we only need one bit in the off-screen buffer (6) to store the selection information (i.e. one being selected, and zero being not selected). For a typical off-screen color buffer with RGBA components, and each component having 8-bit resolution, it is possible to encode 320,000 particle selections information by adding all render colors. Thus, the required off-screen buffer dimension is $\lceil\sqrt{N/32}\rceil$, where N is the total number of particles to be processed from the vertex buffer. The calculated buffer dimension becomes the width and height of the off-screen buffer.

4.2.4. Composing the Result

Finally, the CPU (7) receives the texture and decodes the selected particles to construct two index buffers, one containing indices of the selected particles and the other one for the non-selected particles. These index buffers are then sent and stored in the GPU texture memory for the next processing cycle as well as for rendering purposes. In subsequent sketch operations, the index buffer (8) (storing the indices of the previously selected particles) is used to index into the vertex buffer when the 'fetch particles' command has been issued.

5. Seeded Region Growing

After describing a rough estimate of the target area using the sketch-based volume cutting, the user can navigate the volume and place a seed point directly on the visible surface to obtain an accurate segment. To find out the seed location in the 3D object-space from a 2D input device (e.g. the mouse), we use an intuitive interface that is consistent with our sketch-based system. In this interface, the user inputs a visible voxel (particle) by clicking the mouse on the screen. The entered pixel can be associated with several particles in various depths and we need to find the best candidate. To do this, we extend the pixel area to a larger rectangle whose extrusion in the volume contains all the involved particles (see Figure 7). To extrude the rectangle in the volume, we use the same technique as described in section 4. After determining all the involved particles, we select the one that has the shortest distance to the entered seed point (Figure 7). The selected particle is then used as the actual 3D seed point. For the region growing algorithm, we start from the seed point as the current voxel and move to adjacent voxels with intensity values close to the current intensity. We use the breath-first search algorithm as appears in the context of graph traversing techniques [CLRS01]. This approach helps to maintain a balanced and coherent growth. We use a threshold for the closeness of the intensities. It is obvious that the growing process can be sensitive to thresholds and the resulting region can be dramatically enlarged when the threshold is increased by one or two scales. However, as a benefit of our volume cutting tool, we can constrain the growing region to be inside of the cut sub-volume as a rough estimate of the desired region.

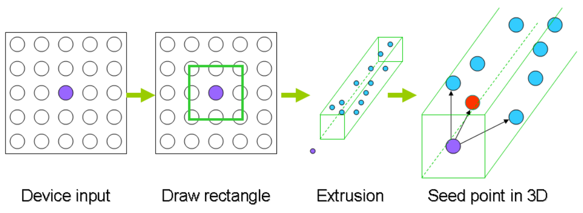

Figure 7: *Searching the seed point.*

6. Rendering

In our approach, we adapt the splatting technique [Wes91] for direct volume rendering using GPU programming. For rendering the volumetric data, each particle associated with a voxel is rendered as a square texture using the OpenGL hardware accelerated point sprite. Point sprite enables us to send only a single vertex information for each particle (voxel) through the rendering pipeline. We adapt the Gaussian kernel as our texture generation function (Figure 8, left).

In the fragment shader, we simply check the incoming opacity value and discard the current fragment if alpha is less than 0.2. We adapt two rendering modes for point-based splatting: X-ray and surface modes (Figure 8, middle and right, respectively).

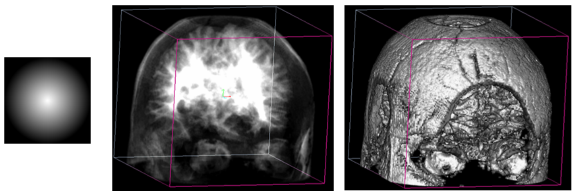
Figure 8: *(left to right) Disk texture with Gaussian distributed transparency values. Different rendering of the brain: X-ray and surface modes.*

The X-ray mode accumulates all fragments to compute the final pixel value with the following OpenGL formulation: $I_f(x) = \alpha_{new}(x) I_{new}(x) + I_f(x)$ [XC04]; where $I_f(x)$ is the frame-buffer intensity value at pixel location x, $I_{new}(x)$ is the incoming fragment value, and $\alpha_{new}(x)$ is the opacity of the new fragment. We use glBlendFunc(GL_SRC_ALPHA, GL_ONE) to perform the accumulation [XC04].

In order to render particles and obtain a surface representation, we apply a two-pass rendering technique that consists of the visibility pass followed by the shading pass [BHZK05]. During the visibility pass, we perform the so-called ε-test operation. For implementing the ε-test, we perform the following steps. First, we scale all the particles with the value of ε in the negative z-direction. Then we render to the depth buffer and turn off the color buffer. During the shading pass, we perform lighting computation for each particle processed by the vertex shader. Note that in both passes, we discard fragments whose opacities are less than 0.2. We also combine the X-ray mode and the surface mode to form the hybrid mode as follows: (1) render the particles (X-ray mode) to the frame buffer using alpha-blending with no lighting and (2) render the particles (surface mode) and perform the visibility pass and the shading pass respectively. However, during the visibility pass while rendering the surface mode particles, we enable writing to both the depth buffer and the color buffer. In the fragment shader, we output black pixels for all fragments processed (i.e. to overwrite the X-ray mode particles).

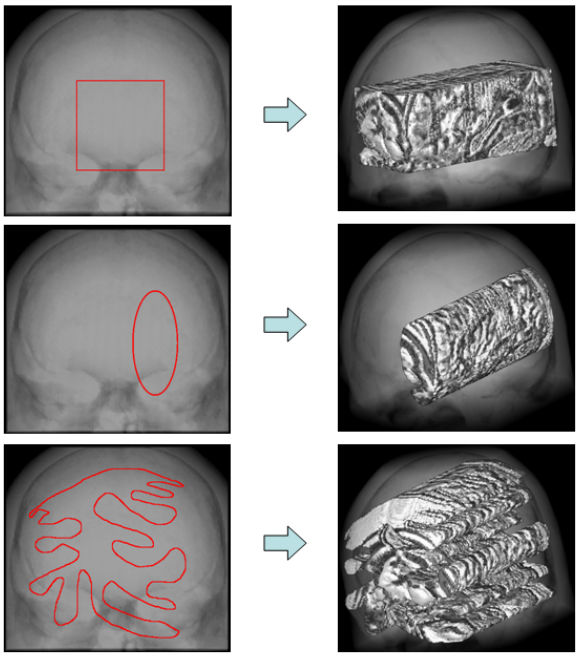
Figure 9: *Sketch types: (a) rectangular strokes, (b) elliptical strokes, and (c) free-form strokes.*

7. Results and Discussions

All the results were generated on an AMD Anthlon 64 X2 3800 with a GeForce 7800 GT, 256 MB card. We selected raw volumetric datasets of the brain (MRI, 152x154x181), skull (MRI, 256^3), and angiography (3T MRT, 256x320x128).

For all datasets, the sketch response time (SRT) was below 1 second. From loading a full-range histogram, Figure 2 shows the segmentation of grey and white matter of the left hemisphere of the brain (SRT = 0.384 sec). Figure 9 illustrates the before/after effects on the brain dataset after sketching rectangular, elliptical and free-form ROIs (SRT = 0.515, 0.392 and 0.384 sec, respectively). Figure 10 (top row) shows a successful segmentation of the right ventricle with SRT = 0.267 sec. Figure 10 (bottom row) illustrates a series of volume cutting after free-form sketched ROIs (SRT = 0.261 sec) and the resulting segmented portions of the teeth. With the 3T MRT time-of-flight angiography dataset of a human head (Figure 11), we were able to quickly segment the carotid and cerebral arteries with SRT = 0.224 sec.

Our system also allowed a real-time preview of seed locations as the user moves the mouse. The interactive rate of seed searching was achieved by utilizing the core system implementation and from the aid of GPU. Note that in order to obtain smooth sketching lines, we froze the background rendering (i.e. the volume splatting) by saving the entire scene to a texture. Thus when the user placed strokes on the screen, we rendered the screen-sized texture first followed by the ROI strokes.

8. Conclusion and Future Work

We presented a novel interface for volume segmentation based on seeded region growing. Instead of the traditional way of browsing from hundreds of cross-sectional slices, we proposed a sketch-based interface for interactive volume exploration and navigation for the ROI. We provided real-time rendering when the user interacts and places the seed point from a truly 3D environment. More importantly, our sketch-based system constrained the region grow from the cut sub-volume to enforce focus-of-attention. In designing from a particle system perspective, our approach can be easily extended to a number of applications including other point-based systems, polygonal meshes, and irregular volume with changing topology.

Future improvements include extending our system with other algorithms for sketch-based volume manipulation. It would also be useful to have the capability of multiple sketched ROIs assigned in different regions of the volume to allow, for instance, better control of the level of detail in selected regions of the dataset. The criteria that we used to judge the quality of the results were solely based on our observations on the speed and flexibility of volume data cutting, exploration, and seed planting/growing control. It is important to conduct more formal evaluations and user/clinical studies to provide quality sketch-based volume segmentation tools for professionals in medical science.

References

[AB94] ADAMS R., BISCHOF L.: Seeded region growing. *IEEE Trans. on PAMI 16*, 6 (June 1994), 641 – 647.

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's gpus. In *Proc. of the Eurographics Symposium on Point-Based Graphics '05* (2005).

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.

[Fra] FRANKLIN W. R.: Pnpoly - point inclusion in polygon test. `http://http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html`.

[Hai94] HAINES E.: Point in polygon strategies. *Graphics Gems IV* (1994), 24–46.

[KD98] KINDLMANN G., DURKIN J.: Semi-automatic generation of transfer functions for direct volume rendering: Methods and applications. In *Proc. of Visualization '98* (1998), pp. 79 – 86.

[KQ03] KIRBAS C., QUEK F.: Vessel extraction techniques and algorithms: a survey. In *Proc. of Bioinformatics and Bioengineering '03* (2003), pp. 238 – 245.

[ONI05] OWADA S., NIELSEN F., IGARASHI T.: Volume catcher. In *Proc. of the Symposium on Interactive 3D graphics and games '05* (2005), pp. 111 – 116.

[ONNI03] OWADA S., NIELSEN F., NAKAZAWA K., IGARASHI T.: A sketching interface for modeling the internal structures of 3d shapes. In *Proc. of 3rd International Symposium on Smart Graphics* (2003), pp. 49 – 57.

[ONOI04] OWADA S., NIELSEN F., OKABE M., IGARASHI T.: Volumetric illustration: Designing 3d models with internal textures. *Proceedings of ACM SIGGRAPH(SIGGRAPH2004)* (2004), 322–328.

[PXP99] PHAM D. L., XU C., PRINCE J. L.: A survey of current methods in medical image segmentation. *In Technical Report JHU/ECE 99-01, The Johns Hopkins University* (1999).

[RK82] ROSENFELD A., KAK A.: Digital picture processing. *New York Academic Press 2* (1982), 138 – 145.

[SHN03] SHERBONDY A., HOUSTON M., NAPEL S.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proc. of IEEE Visualization '03* (2003), pp. 171 – 176.

[SWD05] SCHENKE S., WUENSCHE B., DENZLER J.: Gpu-based volume segmentation. In *Proc. of IVCNZ '05* (2005), pp. 171 – 176.

[TLM03] TZENG F.-Y., LUM E. B., MA K.-L.: A novel interface for higher-dimensional classification of volume data. In *Proc. of IEEE Visualization '03* (2003), pp. 505 – 512.

[Wes91] WESTOVER L.: *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm.* PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1991.

[WNDS99] WOO M., NEIDER J., DAVIS T., SHREINER D.: *OpenGL Programming Guide Third Edition.* Addison-Wesley Publishing Ltd, 1999.

[XC04] XUE D., CRAWFIS R.: Efficient splatting using modern graphics hardware. *Graphics Tools 3*, 8 (2004), 1ąV21.

[YZNC05] YUAN X., ZHANG N., NGUYEN M. X., CHEN B.: Volume cutout. *The Visual Computer (Special Issue of Pacific Graphics 2005) 21*, 8-10 (2005), 745–754.

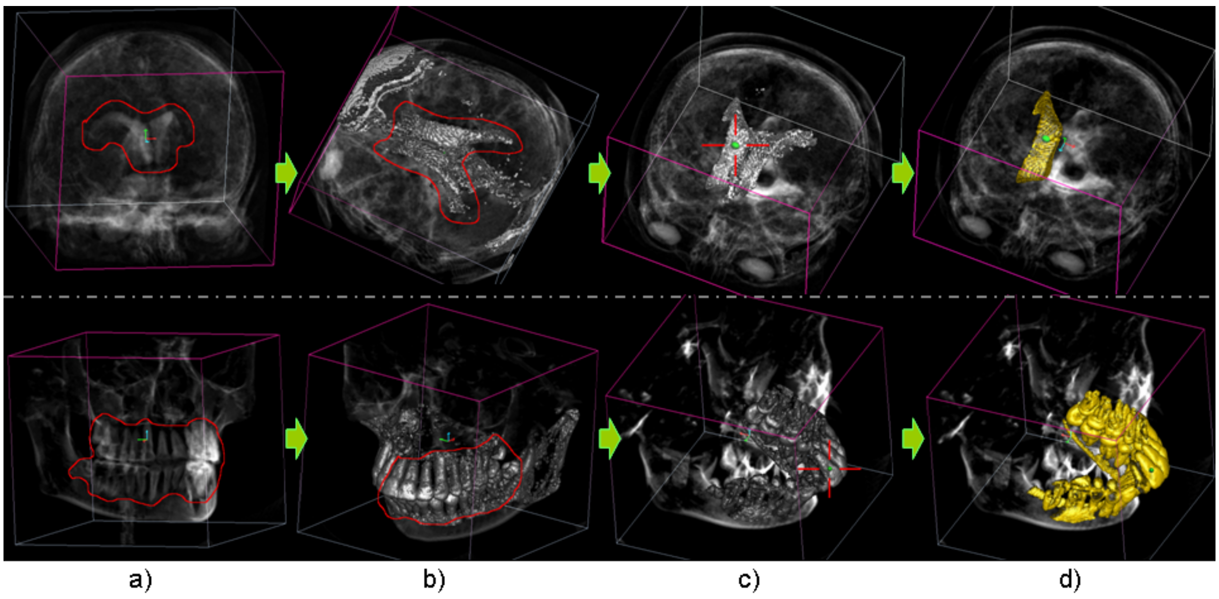

Figure 10: *Segmentation of right ventricle (top) and partial teeth (bottom). (a) Raw volume, X-ray, with sketched-region. (b) Resulting cut, rotating the view, new sketch. (c) Resulting cut, rotating the view, plating the seed. (d) Region growing contained within sketched/resulting volume from (c).*

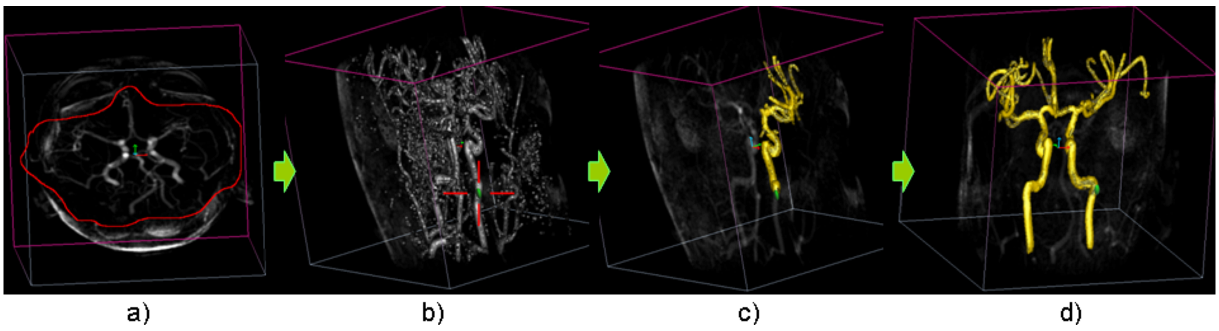

Figure 11: *Segmentation of carotid and cerebral arteries. (a) Raw volume, X-ray, with sketched-region. (b) Resulting cut, rotating the view, plating the seed on the arterial branch. (c,d) Region growing contained within sketched/resulting volume from (b).*