Visual Comput (2008) 24: 689–698
DOI 10.1007/s00371-008-0249-5

ORIGINAL ARTICLE

**Hung-Li Jason Chen**
**Faramarz F. Samavati**
**Mario Costa Sousa**

# GPU-based point radiation for interactive volume sculpting and segmentation

Published online: 29 May 2008

**Electronic supplementary material**
The online version of this article (doi:10.1007/s00371-008-0249-5) contains supplementary material, which is available to authorized users.

H.-L.J. Chen (✉) · F.F. Samavati · M.C. Sousa
Department of Computer Science, University of Calgary,
2500 University Drive, N.W.,
Calgary, Alberta, Canada, T2N 1N4
hljason.chen@ucalgary.ca

**Abstract** Internal structures, features, and properties in volumetric datasets are mostly obscured and hidden. In order to reveal and explore them, appropriate tools are required to remove and carve the occluding materials and isolate and extract different regions of interest. We introduce a framework of interactive tools for real-time volume sculpting and segmentation. We utilize a GPU-based point radiation technique as a fundamental building block to create a collection of high-quality volume manipulation tools for direct drilling, lasering, peeling, and cutting/pasting. In addition, we enable interactive parallel region growing segmentation that allows multiple seed planting by direct sketching on different volumetric regions with segmentation results dynamically modified during the process. We use the same point radiation technique to create high-quality real-time feedback of the segmented regions during the seed growing process. We present results obtained from raw and segmented medical volume datasets.

**Keywords** Point-based techniques · Real-time system · Volume segmentation · Volume cutting

## 1 Introduction

In medical imaging, clinicians and surgeons often use computer-aided techniques to identify and analyze anatomical structures of interest. Many techniques were developed in particular for segmentation (see the survey by Pham et al. [12]). Sherbondy et al. [15] proposed to navigate from 2D images to place a seed point. This point is used for a seeded region growing algorithm for determining the area of interest. This kind of technique has three major issues to be addressed: how to navigate, how to plant the seeds, and how to control the growing threshold. Current medical volume datasets contain hundreds of slices and require domain expertise for understanding the cross-sectional representation. An ideal segmentation environment would simulate physical tools allowing users to directly operate on 3D datasets and carve them. The second issue is how to plant the seeds. Due to the noise in the medical image acquisition process, materials with similar intensity values are sometime disconnected, resulting in an incomplete segmentation of the entire organ. It would be beneficial to perform region growing on a group of the seeds in a parallel fashion. The third issue is how to control the region growing algorithm. Using static thresholding methods is not an ideal solution since different materials in the datasets can have very close intensity values.

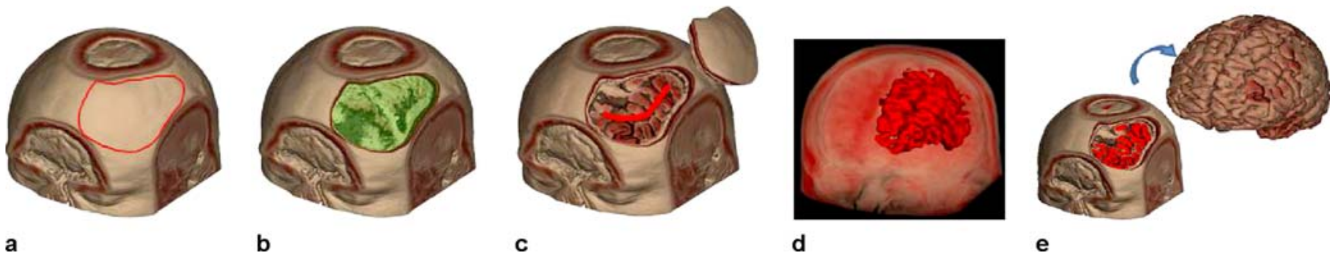

**Fig. 1a–e.** Key processes of our interactive volume manipulation framework: **a** a user indicates a region for opening with a stroke, **b** a surface-based peeling operation is performed with user-specified depth, **c** the skull layer is removed and the user sketches seeds for segmentation, **d** the region grows, and **e** the grey and white matter are segmented and isolated

This is an issue and requires a trial and error process for tweaking the thresholds. Addressing all of these issues, in addition to real-time feedback and high-quality rendering, makes the problem even more challenging.

We present a framework of interactive tools for real-time volume manipulation and segmentation (Fig. 1). As a new contribution, we introduce a GPU-based point radiation technique as a fundamental building block for high-quality volume carving (Sect. 3). The point's radiation is used to create smooth anti-aliased results as well as a collection of interactive raycasting based tools for direct drilling, lasering, peeling, and cutting/pasting the 3D volume (Sect. 4). In addition, we enable interactive region growing segmentation and allow multiple seed planting by direct sketching on different regions (Sect. 5). To obtain rapid feedback, we introduce a new parallel region growing technique that concurrently operates on all the sketched seeds. As a novel feature of this technique, segmentation results are allowed to be dynamically modified through a series of undo, redo, and resume operations. In our approach, seeds are processed as points using the programmable hardware. We maintain multiple seeds by storing their state information in a separate 3D buffer. Again, we utilize point radiation to create an anti-aliased seed map and render the region growing result with high-quality raycasting.

## 2 Related work

Our framework (Fig. 1) is based on components related to volume clipping, volume sculpting, and volume segmentation with seeded region growing.

*Volume clipping.* Volume clipping provides a means to expose parts of the volume with cutting planes or more complicated geometry. Weiskopf et al. [20] proposed interactive clipping techniques that exploit the graphics hardware. They presented depth-based clipping, using the depth structure of an object, as well as clipping via a voxelized clip object, utilizing 3D textures and distance fields. McGuffin et al. [10] presented a method for browsing the volume with interactive manipulation widgets by assigning individual voxels as the fundamental primitive. Huff et al. [8] exploited programmable hardware and proposed erasing, digging, and clipping operations to uncover hidden structures in the volume data. Recently, Correa et al. [5] proposed a set of operators (peeler, retractors, pliers, and dilators) that can be placed anywhere on or within the volume. However, these approaches require either preconstructed clipping objects or procedurally defined operators. And, for most of the hardware-based clipping methods, the algorithm is computed in the context of 3D texture rendering and hence requires every voxel to be processed every frame. In our point-based approach, voxels are clipped and processed only if they are affected by the points emitted by the tool.

*Volume sculpting.* This refers to a modeling technique for sculpting a solid material with a tool, which modifies values in the voxel array. Sculpting tools are used to add, remove, paint, and smooth material. Galyean and Hughes [7] adapted a 3D device to sculpt a block of material bit-by-bit with the additive tool, heat gun, and sandpaper. Avila and Sobierajski [1] incorporated a 3D haptic device to simulate virtual sculpting tools by applying 3D filters on the properties of the volume data. Ferley et al. [6] presented a sculpting metaphor for rapid shape prototyping with 3D input devices. In general, sculpting with a 3D device can be a challenging task as parts of the volume can be occluding the tool itself. Moreover, it can be difficult to visualize the 3D location of the virtual tool relative to the target volume. Wang and Kaufman [18] proposed a carving and sawing tool utilizing a 3D splatting method with a hyper-cone filter. Their approach is primarily in the context of solid modeling.

*Volume segmentation.* Well-known segmentation techniques such as thresholding, k-means clustering, watershed segmentation, and level-set methods have been applied in segmenting volume datasets. Tzeng et al. [17] proposed an intuitive user interface for specifying high-dimensional classification functions by painting directly on sample slices of the volume. Owada et al. [11] developed a volume catcher system in which the user traces the contour of the target region by drawing a 2D free-form stroke over the displayed volume. The seeded region growing method has also been explored for segmenting volumes [9]. Sherbondy et al. [15] developed a volume segmentation system that allows the user to paint seeds by drawing on the sectional views of the volume. However, selecting from hundreds of slices and pinpointing the correct location on the 2D image require both highly trained personnel and time commitment. Chen et al. [3] presented a 3D seeded region segmentation system with splatting rendering. They allowed volume cropping with a sketch-based interface and enabled seed searching directly on the 3D surface. The drawback in their seed selecting approach is that only one seed can be explored at a time and the region growing has to be computed off-line. Recent approaches exploited programmable hardware for accelerating the region growing algorithm [14, 15]. Their seed growing computation requires rendering to sections of a 3D texture and must iterate through all layers of the volume to complete a single growing step. In our point-based approach, we grow seeds based on the currently active voxels that consist of only a small percentage compared to iterating through the entire volume every frame (e.g. 100 vs. $512^3$ computation cycles), achieving a dramatic performance improvement with local updates. In addition, we have found that processing multiple seeds from the input sketch in parallel fashion and having the feature for dynamically modifying the threshold are very crucial in the medical datasets. These features are exclusively supported in our real-time and point-based segmentation method.

## 3 GPU-based point radiation

We propose a set of real-time volume manipulation tools (Sect. 4) all based on the fundamental concept of *point radiation*. The main idea is to create a set of 3D points (from the proposed tools) associated with the existing voxels. Based on the tool, the intensity values of the corresponding places in the dataset are changed (e.g. by removing the material). The binary use of points creates aliasing artifacts. To address this issue, we assume that each point has a continuous field of energy (radiation) that drops smoothly off to zero. Obviously, taking the radiation into consideration increases intensively the computational load for an interactive application. We address this by taking advantage of the programmable graphics hardware. This achieves a dramatic performance gain with a minimum factor of 128 on a GeForce 8800 and 320 on an ATI Radeon 3800. In fact, point radiation extends naturally from a hardware-supported function – point sprite. Point sprite is a two-dimensional billboard method for rendering a textured image from a single point in space, whereas point radiation establishes a three-dimensional metaphor for rendering a filtered volume from a point sample. The basic idea of point radiation is also close to the concept of 2D splatting, which was first described by Westover [21] as a footprint evaluation for object-order volume rendering. The splatting scheme allows every input sample to be treated individually and therefore it is suitable for parallel execution. Our point radiation method, as an energy distribution process, produces a three-dimensional footprint. Finally, the concept of point radiation is also similar to the field function of a point's primitive in implicit modeling [2]. In implicit modeling, various primitive volumes are used and then are blended with a tree-like structure [23]. In our approach, we do not rely on implicit functions and sorting hierarchical blending operators. We only work with point primitives and directly operate on volumetric datasets. Therefore, our approach falls into the category of point-based modeling and rendering.

### 3.1 Point radiation methodology

For the input 3D point $\boldsymbol{p} = (x, y, z)$, we use a Gaussian distribution function for spreading energy radially into the volume space around $\boldsymbol{p}$. The Gaussian function smooths neighborhood elements and provides high-quality anti-aliased rendering. The kernel of the Gaussian function is defined by a radius $R$, in terms of number of voxels. The $2R \times 2R \times 2R$ region forms a 3D footprint in the volume space. To compute the weight $\omega$ at a particular voxel $\boldsymbol{v}$ of the footprint, we only need to apply a 3D Gaussian function: $\omega = \text{Gaussian3D}(\boldsymbol{v})$. In this work, we use a simple summation to accumulate the energy contributions from all input points.

For implementing the point radiation technique, we utilize the geometry shader and its ability to render to 3D textures [16]. The geometry shader allows point primitives to be amplified and redirected to any location in the 3D output texture. One possibility is to use these features for creating and evaluating the 3D footprint of the input point and blend it with the output volume. However, if we directly instruct the geometry shader to generate the set of all footprint points, the geometry shader may become overloaded. To reduce the number of points processed by the hardware, we utilize the point sprite hardware acceleration function, which is commonly used in most splatting techniques to achieve real-time rendering. It allows a point to be rasterized into a square region consisting of fragments containing relative coordinates with respect to the input point (e.g. fragment at the lower-right corner has coordinates (1, 1)). To reconstruct the set of 3D footprint voxels, we use a stack of 2D point sprites (Fig. 2) to largely reduce loads on the geometry shader.

To compute the energy at a footprint voxel, we need to sample the 3D Gaussian function. Theoretically, we could directly sample this function by evaluating it at a 3D coordinate. But this requires us to store a 3D texture in

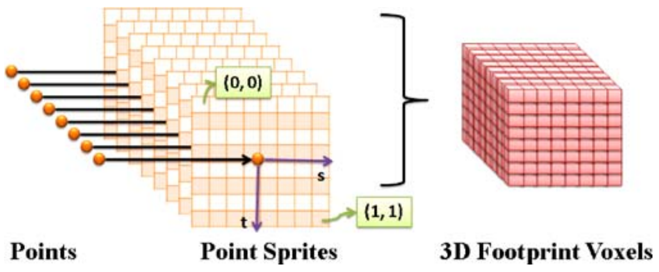

**Fig. 2.** Points are emitted as 2D point sprites by a single geometry program and reconstructed into 3D footprint voxels. Each 2D point sprite contains automatically generated texture coordinates ranging from (0, 0) to (1, 1)

the GPU memory to represent the Gaussian kernel. When composing a high-resolution sampling, a large amount of memory has to be allocated for storing the 3D texture. We use the fact that the 3D Gaussian function can be evaluated by multiplying 2D and 1D Gaussian functions as follows:

$$\begin{aligned}
&f(x, y)f(z)\\
&\quad = a^2 \exp\left(-\frac{(x-x_0)^2+(y-y_0)^2}{2b^2}\right) a \exp\left(-\frac{(z-z_0)^2}{2b^2}\right)\\
&\quad = a^3 \exp\left(-\frac{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}{2b^2}\right)\\
&\quad = f(x, y, z),
\end{aligned}$$

where $a > 0$ and $b$ are the parameters of the Gaussian function and $(x_0, y_0, z_0)$ is its center. This scheme reduces memory consumption since we only need to store a 2D texture and a 1D texture in the GPU. Then, the Gaussian functions are reconstructed using programmable graphics hardware. To begin the radiation process, the geometry shader receives the position and attributes of an input point $\boldsymbol{p}$ fetched via the vertex shader. Next, we compute the point radiation by first transforming the position of $\boldsymbol{p}$ into a screen coordinate to prepare it for rasterization. Then, we compute the first layer of the 3D footprint with a viewport transformation scheme using the equation layer $0 = \text{round}(\boldsymbol{p}_z/\zeta) - R$, where $\boldsymbol{p}_z$ is a normalized depth value of $\boldsymbol{p}$ and $\zeta$ is the depth of the voxel dimensions. Next, we iterate from layer $0$ to layer $2R-1$ and duplicate a point primitive for each layer to reconstruct the 3D footprint. Each point should be associated with two attributes: (1) the ZWeight sampled from the 1D Gaussian texture computed from the normalized layer coordinate and (2) the sample value $s$ of point $\boldsymbol{p}$ (e.g. intensity value). After the rasterization step, the set of points emitted by the geometry shader are converted into 2D point sprite fragments. In the fragment shader, we look up the 2D Gaussian texture with the 2D point sprite coordinate and combine the result with ZWeight to form a 3D energy value $\omega$ corresponding to a 3D footprint location. The final fragment value is then combined with the sample value $s$ and blended into the radiation volume[1].

## 4 Interactive volume tools

Based on the point radiation technique, we develop novel tools for sculpting volumes and removing occluding materials. We were inspired by traditional medical illustrations depicting clinical procedures using physical operations like drilling and peeling. In our system, the user specifies a closed-curve region directly over the volume to define the tip shape of the sculpting tool or the volumetric region of interest for uncovering. This closed-curve region is then used to construct a computational mask in which each element can penetrate/cut through the volume using a local geometric property such as normal directions on the volume surface.

### 4.1 Mask generation

The first step is to generate a binary computational mask, where 1 indicates that the pixels are contained in the sketched area and 0 otherwise. Therefore, a closed curve is necessary to divide the area to inside and outside. This closed curve can be defined in two ways (1) as simple shapes (circles, squares, etc.) approximating specific sculpting tools or surgical medical devices or (2) as closed curves freely sketched by the user approximating surgical cuts, for instance. We create the freely sketched curve by enclosing the piecewise-linear curve strokes created from the input points. Sketching a fine curve on the screen requires fast processing of the stylus input when complex rendering is involved. Since rendering the volume data is a costly operation, simultaneously sketching and rendering the volume is deemed to degrade the curve quality. In order to obtain smooth sketching, we freeze the background rendering (i.e. the volume raycasting) by saving the entire scene to a texture. Thus, when the user places strokes on the screen, we render the screen-sized texture first followed by the input strokes. This avoids delays caused by the concurrent rendering of both the sketch and the volume data. To generate the mask, we fill the enclosing sketch area using the stencil buffer with a 1-bit color [22]. Then, we save the content of the stencil buffer as a texture.

Direct use of a binary mask for sculpting or removing material from the volume can introduce aliasing effects. To avoid aliasing in the image space, we modify the mask generation process and perform a post-filtering step. Instead of using a mask with a binary format, we use floating points to represent intermediate values between areas covered by the sketch and outside the sketch.

[1] For blending, we use glBlendFunc(GL_SRC_ALPHA, GL_ONE) to perform the accumulation.

This method produces a smooth blending on the boundary of the sketched region and further prevents aliasing in the volume space. In the post-filtering process, we adapt stochastic sampling and apply jittering on a regular grid. Jittering is a method that trades aliases with noise [4], where new pixel positions are sampled within a sub-pixel (i.e. grid cell) – the final color is then reconstructed with some scheme. Here, we choose the cubic B-spline reconstruction filter as it provides smooth results while maintaining sharpness and details [19].

## 4.2 Drilling

We propose a drilling tool that allows the user to carve into the volume by sweeping a mask along the viewing direction. The mask represents the shape of the drill's tip that can be a circle, ellipse, square, or any other shape sketched by the user. We match the drilling mask to the view plane. This, in addition to the use of the viewing direction, is a natural selection and helps to reduce the user interventions.

Upon receiving a pressure change (from a stylus) or slight movement, we find the surface points by casting rays from the mask plane into the volume space (Fig. 3a). To perform drilling, we attach the mask elements to the surface points and execute individual point radiation operations. We repeat this process according to the depth of drilling: $\text{depth} = \text{depth}_{\max} \times \text{pressure}_n$, where $\text{depth}_{\max}$ is a maximum drilling depth allowance for each surface detection operation; $\text{pressure}_n$ is the instantaneous pressure normalized to [0, 1], supplied either by a stylus or from a second input source; and depth is an integer defining the succession for the mask radiation process. Based on our experiment, setting the $\text{depth}_{\max}$ value to the range of 8 to 12 is suitable for a volume of size $256^3$. Having a maximum drilling depth of 12 prevents excessive removal of volume materials for every stylus input and helps maintain stylus responsiveness. Successive mask operations offset the positions of the mask elements with a small amount (i.e. relative to the size of a voxel) along the view direction.

## 4.3 Lasering

To enable more flexibility, we also propose another carving tool which is slightly different from the drilling tool

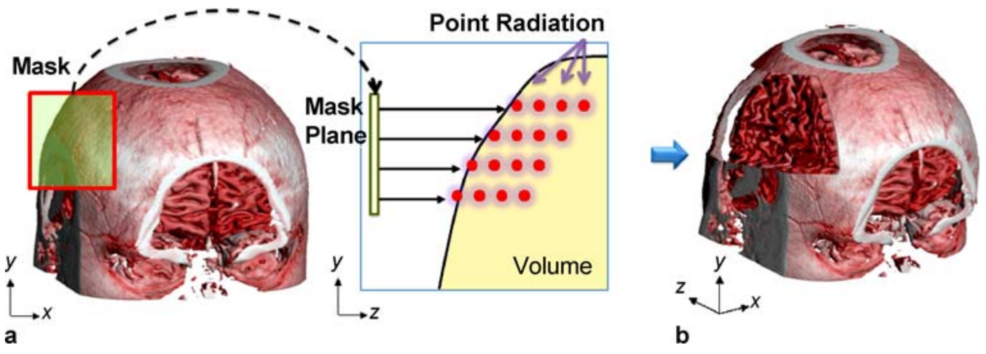

**Fig. 3. a** Drilling the super-brain data with a square mask, raycasting into the volume to find surface points, and executing point radiation along the view direction. **b** Applying multiple pressures to cut through the skull

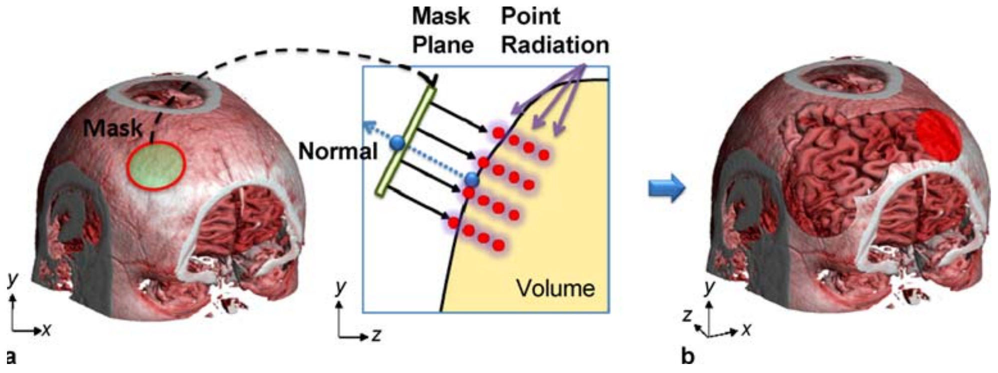

**Fig. 4. a** Lasering the super-brain data with a circular mask, raycasting into the volume to find surface position and normal, and executing point radiation parallel to the mask. **b** Moving and applying various pressures to remove the skull layer

and can simulate a 3D laser wand. In this tool, we sweep the mask along the normal of the visible surface. And, the laser mask (the tool's tip) is orthogonal to the gradient of the visible surface. As the mask moves, we find a surface point by casting a single ray from the center of the mask into the volume space. The geometric information on this point, surface position and normal, is then used for relocating and orienting the mask in 3D. When pressure is applied on the mask, we execute point radiation by emitting points parallel to the oriented mask (Fig. 4a). The depth of lasering is computed similar to the drilling tool with the amount of input pressure applied from a stylus. By setting the maximum depth value to the range of 18 to 22, it works well for a volume of size $256^3$ and prevents the laser operation from removing excessive surface layers (Fig. 4b). To minimize the user input, in our final implementation, we have not given flexibility for changing the sweeping direction, although our framework is capable of this feature.

### 4.4 Peeling, cutting, and pasting

To facilitate removing a large surface region at once (e.g. opening the skull), we propose a peeling tool.

The peeling operation can be seen as wrapping a surface region sketched by user into a mask. To create the impression of peeling, each point of the mask, associated with a point on the visible surface, should move along the normal of the surface at that point. Therefore, for the peeling tool we have a variable sweeping direction as opposed to the drilling or lasering tool's constant direction. Consequently, the condition of mask peeling holds only if the target surface contains a smooth change of gradients. To precisely control the peeling layers, the peeling operation is performed by adjusting a slider or dragging a pen on the tablet instead of applying pressures. After composing the peeling mask, we cast parallel rays from the mask plane into the volume space to find the individual surface position and normal direction for each element on the mask (Fig. 5a). To perform the peeling operation, we relocate the mask elements to the detected surface points and emit independent point radiation along the inverse normal as the cutting path.

To simultaneously indicate the sculpted portion, we introduce the cut and paste tool that allows the region *cut* by the peeling tool to be *pasted* back with a different position and orientation in the scene (Fig. 5b) through an additional rendering criterion. In a *cut* scene, voxels with associated radiation values (i.e. sampled from the radiation volume) less than 0.5 are rendered. In a *paste* scene, the visibility criterion is reversed (i.e. those with radiation values greater than or equal to 0.5 are rendered instead). To simultaneously display the *cut* and *paste* scenes, we perform dual rendering with our GPU-based raycasting engine. As the radiation values range from 0 to 1.0, we select 0.5 as a natural candidate for distinguishing the cut and paste rendering.

## 5 Interactive seeded region segmentation

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). Among 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. This procedure can become a challenging task if the target is, for example, blood vessels in the brain, which by nature involves complex shapes with many components with unpredicted turning directions. In the seeded region growing algorithm [13], we start from the selected seed point (the parent seed) as the current voxel and move to adjacent voxels (the child seeds) with feature values (such

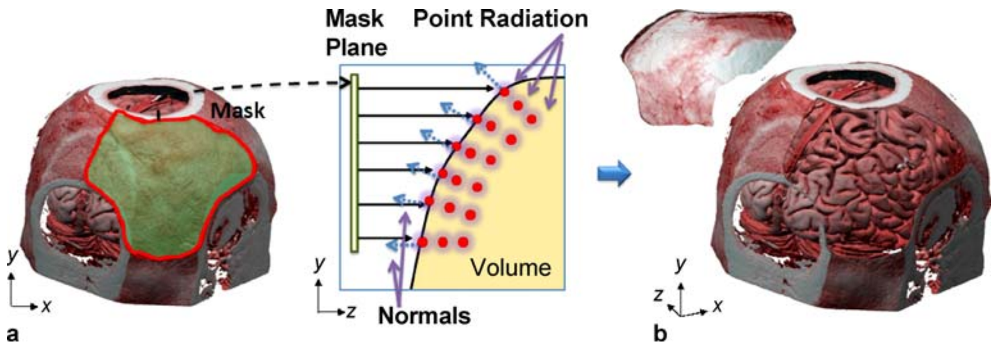

**Fig. 5. a** Peeling the skull with a free-form mask, parallel raycasting into the volume to find separate surface position and normal, and executing point radiation along the detected inverse normal directions. **b** Removing surface layers and pasting back the skull with a second rendering pass

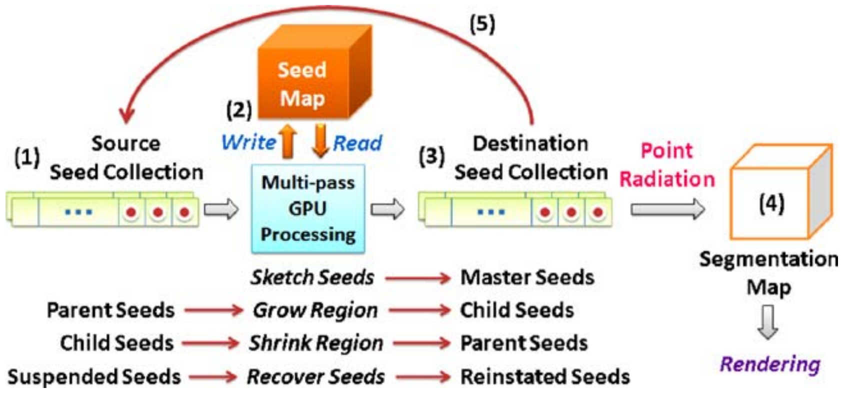
(5)
Seed Map
(2)
Write
Read
(1) Source Seed Collection
Multi-pass GPU Processing
(3) Destination Seed Collection
Point Radiation
(4)
Segmentation Map
Rendering
Sketch Seeds → Master Seeds
Parent Seeds → Grow Region → Child Seeds
Child Seeds → Shrink Region → Parent Seeds
Suspended Seeds → Recover Seeds → Reinstated Seeds

**Fig. 6.** Overview of our GPU-based segmentation framework. The processing initiates from the source seed collection (1), enters the processing unit, exchanges seed growing information with the seed volume (2), and outputs the result in the destination seed collection (3). The result is point radiated into the segmentation volume (4) for smooth rendering. The entire process is repeated (5) by swapping the source and destination seed collections

as intensity or gradient magnitude) close to the values in the current voxel. Standard CPU implementation of this method (e.g. breath first search) requires storing the seeds in a queue with sequential processing of the neighbors (adjacent voxels). For a $512^3$ volume, this amounts to 805 million iterations every frame. In the context of high-quality rendering with raycasting, the segmentation result (i.e. a volume) also needs to be transferred from CPU to GPU (per frame), further impeding interactive control of segmentation. Recent advancement of programmable hardware brings the potential of accelerating the region growing process. However, the architecture of parallel memory access on the GPU makes it hard to parallelize the seed growing process, which requires a single seed to finish with its exploration operations before another seed can proceed. We address this issue and provide a solution to plant and progress seeds entirely under a parallel execution environment.

## 5.1 GPU-based segmentation framework

We propose a GPU-based technique that uses four parallel functions: *Sketch Seeds*, *Grow Region*, *Shrink Region*, and *Recover Seeds*. Each function takes a set of input seeds to produce output seeds with a self-feedback loop (Fig. 6). *Sketch Seeds* converts user strokes into master seeds which are used to initiate the segmentation process. *Grow Region* expands the set of parent seeds to produce child seeds. This function also marks the voxels that could not be advanced (i.e. due to threshold constraints) as *suspended* seeds. *Shrink Region* and *Recover Seeds* support the dynamic control for thresholding. *Shrink Region* reverses the region growing steps. And, finally, *Recover Seeds* allows previously suspended seeds to be reinstated when threshold constraints are modified. These functions are evaluated in the GPU with multi-pass processing techniques and they need to have access to seeds' information. We store the seeds' information in vertex buffers and call it a *seed map*. It is used to control the segmentation process by mapping the seeding states (such as master, parent, child, suspended, or reinstated seeds) into respective voxels. Communication with the seed map is done using the geometry shader, with its unique capability of reading/writing 3D seed points into any location in the volume. Each function starts by feeding a source seed collection, exchanging seed growing information with the seed map, and writing the result into the destination seed collection. For the rendering purpose, the result of each function operation is also recorded in a 3D texture called a *segmentation map*. Direct use of the seed collection in the segmentation map produces aliasing artifacts due to its binary values (segmented/not segmented) (Fig. 7a).

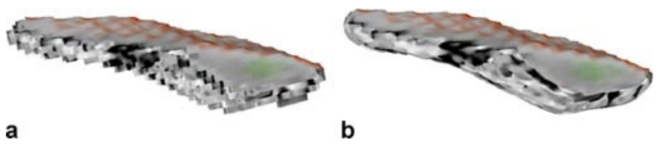

**Fig. 7a,b.** Partial segmentation of ventricles: **a** using a binary segmentation map without point radiation and **b** with point radiation

To obtain smooth and high-quality rendering of the segmentation result, we utilize point radiation to convert the collected seeds into blended regions in the segmented map (Fig. 7b). When the result is rendered, we sample the segmentation map using 0.5 as a natural candidate to distinguish between segmented and non-segmented regions. Finally, the entire simulation and rendering process is repeated by exchanging the source and destination seeds.

### 5.2 Sketch-based seeding

To allow quick multi-seeding directly in 3D, we use sketches to indicate seeds on the displayed volume. We map multiple input strokes onto surface points of the volume using raycasting. The stroke is first discretized and stored in a binary 2D sketch texture. Each pixel in the texture is then projected into the volume space to collect surface points based on rendering parameters. For each point on the surface, we find the closest voxel as the seed. However, as one or more of the detected surface points may be contained within the same voxel due to variance of volume resolution, it is necessary that we remove duplicated voxels to avoid multiple seed points at the same voxel location. To ensure that we collect unique seed points, we first store the collection of detected voxels in a list. Then, for each of the non-repeating voxels in the list, we unproject the respective voxel into a second list, storing the master seeds that map one-to-one to unique voxels in the volume.

### 5.3 Parallel region growing/shrinking

We enable region growing while exploring multiple seeds simultaneously. During the growing process, we start from a set of parent seeds and advance to a set of child seeds using the breath first search. In a number of region growing situations, different materials such as tissues and blood vessels have close intensity values and cannot be easily controlled by means of thresholding. The user initially sketches seeds on the blood vessels (Fig. 8a) but, during the growing process, a group of child seeds may start to spread to unrelated materials (Fig. 8c).

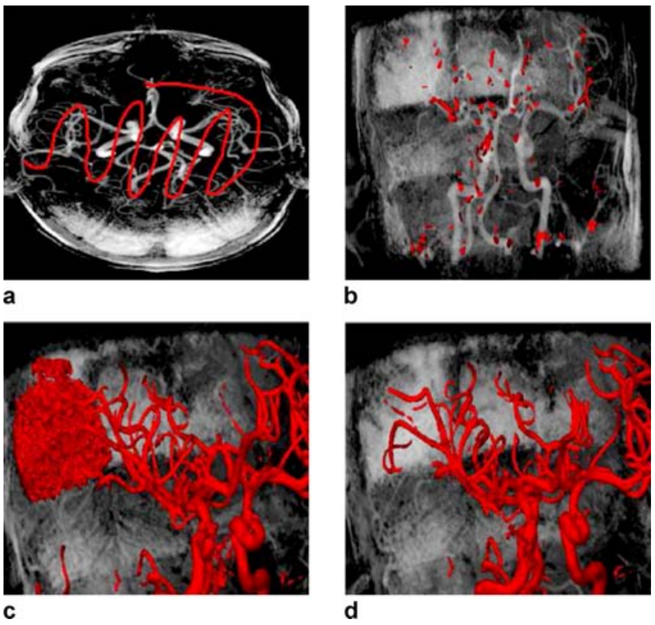
**Fig. 8a–d.** Segmentation of carotid and cerebral arteries from the angiogram dataset: **a** sketches on high-intensity data with X-ray and maximum intensity projection (MIP) rendering, **b** forward region grows in parallel, **c** results in undesired tissues, and **d** removes extra growth with region shrinking

In additional to forward growing, we introduce region shrinking that reverses the grown regions and rewinds the segmentation (Fig. 8d). This provides the user with an opportunity to undo partial region growing and cease the segmentation process. Afterwards, the user could resume the growing process by resketching seeds on the remaining part of the original region of interest.

To have the ability of shrinking, we need to track the parent/child relationship in the region growing. We use the seed map to save the parent information for each seed. With parallel execution, a conflict arises when two or more parent seeds are concurrently exploring the same voxel location as a child. Saving all of the possible parents at the same voxel location (child) requires a bigger field for the seed map's memory unit. To avoid this, we prevent the possibility of multiple parents by simply by over-writing the previous values when a new parent arises. This means that each seed (as a child) has the information of the most recent parent in the growing process.

To shrink the grown region, we first erase the segmented voxels occupied by the last set of child seeds and then we search their neighboring voxels to find potential parent seeds. The reversing process is based on the order in which the child seeds are processed; therefore, it is not guaranteed that we always return to the original set of parent seeds while it is a plausible solution. Another benefit of the reverse operation is that additional strokes can be drawn to indicate the area of removal. In this case, the sketched seeds become the source (i.e. the child seeds) of the reversing operation, allowing targeted region removal.

In addition to searching for the child seeds in the parallel region growing process, we mark the voxels that could not be advanced (i.e. due to threshold constraints) as suspended. The suspension information is recorded using the seed map. When thresholds are modified, we look up the suspended seeds from the seed map and allow them to reparticipate in the normal region growing/shrinking iterations.

## 6 Results and discussion

All the results were generated on an Intel Core2 Duo PC, with a GeForce 8800 GTX, 768 MB graphics card. We selected raw and presegmented volumetric medical datasets in both CT or MRI modality. Our system achieved fast computation rates in mask construction (80 ms), volume tools (50 fps on average), sketching seeds (60 ms), dynamic region growing (58 fps), and high-quality raycasting rendering (50 fps in X-ray mode and 60 fps in surface mode), with screen recording resolution of $1230 \times 870$.

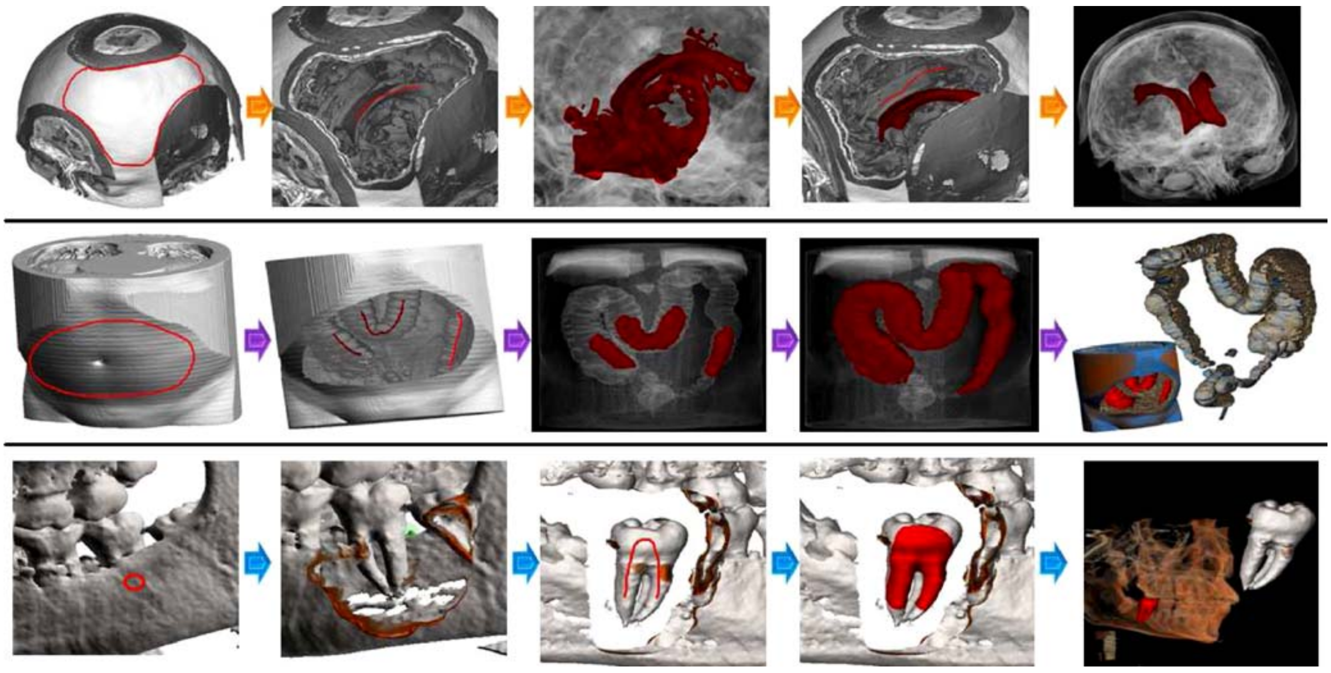

**Fig. 9.** (*Top*) Opening skull with the peeling tool, sketching seeds on the left ventricle, regions growing into surrounding material, reversed growing, sketching seeds on the right ventricle, and completing segmentation. (*Middle*) Opening abdominal wall with the peeling tool, placing multiple strokes on colon, region growing in parallel, covering entire colon, and pasting the segmented part in isolation. (*Bottom*) Lasering the skull with a circular mask, removing occluding bone, sketching seeds on the molar tooth, region growing, and pasting the segmented tooth with zoom

Note that the performance of point radiation scales with the amount of parallel (or stream) processors in the hardware. In our experiment of point radiation, we found that a kernel radius of 3 produced prominent results with the best performance for data ranging from $128^3$ to $512^3$.

In Fig. 1, we demonstrate peeling and segmentation of the super-brain dataset (MRI, $256 \times 256 \times 256$). Figure 9 (top) shows peeling of the skull and segmentation of the left and right lateral ventricles from the same super-brain dataset; (middle) shows peeling of the abdominal wall and segmentation of the colon from the abdomen dataset scanned in supine orientation (CT, $512 \times 512 \times 426$); (bottom) shows lasering, drilling, and segmentation of a molar tooth from the skull dataset (rotational C-arm X-ray scan of phantom of a human skull, $256 \times 256 \times 256$). Figure 8 illustrates seed sketching and segmentation of carotid and cerebral arteries from the angiography dataset (3T MRI time-of-flight of a human head, $256 \times 320 \times 128$).

## 7 Conclusion and future work

We proposed a GPU-based point radiation technique as a real-time manipulation primitive in the context of high-quality raycasting rendering. Instead of the traditional way of browsing from hundreds of cross-sectional slices or adapting 3D devices, we utilized the point-based strategy and introduced interactive volume tools for direct drilling, lasering, peeling, and cutting and pasting. With point radiation, we also developed an interactive region growing segmentation system and multiple seed planting with sketches. In handling the large medical volume datasets, we exploited programmable hardware and obtained a dramatic performance improvement (with local updates) compared to the 3D texture-based (global) approach in volume clipping and region growing segmentation.

Future improvements include performing user/clinical studies in specific medical domains to formally validate the usability of our framework. We plan to expand the set of volume tools, modeled after specific application domains such as virtual surgery. It would also be useful to extend the point radiation technique to create a non-spherical radiation in space and possibly generate a variety of other sculpting primitives and cutting tools. We also foresee adapting our framework with physically based volume deformation in domains where real-time, intuitive cutting and volume manipulation is essential (i.e virtual surgery). Moreover, our framework may provide a ground for a real-time implicit modeler, which requires further investigation.

## References

1. Avila, R.S., Sobierajski, L.M.: A haptic interaction method for volume visualization. In: Proceedings of IEEE Visualization, pp. 197–204. IEEE Computer Society Press, San Francisco, CA (1996)
2. Bloomenthal, J.: Introduction to Implicit Surfaces. Morgan Kaufmann Publishers Inc., San Francisco, CA (1997)
3. Chen, H.L.J., Samavati, F.F., Sousa, M.C., Mitchell, J.R.: Sketch-based volumetric seeded region growing. In: Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling 2006, pp. 123–129. Eurographics, Vienna (2006)
4. Cook, R.L.: Stochastic sampling in computer graphics. ACM Trans. Graph. **5**, 51–72 (1986)
5. Correa, C.D., Silver, D., Chen, M.: Feature aligned volume manipulation for illustration and visualization. IEEE Trans. Vis. Comput. Graph. **12**, 1069–1076 (2006)
6. Ferley, E., Cani, M.P., Gascuel, J.D.: Practical volumetric sculpting. Visual Comput. **16**, 469–480 (2000)
7. Galyean, T.A., Hughes, J.F.: Sculpting: an interactive volumetric modeling technique. In: Proceedings of SIGGRAPH ’91, pp. 267–274. ACM, New York, NY (1991)
8. Huff, R., Dietrich, C.A., Nedel, L.P., Freitas, C.M.D.S., Comba, J.L.D., Olabarriaga, S.D.: Erasing, digging and clipping in volumetric datasets with one or two hands. In: Proceedings of the ACM International Conference on Virtual Reality Continuum and its Applications, pp. 271–278. ACM, Hong Kong (2006)
9. Justice, R.K., Stokely, E.M.: 3-D segmentation of MR brain images using seeded region growing. In: 18th Annual International Conference of the IEEE Proceedings 1996, pp. 1083–1084. IEEE Computer Society, Amsterdam (1996)
10. McGuffin, M., Tancau, L., Balakrishnan, R.: Using deformations for browsing volumetric data. In: Proceedings of IEEE Visualization, pp. 401–408. IEEE Computer Society, Seattle, WA (2003)
11. Owada, S., Nielsen, F., Igarashi, T.: Volume catcher. In: Proceedings of the Symposium on Interactive 3D Graphics and Games ’05, pp. 111–116. ACM, Washington, District of Columbia (2005)
12. Pham, D.L., Xu, C., Prince, J.L.: A survey of current methods in medical image segmentation. Tech. Rep. JHU/ECE 99-01 (1999)
13. Rosenfeld, A., Kak, A.: Digital picture processing, vol. 2, pp. 138–145. Academic, New York (1982)
14. Schenke, S., Wuensche, B.C., Denzler, J.: GPU-based volume segmentation. In: Proceedings of Image and Vision Computing New Zealand ’05, pp. 171–176. University of Auckland, Dunedin (2005)
15. Sherbondy, A., Houston, M., Napel, S.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In: Proceedings of IEEE Visualization, pp. 171–176. IEEE Computer Society, Seattle, WA (2003)
16. Technical Brief: Microsoft DirectX 10: The Next-Generation Graphics API. NVIDIA Corporation (2006)
17. Tzeng, F.-Y., Lum, E.B., Ma, K.-L.: A novel interface for higher-dimensional classification of volume data. In: Proceedings of IEEE Visualization ’03, pp. 505–512. IEEE Computer Society, Seattle, WA (2003)
18. Wang, S.W., Kaufman, A.E.: Volume sculpting. In: Proceedings of the 1995 Symposium on Interactive 3D Graphics, pp. 151–156. ACM, Monterey, CA (1995)
19. Watt, A.: 3D Computer Graphics. Addison-Wesley, Harlow, Essex (1989)
20. Weiskopf, D., Engel, K., Ertl, T.: Interactive clipping techniques for texture-based volume visualization and volume shading. IEEE Trans. Vis. Comput. Graph. **9**(3), 298–312 (2003)
21. Westover, L.: Footprint evaluation for volume rendering. In: Proceedings of SIGGRAPH ’90, pp. 367–376. ACM, Dallas, TX (1990)
22. Woo, M., Neider, J., Davis, T., Shreiner, D.: OpenGL Programming Guide, third edn. Addison-Wesley, Harlow, Essex (1999)
23. Wyvill, B., Guy, A., Galin, E.: Extending the CSG tree – warping, blending and boolean operations in an implicit surface modeling system. Comput. Graph. Forum **18**, 149–158 (1999)

HUNG-LI JASON CHEN received his B.Sc. First Class Honours degree (2005) and M.Sc. degree (2008) in Computer Science from the University of Calgary, Canada. Jason’s research interests are computer graphics, interactive modeling/rendering, volume visualization, volume manipulation, and 3D medical imaging. His M.Sc. thesis describes interactive volume manipulation and high-quality visualization for medical data. He has worked for Autodesk and Electronic Arts Canada.

FARAMARZ F. SAMAVATI is an associate professor in the Department of Computer Science, University of Calgary. He received his Ph.D. degree from Sharif University of Technology in 1999. Prof. Samavati’s research interests are computer graphics, geometric modeling, 3D imaging, and interactive modeling. He has authored more than 50 research papers in subdivision surfaces, sketch based modeling, multiresolution and wavelets, surface modeling, and scientific visualization.

MARIO COSTA SOUSA is an associate professor in the Department of Computer Science, University of Calgary. He holds a Ph.D. degree in computer science from the University of Alberta. His research interests focus on illustrative visualization, non-photorealistic rendering, sketch-based interfaces and modeling, perceptual issues in illustration and visualization, interactive simulations, and real-time volume graphics.