Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Perspective Isosurface and Direct Volume Rendering for Virtual Endoscopy Applications

Henning Scharsach[1] Markus Hadwiger[1] André Neubauer[2] Stefan Wolfsberger[3] Katja Bühler[1]

[1]VRVis Research Center [2]Tiani Medgraph [3]Department of Neurosurgery, Medical University Vienna

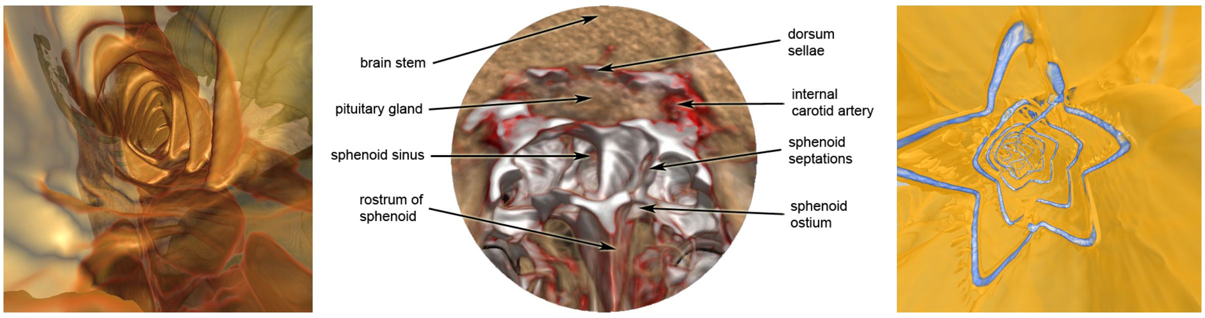

Figure 1: *The three main application areas of our system. Left: Virtual colonoscopy. The semi-transparent isosurface is blended with a direct volume ray-casting of the area behind. Center: Planning of pituitary surgery for tumor removal. Using direct volume rendering in combination with semi-transparent isosurfaces allows for visualization of otherwise occluded objects. Right: Virtual angioscopy of the aorta and a stent that supports it. Two different isovalues have to be used in this case.*

Abstract

Virtual endoscopy has proven to be a very powerful tool in endoscopic surgery. However, most virtual endoscopy systems are restricted to rendering isosurfaces or require segmentation in order to visualize additional objects behind occluding tissue. This paper presents a system for real-time perspective direct volume and isosurface rendering, which allows to simultaneously visualize both the interesting tissue and everything that is behind. Large volume data can be viewed seamlessly from inside or outside the volume without any pre-computation or segmentation. Our system uses a novel ray-casting pipeline for GPUs that has been optimized for the requirements of virtual endoscopy and also allows easy incorporation of auxiliary geometry, e.g., for displaying parts of the endoscopic device, pointers, or grid lines for orientation purposes. We present three main applications of this system and the underlying ray-casting algorithm. Although our ray-casting approach is of general applicability, we have specifically applied it to virtual colonoscopy, virtual angioscopy, and virtual pituitary surgery.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Color, shading, shadowing, and texture

1. Introduction

Virtual endoscopy has become a very powerful tool for aiding endoscopic surgery procedures, from initial surgeon training to preoperative planning and intraoperative support [BHH*04, Bar05]. For rendering, most systems for virtual endoscopy are focusing on rendering isosurfaces, e.g., to depict tissue walls surrounding the current position of the endoscope. The two main approaches for rendering isosurfaces are to extract explicit geometry [BHH*04, Bar05], e.g., with a variant of the marching cubes algorithm [LC87], or to use first-hit ray-casting in order to determine the intersection of viewing rays with the isosurface [NMHW02, NWF*]. While the first category is able to achieve very high performance, especially when graphics hardware is used for rendering the surface geometry [HMK*97, BS99], the fact that explicit geometry can usually not be extracted interactively hampers isovalue changes. Furthermore, high-resolution isosurface geometry is very memory intensive. On the other hand, ray-casting approaches for virtual endoscopy have the flexibility to change the isovalue interactively, but cannot make use of hardware-accelerated polygon rasterization and are usually implemented with custom CPU algorithms [NWF*].

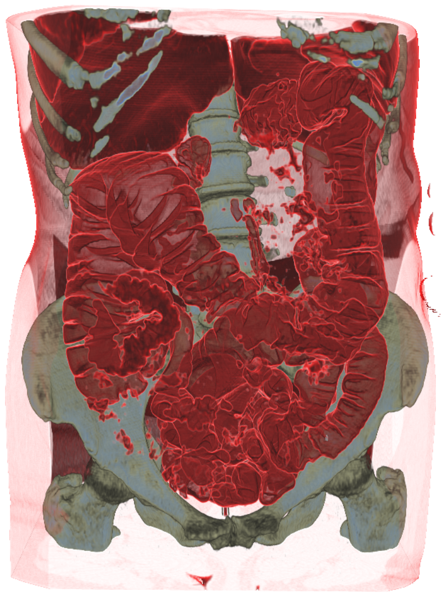

Figure 2: *Overview of a 512x512x478 colonoscopy data set rendered with our system. Figure 1 (left) and Figure 3 show inside views using the same data set and transfer function.*

Though isosurface-renderings can be enhanced to include more information about surface properties [NWF*], a large part of the information contained in a medical scan is still discarded during rendering because everything in front and especially most of the information behind the isosurface is not rendered. A crucial example is pituitary surgery for removing a tumor at the pituitary gland near the internal carotid artery, which is hidden behind a bone structure that must be punctured by the surgeon without damaging the artery behind it [NWF*] (see Figure 1 (center)). Most previous approaches have required segmentation in order to visualize these crucial occluded structures, which is a very time-consuming task and only allows to depict objects as isosurfaces of smoothed binary masks [NFW*04].

In contrast, direct volume rendering is only common for viewing volume data from the outside, because most approaches either use orthogonal projection or incur artifacts with perspective projection due to incorrect opacity correction [EHK*04]. Endoscopic views, however, require perspective projection with large fields of view and correct computation of the volume rendering integral, which necessitated special hardware architectures so far to achieve interactivity [MB01]. GPU-based ray-casting on the other hand can deliver this interactivity for either perspective or orthogonal projections [KW03], but requires far more effort to overcome the inherent problems. Most recent approaches build on single-pass ray-casting, where the entire volume is traversed in a single rendering pass using data-dependent looping in the hardware fragment shader [SSK*05, HSS*05, KSS*05]. In order to support large data sets, a bricked volume can be rendered in correct visibility order by performing ray-casting for each brick individually [HQK05]. This, however, incurs per-brick setup overhead in contrast to single-pass approaches.

Our system uses a novel ray-casting pipeline for GPUs supporting Shader Model 3.0 (e.g., NVIDIA GeForce 6800 or ATI Radeon X1800) that uses perspective projection and also allows easy incorporation of auxiliary geometry, e.g., for displaying parts of the endoscopic device, pointers, or grids for orientation. We seamlessly integrate direct volume rendering with isosurface rendering and achieve real-time performance for both fly-through and outside views without the need for pre-computation or segmentation, which greatly facilitates use by physicians that are working under enormous time pressure.

Although our system can essentially be used for virtual endoscopy in general, we have specifically applied it to three different applications: Virtual colonoscopy [HMK*97], virtual angioscopy [NMHW02], and virtual endonasal transsphenoidal pituitary surgery [NWF*]. We have integrated our system into a commercial medical workstation software, and the application to neurosurgery is already in regular clinical use for operation planning at the department of neurosurgery at the Medical University Vienna.

2. Applications

Virtual endoscopy has proven to be a useful tool for a number of different applications, including pre-operative planning, diagnostic purposes, teaching and practicing with endoscopic tools and even the emerging field of aided intra-operative navigation. The aim of all of these systems - especially the latter one - is to aid medical doctors with a computer generated view of a certain position and orientation of the endoscope that resembles the real endoscopic view as closely as possible while at the same time supplying additional information that would not be visible otherwise. This additional information may include waypoints that prevent deviation from the optimal path, tissue right behind visible structures, the endoscope and attached endoscopic tools themselves, means to measure certain structures to get a better impression of the surrounding tissue and emphasizing important parts like nerves or bigger blood vessels that must not be damaged by any means.

To fulfill all these requirements, a suitable renderer has to be able to visualize many semi-transparent structures at the same time while always keeping the focus on the main object of interest: The surrounding walls of the blood vessel, colon or other structure that the endoscope is travelling through. What makes this task even more difficult is the fact that we are facing three different types of objects here that all have to be visualized accordingly: Isosurfaces like the walls have to be extracted and lit in a way that gives a good overview about surface properties and makes small deformations easily detectable. Regions of interest like tissue behind

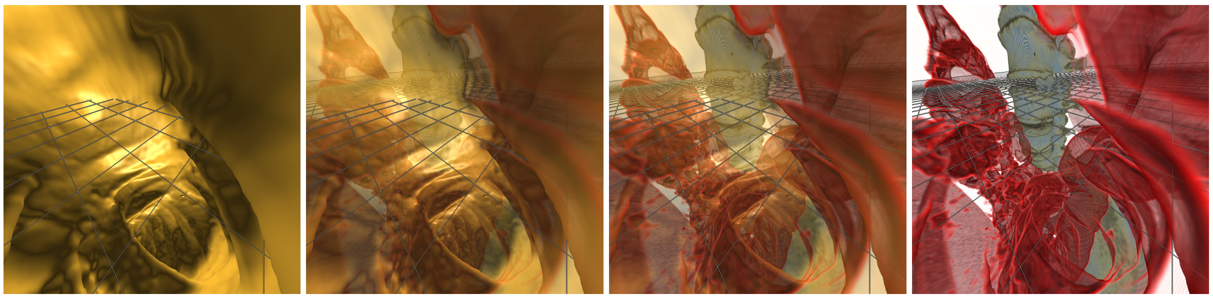

Figure 3: *Changing the isosurface opacity allows to focus more on the foreground rendered with isosurfacing (e.g., the colon) or the background rendered with direct volume rendering. Correctly integrating polygonal geometry such as the grid shown here facilitates spatial orientation. Isosurface opacity also influences rendering performance, as illustrated in Tables 1 and 2.*

walls, important nerves or organs should be visualized using direct volume rendering, giving the user the possibility to distinguish different parts of tissue by assigning a suitable transfer function. Finally, endoscopic tools, grids and pointers, which will aid the surgeon in orientation and estimation of magnitude, are made up of lines and polygons and should be visualized using the normal OpenGL pipeline.

Bringing these visualization techniques together while still preserving interactive framerates was the primary goal of our system, and in the following we will present three fields of application where this leads to significant improvements over previous systems in both expressiveness and applicability.

2.1. Virtual Colonoscopy

A regular colonoscopy enables doctors to examine the last portion of the gastrointestinal tract and look for problem areas such as inflamed tissue, abnormal growths and ulcers. The main purpose is to detect early signs of cancer in the colon and rectum and further analyze or remove them, if possible. Virtual colonoscopy can greatly speed up this process by enabling doctors to check for any kind of abnormality beforehand, and only perform a real colonoscopy when polyps have to be removed or samples have to be taken.

Unfortunately, most virtual colonoscopy systems suffer from the limited expressiveness of the common isosurface extraction, which makes it very difficult to identify all abnormalities and may even lead to false diagnoses because important features where simply overlooked. With doctors hesitating to adopt a technology that has not really proven reliability yet, not many of these systems have found their way into clinical practice. Another shortcoming is that the visualization often lacks resemblance to the real images, making it very difficult for doctors to estimate tissue properties and recognize certain parts later on.

The solution we propose is to combine isosurface extraction for visualization of the colon with a DVR of the regions of interest behind the intestinal wall, thus gaining additional information about the respective tissue. This not only leads to a much more expressive image, but also provides important clues about the position and orientation within the colon, because the whole gastrointestinal tract can be seen at any time. In the case of an intra-operative system, this makes it extremely simple to quickly find areas of interest again, thus greatly speeding up the surgery.

For diagnostic purposes, an automatic path can be calculated which provides a convenient and quick fly-through from the beginning of the colon to the rectum, giving an overview over the large intestine in a matter of minutes.

2.2. Virtual Angioscopy

Virtual angioscopy (virtual endoscopy inside blood vessels) is primarily used for detecting stenoses and calcifications in blood vessels. With many blood vessels being too narrow for a normal endoscope, virtual angioscopy is in many cases the only alternative to the tedious process of examining 2D-slices of a CT scan. Besides the small size of the structures, the specific nature of an angiography requires a slight modification of the rendering pipeline: Because some kind of contrast medium is injected to identify important vessels more easily, density values inside the vessels are higher than those outside. On the other hand, calcifications inside the vessel have an even higher density value, which means that we face two different isosurfaces: One at a certain minimum threshold (i.e., the outer walls) and one at a maximum threshold above that of the contrast medium, which in most cases are calcifications but can also be structures such as stents [BDV*97]. An example that has been generated with our system is illustrated in Figure 1 (right). In this case, the first isosurface corresponds to the stent, and the second isosurface corresponds to the aortic wall. Everything outside one of the isosurfaces (i.e. below the minimum or above the maximum threshold) will be rendered with DVR again, which supplies additional information about the tissue density thus aiding doctors in detecting calcifications and again facilitates estimating the absolute position and orientation through the visualization of certain body landmarks.

A further interesting application is the implantation of stents, which are small tubular prostheses that are inserted into an artery via an endovascular procedure and are mainly used to enlarge a stenosis (i.e. a local narrowing of the arterial lumen). Like calcifications, stents have a density threshold well above that of the contrast medium, which requires the rendering of two different isosurfaces again. When implanting a stent, the position in respect to other landmarks of the body (e.g. heart, lung or bones) is especially important, which is achieved through a whole DVR behind the semi-transparent isosurfaces again.

2.3. Endonasal Transsphenoidal Pituitary Surgery

Endonasal transsphenoidal pituitary surgery is a minimally invasive endoscopic procedure, mainly applied to remove various kinds of pituitary tumors. Virtual endoscopy can aid medical doctors by simulating this challenging surgery beforehand and planning the approach and ideal target position of the endoscopic intervention. Especially in the case of an intra-operative environment, the system should provide visual feedback about important nerves, blood vessels and other significant landmarks that should not be damaged, thus assisting the doctor in finding the optimal path.

In order to do this, those important landmarks have to be identified and should be visualized throughout the whole process. To avoid the necessity of pre-segmenting every single one of those objects, the whole head should be rendered with a semi-transparent DVR. If a suitable transfer function is selected, this not only warns the doctor whenever an important object is nearby that requires special attention, but also avoids deviation of the optimal path by always visualizing the main object of interest (e.g. the tumor).

Visualizing the endoscopic tools is another important aspect: Especially when planning the optimal path and ideal target position, the size and proportions of these tools with respect to the surrounding tissue is crucial. Because a volumetric approach would raise some serious issues (rotating the tools, limited resolution etc.) and probably not resemble the real appearance closely enough, a polygonal representation of the tools should be merged with the volumetric scene. Of course, intersections have to be calculated and displayed correctly, otherwise the perceived information about available space and proportions may be misleading.

Furthermore, when encountering objects of interest like small passages or even the tumor itself, there should be a way to measure this object or at least get a first estimation of its size. Other auxiliary graphical elements like grids and pointers can assist the user here, again necessitating a rendering pipeline that can deal with both polygonal and volumetric objects.

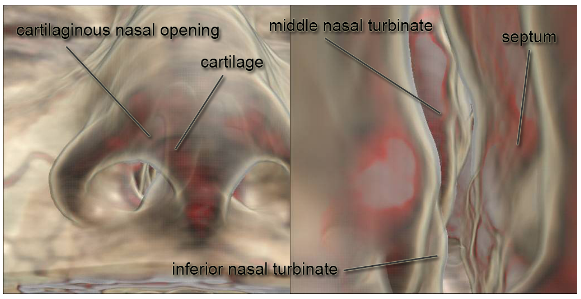

Figure 4: *In endonasal pituitary surgery the endoscope enters through the nose and is advanced to the sphenoid sinus and the pituitary gland behind it (see Figure 1 (center)).*

3. Hybrid Ray-Casting for Virtual Endoscopy

Choosing the right volume rendering mode for a particular application is crucial for extracting the useful and important information that the user wants to see. In the case of virtual endoscopy, using only isosurface rendering discards a lot of information behind the surface that can provide additional insight into the properties of the underlying tissue, as well as crucial information about occluded structures. In order to alleviate this problem, our system combines isosurface and direct volume ray-casting in a single rendering. We render a shaded semi-transparent isosurface in front, and perform unshaded direct volume rendering behind it. This rendering mode achieves our goals and also avoids visual confusion of isosurface shading and volume shading. The focus is still on the isosurface, but important background information is available at any time. Shading the isosurface is important for shape perception. Direct volume rendering then provides additional information, such as the tissue structure close to the isosurface, as well as depicting background objects farther behind. This leads to very expressive images and allows for considerable flexibility. The basic algorithm is very simple and leverages standard polygon rasterization for ray setup, and a very short fragment shader loop for actual ray-casting. In addition to the volume, intersecting polygonal geometry is integrated seamlessly.

3.1. Algorithm Overview

The ray-casting pipeline of our system combines object-order and image-order stages in order to find a balance between the two and leverage the parallel processing of modern GPUs. For culling of irrelevant subvolumes, a regular grid of min-max values for bricks of size 8^3 is stored along with the volume. Ray-casting itself is performed in a single rendering pass in order to avoid the setup overhead of casting each brick separately [HQK05]. The first step of the algorithm culls bricks on the CPU and generates two separate bit arrays that determine whether a brick is *active* or *inactive*.

The first bit array contains the state of bricks with respect to the isosurface. A brick is active when it intersects the isosurface. The second bit array contains the state of bricks with respect to the transfer function. A brick is active when it is not completely transparent.

In the object-order stage on the GPU, these two bit arrays are used to rasterize brick boundary faces in several rendering passes. The result of these rendering passes are two images that drive the subsequent ray-casting stage. The first image, the *ray start position image*, contains the volume coordinate positions where ray-casting should start for each pixel. Coordinates are stored in the RGB components, and the alpha component is one when a ray should be started, and zero when no ray should be started. The second image, the *ray length image* contains the direction vectors for ray-casting in the RGB components and the length of each ray in the alpha component. Note that the direction vectors could easily be computed in the fragment shader from the camera position and the ray start positions as well. However, the ray length must be rendered into an image that is separate from the ray start positions due to read-write dependencies, which can then also be used for storing the direction vectors that are needed for ray length computation anyway.

The main steps of our ray-casting approach for each pixel are:

1. Compute the initial ray start position on the near clipping plane of the current viewport. When the start position is in an inactive brick with respect to the isosurface, calculate the nearest intersection point with the boundary faces of active isosurface bricks, in order to skip empty space. The result is stored in the *ray start position image*.
2. Compute the ray length until the last intersection point with boundary faces of bricks that are active either due to the isosurface or the transfer function or both. The result is stored in the *ray length image*.
3. Optionally render opaque polygonal geometry and overwrite the ray length image where the distance between the ray start position and the geometry position is less than the stored ray length.
4. Cast from the start position stored in the *ray start position image* along the direction vector until the accumulated opacity reaches a specified threshold (*early ray termination*) or the ray length given by the *ray length image* is exceeded. The result of ray-casting is stored in a separate compositing buffer.
5. Blend the ray-casting compositing buffer on top of the polygonal geometry.

The two main acceleration schemes exploited here are *empty space skipping* and *early ray termination*. For the former, view-independent culling of bricks and rasterization of their boundary faces are employed (Section 3.2), whereas the latter is handled during ray-casting (Section 3.3).

3.2. Culling and Brick Boundary Rasterization

Because we are using hybrid isosurface and direct volume rendering, culling has to determine two different sets of active/inactive states for all bricks, which are stored in separate bit arrays. Each brick is either inactive, active with respect to the isosurface, active with respect to the transfer function, or active with respect to both. In order to determine ray start positions and ray lengths, we employ rasterization of the boundary faces between active and inactive bricks, which is illustrated in Figure 5. To handle brick culling efficiently, the minimum and maximum voxel values of each brick are stored along with the volume, which are compared at run-time with the isovalue and the transfer function, respectively. A brick can be safely discarded when the opacity is always zero between those two values, which can be determined very quickly using summed area tables [GBKG04].

Rasterizing the boundary faces between active and inactive bricks results in object-order empty space skipping. It prunes the rays used in the ray-casting pass and implicitly excludes most inactive bricks. Note, however, that our approach does not exclude all empty space from ray-casting, which can be seen for ray r_3 in Figure 5 (left). This is a trade-off that enables ray-casting without any per-brick setup overhead and works extremely well in practice, which is also illustrated by the performance figures in Section 4. The border between active and inactive bricks defines a surface that can be rendered as standard OpenGL geometry with the corresponding position in volume coordinates encoded in the RGB colors. Corresponding to the two bit arrays of active bricks that results from culling, two sets of boundary faces must be considered. All vertices of brick bounding geometry are constantly kept in video memory. Only two additional index arrays referencing the vertices of active boundary faces have to be updated every time the isovalue or the transfer function changes. As long as the near clipping plane does not intersect the bounding geometry, rays can always be started at the brick boundary front faces. However, if such an in-

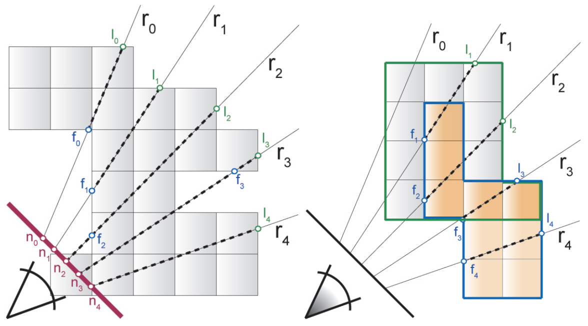

Figure 5: *Determining ray start positions and ray lengths using rasterization of brick boundary faces. Left: The basic case described in Section 3.2.1. Right: Extended cases for endoscopy rendering, which are described in Section 3.2.2.*

tersection occurs, it will produce holes in the front-facing geometry, which results in some rays not being started at all, and others started at incorrect positions. Figure 6 illustrates this problem. In an endoscopic view, we constantly face this situation, so rays typically need to be started at the near clipping plane, which is shown in Figure 5 in the case of points n_2-n_4. To avoid casting through empty space, rays should not be started at the near clipping plane if the starting position is in an inactive brick but at the next intersection with active boundary faces, such as rays r_0 and r_1 in Figure 5. These rays are started at f_0 and f_1, instead of being starting at n_0 and n_1. We achieve this by drawing the near clipping plane first and the front faces afterwards, which ensures that whenever there are no front faces to start from, the position of the near clipping plane will be taken. However, since the non-convex bounding geometry often leads to multiple front faces for a single pixel, the next front face is used when the first front face is clipped, which results in incorrect ray start positions. The solution is to detect when a ray intersects a back face before the first front face that is not clipped.

3.2.1. The Basic Case

When only one bit array of active bricks is used, e.g., direct volume rendering is used without isosurfacing, the basic steps to obtain the *ray start position image* are as follows:

1. Disable depth buffering. Rasterize the entire near clipping plane into the color buffer. Set the alpha channel to zero everywhere.
2. Enable depth buffering. Disable writing to the RGB components of the color buffer. Rasterize the *nearest back faces* of all active bricks into the depth buffer, e.g., by using a depth test of `GL_LESS`. Set the alpha channel to one where fragments are generated.
3. Enable writing to the RGB components of the color buffer. Rasterize the *nearest front faces* of all active bricks, e.g., by once again using a depth test of `GL_LESS`. Set the alpha channel to one where fragments are generated.

This ensures that all possible combinations shown in Figure 5 (left) are handled correctly. Rasterizing the nearest front faces makes sure that all near plane positions in inactive bricks will be overwritten by start positions on active

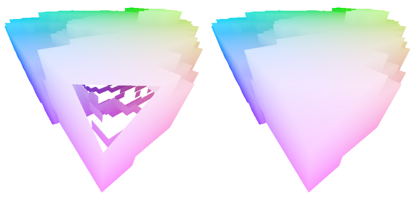

Figure 6: *Holes resulting from near clipping plane intersection (left) must be filled with valid starting positions (right).*

bricks that are farther away (rays r_0 and r_1). Rasterizing the nearest back faces before the front faces ensures that near plane positions inside active blocks will not be overwritten by front faces that are farther away (rays r_2 and r_3). Brick geometry that is nearer than the near clipping plane is automatically clipped by the graphics subsystem. After that, the *ray length image* can be computed, which first of all means finding the last intersection points of rays with the bounding geometry. The basic steps are:

1. Rasterize the *farthest back faces*, e.g., by using a depth test of `GL_GREATER`.
2. During this rasterization, sample the ray start position image and subtract it from the back positions obtained via rasterization of the back faces. This yields the ray vectors and the ray lengths from start to end position.
3. Multiply all ray lengths with the alpha channel of the ray start position image (which is either 1 or 0).

These steps can all be performed in the same fragment shader. Drawing the back faces of the bounding geometry results in the last intersection points of rays and active brick geometry, which are denoted as l_i in Figure 5. Subtracting end positions from start positions yields the ray vectors, which can then be normalized and stored in the RGB components of the *ray length image* together with the ray lengths in the alpha channel. Note that the alpha channel of the ray length image has consistently be set to zero where a ray should not be started at all, which is exploited in the ray-casting pass (Section 3.3).

3.2.2. Combining Isosurfacing and DVR

The basic case described in the previous section must be extended when isosurface and direct volume rendering are combined:

1. Rasterization passes for the ray start position image must use the bit array containing the state of bricks with respect to the isosurface. The transfer function is disregarded.
2. Rasterization for generation of the ray length image must treat all bricks as active that are active with respect to either the isosurface or the transfer function or with respect to both.

Figure 5 (right) illustrates all possible cases. Ray r_0 is never cast because it never intersects isosurface bricks. Both ray r_1 and ray r_2 start at isosurface bricks and terminate at transfer function bricks that are inactive with respect to the isosurface. Ray r_3 starts at an isosurface brick but terminates at a transfer function brick that is also active with respect to the isosurface. Ray r_4 starts and ends at isosurface bricks that are inactive with respect to the transfer function.

3.3. Ray-Casting

Our system employs a ray-casting fragment shader that performs the entire casting step for both the isosurface and the DVR part behind it in a single rendering pass. Therefore, the GPU is required to support data-dependent looping and branching in the fragment shader, e.g., an NVIDIA GeForce 6 or an ATI Radeon X1800. The shader is essentially comprised of two successive ray-casting loops, which perform first-hit ray-casting followed by DVR ray-casting. The first loop in the fragment shader starts at the position given by the *ray start position image* and traverses the ray until it finds an intersection with the isosurface. After an intersection has been detected, the actual intersection position is refined using iterative bisection with a fixed number of four iterations [HSS*05]. Then, the gradient at the intersection position is computed using central differences and the isosurface is shaded using the standard Blinn-Phong model. The shaded result is weighted according to the opacity of the isosurface, which can be set arbitrarily via a fragment shader parameter. Figure 3 illustrates changing isosurface opacity. After the isosurfacing part of the shader, DVR ray-casting continues with the initial alpha set to the opacity of the isosurface and performs sampling and compositing until a specified alpha threshold is exceeded. Checking against this threshold results in *early ray termination*. That is, the DVR ray-casting loop is terminated as soon as all subsequent samples cannot contribute to the final pixel color anymore. Note that early ray termination naturally depends significantly on the constant opacity of the isosurface. Tables 1 and 2 clearly show that the frame rate increases considerably with increasing isosurface opacity. In order to avoid visual interference with the shaded isosurface that is in front, no further shading is performed in the DVR compositing loop. Naturally, this also increases performance accordingly. Note that the ray-casting shader only performs ray-casting for pixels with a ray length greater than zero, which also excludes rays from processing that do not intersect active bricks at all as described in Section 3.2.1.

3.4. Geometry Intersection

Many applications for virtual endoscopy require both volumetric and polygonal data to be present in the same scene.

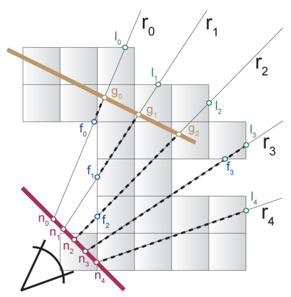

Figure 7: *When rays intersect opaque polygonal geometry, they are terminated immediately. This is achieved by modifying the ray length image accordingly.*

Naturally, intersections of the volume and geometry have to achieve a correct visibility order, and in many cases looking at the intersections of the geometry and the isosurface is the reason for rendering geometry in the first place. A very powerful use of combining geometry with volume rendering is to display grid lines for orientation purposes, which is illustrated in Figure 3. We use a planar grid consisting of lines, which is a very simple but powerful means for assessing spatial location. This grid plane can be translated in the orthogonal direction and can also be rotated arbitrarily. Also, parts that do not contribute to the final image because they are occluded by geometry should not perform ray-casting at all. An easy way to achieve this is to terminate rays once they hit a polygonal object by modifying the ray length image accordingly. This is illustrated in Figure 7. Of course, ray lengths should only be modified if a polygonal object is closer to the view point than the initial ray length. This problem can again be solved by using the depth test and extending the algorithm described in Section 3.2.1, leading to the complete algorithm outlined in Section 3.1.

After rendering the back faces of active/inactive brick boundaries with their respective depth values (and depth test set to `GL_GREATER`), the intersecting geometry is rendered to the same buffer, with the corresponding volume coordinates encoded in the color channel. With the depth test reversed to `GL_LESS`, only those parts will be drawn that are closer to the view point than the initial ray lengths. This approach modifies ray-casting such that it results in an image that looks as if it was intersected with an invisible object. Blending this image on top of the actual geometry in the last pass of the algorithm results in a rendering with correct intersections and visibility order.

4. Rendering Performance

Tables 1 and 2 give rendering performance results of our system. Setting the transfer function to a simple alpha ramp illustrates the effectiveness of early ray termination (Table 1). Setting the isosurface to full opacity will result in immediate ray termination when the isosurface is hit, which yields performance figures similar to rendering isosurfaces only. The less opacity the isosurface adds to the image, the longer the ray has to travel to accumulate full opacity in the direct volume rendering stage. Choosing a more complex transfer

alpha		**isosurface**	**opacity**	
ramp TF	**100%**	**80%**	**50%**	**0%**
Minimum	40.3 fps	32.4 fps	29.5 fps	28.3 fps
Maximum	64.7 fps	54.8 fps	48.6 fps	44.4 fps
Average	58.2 fps	46.3 fps	41.2 fps	37.5 fps

Table 1: *Performance comparison of different isosurface opacities for the colonoscopy dataset with a simple ramp as transfer function. Measured for a 512x512 viewport on a GeForce 7800.*

complex		**isosurface**	**opacity**	
TF	**100%**	**80%**	**50%**	**0%**
Minimum	40.3 fps	11.1 fps	10.0 fps	9.4 fps
Maximum	64.7 fps	18.6 fps	16.7 fps	15.9 fps
Average	58.2 fps	16.8 fps	14.6 fps	13.2 fps

Table 2: *Performance comparison of different isosurface opacities for the colonoscopy dataset with a complex transfer function that prevents early ray termination most of the time. Measured for a 512x512 viewport on a GeForce 7800.*

function with low alpha values results in performance reduction because in this case early ray termination is ineffective for many rays (Table 2).

5. Conclusions

We have described an effective system for virtual endoscopy that uses GPU-based ray-casting in order to achieve real-time performance. The combination of isosurface and direct volume ray-casting has proven to be very useful during endoscopic planning in order to inspect structures that would otherwise be hidden behind the isosurface.

Using the computational power of today's GPUs in a hardware-based approach as described here, simultaneous isosurface and direct volume ray-casting is feasible at interactive frame rates, which has traditionally been substituted by pure isosurfacing or requiring segmentation. Merging this capability with a flexible rendering pipeline that can handle both volumetric and polygonal data, we have presented a system that is capable of meeting the visualization demands of medical doctors in diagnostic as well as intra-operative environments. The effectiveness and applicability of this virtual endoscopy system has been shown in three different fields of endoscopic procedures: virtual colonoscopy, virtual angioscopy and pituitary surgery. For neurosurgery, our system is already in clinical use, and we will investigate the clinical applicability of the other applications we have presented in the future.

Acknowledgments. The authors would like to thank Christian Sigg for important input. The VRVis research center is funded in part by the Austrian Kplus project. The medical data sets are courtesy of Tiani MedGraph.

References

[Bar05] BARTZ D.: Virtual endoscopy in research and clinical practice. In *Computer Graphics Forum* (2005), p. 24(1).

[BDV*97] BEIER J., DIEBOLD T., VEHSE H., BIAMINO G., FLECK E., FELIX R.: Virtual endoscopy in the assessment of implanted aortic stents. In *Proc. of Computer Assisted Radiology* (1997), pp. 183–188.

[BHH*04] BARTZ D., HARDENBERGH J., HAUTH M., MUELLER K., WACKER M., WU Y.: Advanced virtual medicine: Techniques and applications for virtual endoscopy and soft-tissue-simulation. In *IEEE Visualization 2004 Tutorial Notes* (2004).

[BS99] BARTZ D., SKALEJ M.: VIVENDI - a virtual ventricle endoscopy system for virtual medicine. In *Data Visualization (Proc. of Symposium on Visualization)* (1999), pp. 155–166.

[EHK*04] ENGEL K., HADWIGER M., KNISS J., LEFOHN A., REZK-SALAMA C., WEISKOPF D.: *Real-Time Volume Graphics*. Course Notes SIGGRAPH 2004, 2004.

[GBKG04] GRIMM S., BRUCKNER S., KANITSAR A., GRÖLLER E.: Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In *Proceedings IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics* (2004), pp. 1–8.

[HMK*97] HONG L., MURAKI S., KAUFMAN A., BARTZ D., HE T.: Virtual voyage: Interactive navigation in the human colon. In *Proceedings of SIGGRAPH'97* (1997), pp. 27–34.

[HQK05] HONG W., QIU F., KAUFMAN A.: Gpu-based object-order ray-casting for large datasets. In *Proceedings of Volume Graphics 2005* (2005).

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proceedings of Eurographics 2005* (2005), pp. 303–312.

[KSS*05] KLEIN T., STRENGERT M., STEGMAIER S., , ERTL T.: Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware. In *Proceedings of IEEE Visualization 2005* (2005), pp. 123–230.

[KW03] KRÜGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proc. of IEEE Visualization 2003* (2003), pp. 287–292.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. of SIGGRAPH '87* (1987), pp. 163–169.

[MB01] MEISSNER M., BARTZ D.: Translucent and opaque direct volume rendering for virtual endoscopy applications. In *Proceedings of Volume Graphics 2001* (2001).

[NFW*04] NEUBAUER A., FORSTER M., WEGENKITTL R., MROZ L., BÜHLER K.: Efficient display of background objects for virtual endoscopy using flexible first-hit ray casting. In *Proceedings of VisSym 2004* (2004), pp. 301–310.

[NMHW02] NEUBAUER A., MROZ L., HAUSER H., WEGENKITTL R.: Cell-based first-hit ray casting. In *Proceedings of VisSym 2002* (2002), pp. 77–ff.

[NWF*] NEUBAUER A., WOLFSBERGER S., FORSTER M., MROZ L., WEGENKITTL R., BÜHLER K.: Advanced virtual endoscopic pituitary surgery. *IEEE Transactions on Visualization and Computer Graphics 11*, 5, 497–507.

[NWF*04] NEUBAUER A., WOLFSBERGER S., FORSTER M., MROZ L., WEGENKITTL R., BÜHLER K.: STEPS - an application for simulation of transsphenoidal endonasal pituitary surgery. In *Proc. of IEEE Visualization* (2004).

[SSK*05] STEGMAIER S., STRENGERT M., KLEIN T., , ERTL T.: A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Proceedings of Volume Graphics 2005* (2005), pp. 187–195.

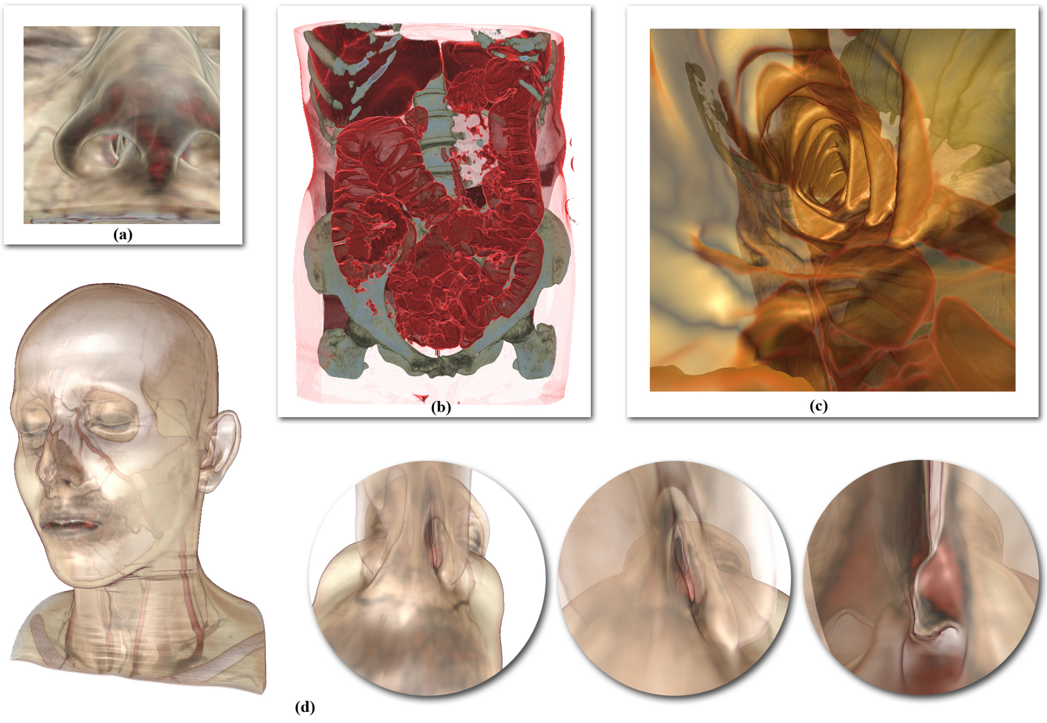

Figure 8: *(a) A view from a planning session for virtual pituitary surgery. (b) and (c) illustrate virtual colonoscopy with an overview and an inside view, respectively. Both images use the same transfer function, and in (c) also the surface of the colon is shaded and rendered semi-transparently. (d) During virtual pituitary surgery an endoscope is inserted through the nose.*