A Practical Evaluation of Popular Volume Rendering Algorithms

Michael Meißner[1] Jian Huang[2] Dirk Bartz[1]
Klaus Mueller[3] Roger Crawfis[2]

[1] Department of Computer Science, University of Tübingen, Germany
[2] Department of Computer Science, The Ohio State University, Columbus, USA
[3] Department of Computer Science, State University of New York at Stony Brook, USA

ABSTRACT

This paper evaluates and compares four volume rendering algorithms that have become rather popular for rendering datasets described on uniform rectilinear grids: raycasting, splatting, shear-warp, and hardware-assisted 3D texture-mapping. In order to assess both the strengths and the weaknesses of these algorithms in a wide variety of scenarios, a set of real-life benchmark datasets with different characteristics was carefully selected. In the rendering, all algorithm-independent image synthesis parameters, such as viewing matrix, transfer functions, and optical model, were kept constant to enable a fair comparison of the rendering results. Both image quality and computational complexity were evaluated and compared, with the aim of providing both researchers and practitioners with guidelines on which algorithm is most suited in which scenario. Our analysis also indicates the current weaknesses in each algorithm's pipeline, and possible solutions to these as well as pointers for future research are offered.

1 INTRODUCTION

Volume visualization methods display volumetric datasets, represented as sample points, on computer screens, head-mounted displays, group-immersive projection media (i.e., virtual workbench, CAVE), and large projection screens (i.e., PowerWall). Two avenues can be taken to achieve this:

- The volumetric data are first converted into a set of polygonal isosurfaces (i.e., via Marching Cubes [23]) and subsequently rendered with polygon rendering hardware. This is referred to as indirect volume rendering (IVR).
- The volumetric data are directly rendered without the intermediate conversion step. This is referred to as direct volume rendering (DVR) [10][11][35][42].

The former assumes (i) that a set of extractable iso-surfaces exists, and (ii) that with the infinitely thin surface the polygon mesh models the true object structures at reasonable fidelity. Neither is always the case, and as illustrative examples may serve: (i) amorphous cloud-like phenomena, (ii) smoothly varying flow fields, or (iii) structures of varying depth (and varying transparencies of an isosurface) that attenuate traversing light corresponding to the material thickness. But even if both of these assumptions are met, the complexity of the extracted polygonal mesh can overwhelm the capabilities of the polygon subsystem, and a direct volume rendering may prove more efficient [33], especially when the object is complex or large, or when the isosurface is interactively varied and the repeated polygon extraction overhead must be figured into the rendering cost [3].

In this paper, we concern ourselves solely with the direct volume rendering approach and with volumetric datasets that are described on cubic and uniform rectilinear grids, i.e., grids with anisotropic spacing along the three grid axes. Datasets of this nature are commonly obtained by means of volumetric scanning devices, such as MRI, CT, and confocal microscopy, by simulations, or by resampling of formerly irregularly gridded scientific datasets. Four techniques have emerged as particularly popular in this arena: Raycasting [41][19], Splatting [46], Shear-warp [18], and 3D texture-mapping hardware-based approaches [6].

Over the years, many researchers have worked independently on refining these four methods, and due to this multifarious effort, all methods have now reached a high level of maturity. Most of this development, however, has evolved along separate paths (although some fundamental scientific progress has benefited all methods - such as advances in filter design [7][4][28] or efficient shading [36][44]). A number of frequently used and publicly available datasets exists (e.g., the UNC CT / MRI heads or the CT lobster), however, due to the large number of parameters that were not controlled across presented research, it has so far been difficult to assess the benefits and shortcomings of each method in a decisive manner. The generally uncontrolled parameters include (apart from hardware architecture, available cache, and CPU clock speed): shading model, viewing parameters, scene illumination, transfer functions, optical model, image sizes, and magnification factors. Further, so far, no common set of evaluation criteria exists that enables fair comparisons of proposed methods with existing ones. Addressing this problem, it is one of the goals of this paper to establish an appropriate set of benchmark scenarios and evaluation criteria that can be used to compare both newly evolving and improved DVR methods with existing ones. The need for a clear specification of rendering parameters before any image comparison takes place was recognized by Williams and Uselton in [47], where a detailed specification of these influential rendering parameters is given. In the same work, metrics to identify the amount of noise, bias, and structural artifacts are also introduced. Some work in comparing different rendering methods has been conducted by Bartz et. al. [3] who compared DVR using raycasting with IVR using marching cubes for iso-surface extraction, while Tiede et. al. [39] have compared gradient filters for raycasting and marching cubes. Finally, Kwansik et.al [16] have contrasted a variety of raycasting implementations and other volume rendering algorithms by ways of artificial test datasets and assessed the rendering quality by computing the RMS error and other statistical metrics. Both Williams and Uselton and Kwansik et.al have focused mainly on qual-

{meissner, bartz}@gris.uni-tuebingen.de, Auf Der Morgenstelle 10/C9, D-72076 Tübingen, Germany
{huangj, crawfis)@cis.ohio-state.edu, 2015 Neil Ave, 395 Dreese Lab, Columbus, OH, 43210, USA
(mueller}@cs.sunysb.edu, Stony Brook, NY 11794-4400, USA

ity. We shall complement these endeavors here with comparing the aforementioned volume rendering methods also in terms of rendering performance, on real-life datasets. Rendering quality will only be assessed visually.

The time for such comparative studies is clearly appropriate, as all of these volume rendering techniques have reached a high degree of maturity and quite a few heated arguments among researchers have revolved around the question of which algorithm is best. It is clear that such a clear-cut answer is unlikely to exist, and thus it is not expected that one single algorithm will emerge as an overall first choice in our, or any other, study. Rather, the results gathered here are aimed at providing the community with certain guidelines to determine under what conditions and premises each volume rendering algorithm is most adequately chosen and applied.

The outline of this paper is as follows. First, in Section 2, we describe two common optical models in which all four algorithms operate. Then, in Section 3, we discuss the peculiarities of the four algorithms with respect to this theoretical framework. In Section 4, we outline our benchmark scenarios and their evaluation criteria, and in Section 5 we present the results of our study. Section 6, closes the paper with concluding remarks and gives an outlook onto further studies and enhancements of our benchmarks.

2 COMMON OPTICAL MODEL

We can write all four investigated volume rendering methods as approximations of the well-known low-albedo volume rendering integral, VRI [5][17][26][15]. The VRI analytically computes $I_\lambda(\boldsymbol{x},\boldsymbol{r})$, the amount of light of wavelength λ coming from ray direction $\boldsymbol{r}$ that is received at location $\boldsymbol{x}$ on the image plane:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \int_0^L C_\lambda(s)\mu(s)e^{\left(-\int_0^s \mu(t)dt\right)}ds \qquad (1)$$

Here, L is the length of ray $\boldsymbol{r}$. If we think of the volume as being composed of particles with certain densities (or light extinction coefficients [26]) μ, then these particles receive light from all surrounding light sources and reflect this light towards the observer according to their specular and diffuse material properties. In addition, the particles may also emit light on their own. Thus, in (1), C_λ is the light of wavelength λ reflected and/or emitted at location s in the direction of $\boldsymbol{r}$. To account for the higher reflectance of particles with larger densities, we must weigh the reflected color by the particle density. The light scattered at s is then attenuated by the densities of the particles between s and the eye according to the exponential attenuation function.

At least in the general case, the VRI cannot be computed analytically [26]. Hence, practical volume rendering algorithms discretize the VRI into a series of sequential intervals i of width Δs:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\mu(s_i)\Delta s \cdot \prod_{j=0}^{i-1} e^{(-\mu(s_j)\Delta s)} \qquad (2)$$

Using a Taylor series approximation of the exponential term and dropping all but the first two terms, we get the familiar compositing equation [20]:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1} (1-\alpha(s_j)) \qquad (3)$$

We denote this expression as discretized VRI (DVRI), where opacity $\alpha = 1.0 - \text{transparency}$. Expression (3) represents a common theoretical framework for all surveyed volume rendering algorithms. All algorithms obtain colors and opacities in discrete intervals along a linear path and composite them in front to back order. However, the algorithms can be distinguished by the process in which the colors $C(s_i)$ and opacities $\alpha(s_i)$ are calculated in each interval i, and how wide the interval width Δs is chosen.

The position of the shading operator in the volume rendering pipeline also affects $C(s_i)$ and $\alpha(s_i)$. For this purpose, we distinguish the pre-shaded from the post-shaded volume rendering pipeline. In the pre-shaded pipeline, the grid samples are classified and shaded *before* the ray sample interpolation takes place. We denote this as Pre-DVRI (pre-shaded DVRI) and its mathematical expression is identical to (3). It was recently shown by Wittenbrink et. al. [48] that it is incorrect to interpolate voxel colors and opacities separately and then compute the product $C(s_i)\cdot\alpha(s_i)$. Rather, one must multiply color and opacity beforehand at each voxel and then interpolate the product. One disturbing feature of Pre-DVRI is that it tends to excessive blurring when the image resolution exceeds that of the volume, which occurs in zoomed viewing or at wide perspectives [30][13].

The blurriness can be eliminated by switching the order of classification/shading and ray sample interpolation. Then, the original density volume f is interpolated and the resulting sample values $f(i)$ are classified, via transfer functions, to yield material, opacity, and color. All blurry parts of the edge image can be clipped away using the appropriate classification function [30]. Shading follows immediately after classification and requires the computation of gradients from the density grid. The resulting expression is termed Post-DVRI (post-shaded DVRI) and is written as follows:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s} C_\lambda(f(s_i))\alpha(f(s_i)) \cdot \prod_{j=0}^{i-1} (1-\alpha(f(s_j))) \qquad (4)$$

C and α are now transfer functions, commonly implemented as lookup-tables. Since in Post-DVRI the raw volume densities are interpolated and used to index the transfer functions for color and opacity, fine detail in these transfer functions is readily expressed in the final image. One should note, however, that Post-DVRI is not without problems: Due to the partial volume effect, a density might be interpolated that is classified as a material not really present at the sample location, which can lead to false colors in the final image. This can be avoided by prior segmentation, which, however, can add severe staircasing artifacts due to introduced high-frequency.

Expressions (3) and (4) represent two common optical models for the four surveyed algorithms, depending on the order of the classification/interpolation components. The algorithms differ mostly in the way interpolation and sampling is performed and also how interpolations are avoided in less relevant or irrelevant volume regions.

3 DISTINGUISHING FEATURES

Our comparison will focus on the conceptual differences between the algorithms, and not so much on ingenious measures that speed runtime. Since numerous implementations for each algorithm exist — mainly providing acceleration — we will select the most general implementation for each, employing the most popular components and parameter settings. More specific implementations can then use the benchmarks introduced later to compare the impact of their improvements. We have summarized the conceptual differences of the four surveyed volume rendering algorithms as well as our assumptions in Table 1, for convenience.

3.1 Raycasting

Of all volume rendering algorithms, Raycasting has seen the largest body of publications over the years. Researchers have used Pre-DVRI [20][21] as well as Post-DVRI [12][39][2][40]. The density and gradient (Post-DVRI), or color and opacity (Pre-DVRI), in each DVRI interval are generated via point sampling, most commonly by means of a trilinear filter from neighboring voxels (grid points) to maintain computational efficiency, and subsequently composited. Most authors space the ray samples apart in equal distances Δs, but some approaches exist that jitter the sampling positions to eliminate patterned sampling artifacts, or apply space-leaping [10][49] for accelerated traversal of empty regions. For strict iso-surface rendering, recent research analytically computes the location of the iso-surface, when the ray steps into a voxel that is traversed by one [33].

But in the general case, the Nyquist theorem needs to be obeyed which states that we should choose $\Delta s \leq 1.0$ (i.e., one voxel length) if we do not know anything about the frequency content in the sample's local neighborhood. Then, for Pre-DVRI and Post-DVRI raycasting, the $C(s_i)$, $\alpha(s_i)$, and $f(s_i)$ terms in equations (3) and (4), respectively, are written as:

$$C_\lambda(s_i) = C_\lambda(i\Delta s) \qquad \alpha(s_i) = \alpha(i\Delta s) \qquad f(s_i) = f(i\Delta s) \tag{5}$$

Note that α needs to be normalized for $\Delta s \neq 1.0$ [22]. Our implementation uses early ray termination, which is a powerful acceleration method of raycasting where rays can be terminated when the accumulated opacity has reached a value close to unity. Furthermore, all samples and corresponding gradient components are computed by trilinear interpolation of the respective grid data.

3.2 Splatting

Splatting was proposed by Westover [46], and it works by representing the volume as an array of overlapping basis functions, commonly Gaussian kernels with amplitudes scaled by the voxel values. An image is then generated by projecting these basis functions to the screen. The screen projection of these radially symmetric basis function can be efficiently achieved by the rasterization of a precomputed footprint lookup table. Here, each footprint table entry stores the analytically integrated kernel function along a traversing ray. A major advantage of splatting is that only voxels relevant to the image must be projected and rasterized. This can tremendously reduce the volume data that needs to be both processed and stored [31].

The traditional splatting approach [46] summed the voxel kernels within volume slices most parallel to the image plane. This was prone to severe brightness variations in animated viewing and also did not allow the variation of the DVRI interval distance Δs. Mueller et. al. proposed a method [32] that eliminates these drawbacks by processing the voxel kernels within slabs of width Δs, aligned parallel to the image plane — hence the approach was termed *image-aligned splatting*: All voxel kernels that overlap a slab are clipped to the slab and summed into a sheet buffer, followed by compositing the sheet with the sheet before. Efficient kernel slice projection is achieved by analytical pre-integration of an array of kernel slices and using fast slice footprint rasterization methods [14]. Both Pre-DVRI [46] and Post-DVRI [30] are possible[1], and the $C(s_i)$, $\alpha(s_i)$, and $f(s_i)$ terms in equations (3) and (4) are now written as:

$$C_\lambda(s_i) = \frac{\int_{i\Delta s}^{(i+1)\Delta s} C_\lambda(s)ds}{\Delta s} \qquad \alpha(s_i) = \frac{\int_{i\Delta s}^{(i+1)\Delta s} \alpha(s)ds}{\Delta s} \qquad f(s_i) = \frac{\int_{i\Delta s}^{(i+1)\Delta s} f(s)ds}{\Delta s} \tag{6}$$

We observe that splatting replaces the point sample of raycasting by a sample average across Δs. This introduces an additional low-pass filtering step that helps to reduce aliasing, especially in isosurface renderings and when $\Delta s > 1.0$. Splatting also typically uses rotationally symmetric Gaussian kernels, which have better anti-aliasing characteristics than linear filters, with the side effect that they perform some signal smoothing. Splatting can use a concept similar to early ray termination: early splat elimination, based on a dynamically computed screen occlusion map, that (conservatively) culls invisible splats early from the rendering pipeline [31]. The main operations of splatting are the transformation of each relevant voxel center into screen space, followed by an index into the occlusion map to test for visibility, and in case it is visible, the rasterization of the voxel footprint into the sheetbuffer. The dynamic construction of the occlusion map requires a convolution operation after each sheet-buffer composite, which, however, can be limited to buffer tiles that have received splat contributions in the current slab [31]. It should be noted that, although early splat elimination saves the cost of footprint rasterization for invisible voxels, their transformation must still be performed to determine their occlusion. This is different from early ray termination where the ray stops and subsequent voxels are skipped.

3.3 Shear-Warp

Shear-warp was proposed by Lacroute and Levoy [18] and has been recognized as the fastest software renderer to date. It achieves this by employing a clever volume and image encoding

1. Note that the original splatting method [46] was commonly only used with Pre-DVRI, and therefore partial volume effects were never observed. The more recent, higher-quality splatting approach has been used with both Pre-DVRI [32] and Post-DVRI [30], with the latter possibly introducing artifacts due to the partial volume effect.

	Raycasting	Splatting	Shear-Warp	3D Texture Mapping
Sampling rate	freely selectable	freely selectable	fixed [1.0, 0.58]	freely selectable
Sample evaluation	point sampled	averaged across Δs	point sampled	point sampled
Interpolation kernel	trilinear	Gaussian	bilinear	trilinear
Rendering pipeline	post-classified	post-classified	pre-classified, opacity-weighted colors	pre-classified, no opacity-weighted colors
Acceleration	early ray termination	early splat elimination	RLE opacity encoding	graphics hardware
Precision/channel	floating point	floating point	floating point	8-12 bits
Voxels considered	all	relevant	relevant	all

Table 1: Distinguishing features and commonly used parameters of the four surveyed volume rendering algorithms

scheme, coupled with a simultaneous traversal of volume and image that skips opaque image regions and transparent voxels. In a pre-processing step, voxel runs are RLE-encoded based on pre-classified opacities. This requires the construction of a separate encoded volume for each of the three major viewing directions. The rendering is performed using a raycasting-like scheme, which is simplified by shearing the appropriate encoded volume such that the rays are perpendicular to the volume slices. The rays obtain their sample values via bilinear interpolation within the traversed volume slices. A final warping step transforms the volume-parallel baseplane image into the screen image. The DVRI interval distance Δs is view-dependent, since the interpolation of sample values only occurs in sheared volume slices. It varies from 1.0 for axis-aligned views to 1.41 for edge-on views to 1.73 for corner-on views, and it cannot be varied to allow for supersampling along the ray. Thus the Nyquist theorem is potentially violated for all but the axis-aligned views.

The Volpack distribution from Stanford [1] only provides for Pre-DVRI, but conceptually Post-DVRI is also feasible, however, without opacity classification if shear-warp's fast opacity-based encoding is supposed to be used. Shear-warp uses opacity-weighted colors in its Pre-DVRI interpolation. The $C(s_i)$, $\alpha(s_i)$, and $f(s_i)$ terms in equations (3) and (4) are written similarly to raycasting, but with the added constraint that Δs is dependent on the view direction:

$$C_\lambda(s_i) = C_\lambda(i\Delta s) \qquad \alpha(s_i) = \alpha(i\Delta s) \qquad f(s_i) = f(i\Delta s)$$
$$\Delta s = \sqrt{\left(\frac{dx}{dz}\right)^2 + \left(\frac{dy}{dz}\right)^2 + 1} \tag{7}$$

where $[dx, dy, dz]^T$ is the normalized viewing vector, reordered such that *dz* is the major viewing direction. In Volpack, the number of rays sent through the volume is limited to the number of pixels in the base plane (i.e., the resolution of the volume slices in view direction). Larger viewports are achieved by bilinear interpolation of the resulting image (after back-warping of the base plane), resulting in a very low image quality if the resolution of the viewport is significantly larger than the volume resolution. This can be fixed by using a scaled volume with a higher volume resolution.

3.4 3D Texture-Mapping Hardware

The use of 3D texture mapping was popularized by Cabral [6] for non-shaded volume rendering. The volume is loaded into texture memory and the hardware rasterizes polygonal slices parallel to the viewplane. The slices are then blended back to front, due to the missing accumulation buffer for α. The interpolation filter is a trilinear function (on SGI's RE^2 and IR architectures, quadlinear interpolation is also available, where it additionally interpolates between two mipmap levels), and the slice distance Δs can be chosen freely. A number of researchers have added shading capabilities [9][27][43][45], and both Pre-DVRI [43] and Post-DVRI [45][9][27] are possible. The latter requires multi-pass methods. Usually, the rendering is brute-force, without any opacity-based termination acceleration, but some researchers have done this [9]. The drawbacks of 3D texture mapping is (i) that it is still limited to expensive, proprietary graphics hardware (SGI, Hewlett-Packard as OpenGL extension), and (ii) that larger volumes require the swapping of volume bricks in and out of the limited-sized texture memory (usually 4MB for smaller machines). Texture-mapping hardware interpolates samples in similar ways to raycasting and hence the $C(s_i)$, $\alpha(s_i)$, and $f(s_i)$ terms in equations (3) and (4) are written as:

$$C_\lambda(s_i) = C_\lambda(i\Delta s) \qquad \alpha(s_i) = \alpha(i\Delta s) \qquad f(s_i) = f(i\Delta s) \tag{8}$$

The factor that constrains quality in 3D texture mapping hardware approaches is the fact that the framebuffer has limited bit resolution (8-12 bits). This is far less than the floating point precision that can be obtained with the software algorithms. We have found that the limited bit-resolution severely limits the use of opacity-weighted colors, since the opacity-weighting reduces the color values below the resolution of the framebuffer when the opacities are low. This limits the compositing of low intensity volume areas with low opacities. As a remedy, one could scale the voxel colors up before opacity-weighting, but then one may saturate other areas. We have therefore chosen not to use opacity-weighted colors for the 3D texture mapping hardware.

4 EXPERIMENTAL METHODS

In this section, we will describe the complete rendering environment, i.e., shading model, viewing parameters, lighting, and transfer functions. We also introduce the benchmark datasets. This section completes the full specification of the rendering parameters used in our experiments and yields a comprehensive set of benchmark scenarios for the evaluation of DVRs.

4.1 Shading model and compositing

All approaches use the same shading model:

$$C_\lambda = CLUT_\lambda[f](k_a I_a + k_d(NL)I_L) + k_s(NH)^{ns} I_L \tag{9}$$

where C_λ is the rendered color, f is the (grid or interpolated) density value, CLUT is the color transfer function implemented as a lookup table, I_L is the color of the lightsource, and k_a, k_d, k_s are the ambient, diffuse, and specular coefficients of the object material, respectively. N is the normal vector and H is the half vector, respectively.

All algorithms, except 3D texture-mapping hardware (see above), use front-to-back compositing [34][20]:

$$\begin{aligned} c_f &= c_f + C_b\alpha_b(1-\alpha_f) \\ \alpha_f &= \alpha_f + \alpha_b(1-\alpha_f) \end{aligned} \tag{10}$$

The opacity α is obtained from the (interpolated or grid) density via indexing of an α-transfer function, implemented as a lookup table. For the pre-shaded renderers, the color is opacity-weighted [48] (i.e., the $c_b\alpha_b$ product in (10) is interpolated).

4.2 Viewing, lighting, transfer functions and datasets

We can characterize a volumetric object by the amount of relevant material contained in its convex hull. We will denote this measure as *compactness*. A highly compact object, such as a brain or an engine block, fills a large percentage of its enclosing space. On the other hand, a highly dispersed object, such as a ganglion nerve cell or a blood vessel tree, has relatively few relevant voxels within its convex extent. We may add that the compactness of a volume is not always pre-defined, but can be altered by the opacity transfer function. Thus, a formerly compact object, such as an MRI head, may turn into a sparse and dispersed object, such as a blood vessel tree. Volume rendering algorithms can take advantage of this, with varying level of complexity. Splatting, by default, only renders the relevant voxels, no matter how compact they are spatially. Raycasting must traverse the empty regions, but can use space-leaping [49] or polygon-assisted raycasting (i.e., PARC [37]) to traverse these regions quickly. However, the more twisted the object is, the more polygons are needed for PARC and the less effective space-leaping becomes. Finally, shear-warp encodes the sparseness in its RLE runs. Both of the latter methods require a pre-processing step to generate the volume encoding.

Apart from compactness, another useful measure is the amount of voxel material that contributes to a pixel. This will be referred to as the *pixel content*. Semi-transparent renderings of a particular object consider significantly more object voxels than opaque renderings. Here, we distinguish the following rendering modes: Semi-transparent with embedded opaque objects, fully semi-transparent with no embedded opaque objects, or fully opaque objects (i.e, a single iso-surface). To test our volume rendering algorithms under a broad scope of scenarios, we have selected a set of five benchmark datasets and opacity transfer functions, each representing a different combination of compactness and pixel content[1]. These are summarized in Table 2. In this table, we have also listed a dataset named Marschner-Lobb [25], which will be used to assess rendering quality in a quantitative manner. We will expand on this further below. The datasets are:

- Fuel injection: Physical simulation of diesel being injected into a cylinder of an engine filled with air. This is a semi-transparent, but compact, representation that requires many samples to be taken for each pixel.
- Neghip: Physical simulation of a high potential protein representing the electron probability around each atom (blue is high, green is medium, and red is low). This dataset has some dispersed elements.
- Skull: Rotational biplane X-ray scan of a human head. Bones and teeth are well scanned. The classification of the data into skull and teeth yields moderate compactness. The opacity transfer function also enables early ray termination, and many voxels are occluded.
- Blood vessel: Rotational biplane X-ray (rotational angiography) scan of a human brain where a contrast agent has been injected into the blood to capture the blood vessel. This dataset is characterized by its very low compactness and pixel content.
- Shockwave: Simulation of a unsteady interaction of a planar shockwave with a randomly-perturbed contact discontinuity, rendered with a highly translucent transfer function. All voxels potentially contribute to the display.
- Marschner-Lobb: High frequency test dataset, rendered as an isosurface.

Both rendering quality and expense is likely to be dependent on viewpoint, magnification, as well as image size. To study these effects, we have rendered the datasets at four to five magnification levels each and into six image dimensions, i.e., 64, 128, 256, 512, 1024, and 2048 pixels squared. Here, we aimed to cover all available display media ranges (sorted from small to large): Java rendering over the web, head-mounted display, computer screen, high-definition screen, CAVE projection wall, and PowerWall. To get statistically valid results we have also rendered a series of 24 images at random viewpoints over an enclosing sphere. No frame-to-frame coherence in either data access or rendering was exploited. A specific storage order of the volumes was not allowed (Volpack with its three encoded volumes is an exception). Diagonal views are especially testing for the shear-warp algorithm in terms of quality (the ray step size is significantly below Nyquist), and for the splatting algorithm in terms of rendering time (the bucket-tossing of the splats into the slabs is non-trivial). Raycasting may incur some caching delays when the volume is accessed out of stride, while the texture mapping hardware will most likely be least sensitive to the change of viewing directions.

1. These datasets, transfer functions, and viewing parameters are publicly available on the internet at http://www.volvis.org

4.3 Assessment of image quality

It is difficult to evaluate rendering quality in a quantitative manner. Often, researchers simply put images of competing algorithms side by side, appointing the human visual system (HVS) to be the judge. It is well known that the HVS is less sensitive to some errors (stochastic noise) and more to others (regular patterns), and interestingly, sometimes images with larger numerical errors, e.g., RMS, are judged as worse by a human observer than images with lower numerical errors [38]. So it seems that the visual comparison is more appropriate than the numerical, since after all we produce images for the human observer and not for error functions. In that respect, an error model that involves the HVS characteristics [24] would be more appropriate than a purely numerical one. But nevertheless, to perform such a comparison we still need the true volume rendered image, obtained by analytically integrating the volume via equation (1) (neglecting the prior reduction of the volume rendering task to the low-albedo case). As was pointed out by Max [26], analytical integration can be done when assuming that C(s) and μ(s) are piecewise linear. This is, however, somewhat restrictive on our transfer functions. Hence we decided to employ visual quality assessment only.

Apart from the real-life datasets, we also chose a particularly challenging dataset for visual quality assessment: the Marschner-Lobb function [25]. This three-dimensional function is made of a combination of sinusoids and contains very high frequencies, however, 99.8% of these are below a frequency of 10 times the Nyquist rate when sampled into a 41^3 lattice. It is extremely sensitive to filter and sampling inaccuracies and has been used at many occasions for reconstruction error evaluations [25][28].

5 RESULTS

Before one goes ahead and compares rendering times and quality, one needs to realize that not all evaluated volume renderers were created with identical priorities in mind. While shear-warp and 3D texture mapping hardware were devised to maximize framerates on the expense of rendering quality, image-aligned splatting and raycasting have been devised to achieve images of high quality, not to be compromised by acceleration strategies employed. To account for this, we have divided the four renderers into two groups of two renderers each:

- High-performance volume renderers: Shear-warp and 3D texture mapping hardware (from now on abbreviated *TEX*). These renderers use the Pre-DVRI optical model with pre-multiplied opac-

Dataset	Size	Relevant voxels	Rendering mode	Compactness	Pixel content
Blood vessel	256^3	79,442 (0.5%)	opaque isosurface	low	low
Neghip	64^3	207,872 (79.3%)	moderately semi-transparent	high	medium
Skull	256^3	1,384,817 (8.2%)	opaque isosurface	medium	low
Fuel injection	64^3	32,768 (12.5%)	semi-transparent with interior opaque structure	high	medium
Shockwave	$64^2 \times 512$	1,245,184 (59%)	fully semi-transparent	high	high
Marschner-Lobb	41^3	35,415 (51%)	opaque isosurface	high	low

Table 2: Benchmark datasets and rendering modes

	Fuel	Neghip	Skull	Blood	Shock
Raycasting	4.96	8.15	7.78	12.31	3.02
Splatting	1.41	7.35	11.09	1.87	21.77
Shear-warp	0.09	0.24	0.27	0.09	0.91
Texture map	0.06	0.04	0.7	0.7	0.14

Table 3: Average frame times (in secs) for 24 random views onto the five datasets. The image size was 256^2, and the object was viewed in a screen filling shot. Note that 3D Texture mapping is slower than shear-warp for large datasets (skull, blood) which exceed the 4 MB of texture memory of the used SGI Octane/MXE and require texture swaps.

ities [48], which benefits their rendering speed but limits the image quality under magnification.

- High-quality volume renderers: Splatting and raycasting. These renderers use the Post-DVRI optical model.

All presented results were generated on an SGI Octane (R10000 CPU, 250MHz) with 250 MB main memory and MXE graphics with 4MB of texture memory. The graphics hardware was only used by the 3D texture mapping approach. Figure 3 shows representative still frames of the six datasets that we rendered with the four volume rendering algorithms. Figure 1 relates frame times to magnification factors. Note that we did not include the time/frame for the high performance renderers shear-warp and TEX into Figures 1 and 2 since (i) the dependencies are relatively small and (ii) the scale of the plot forces both graphs to the x-axis. The icon images next to the graphs indicate the level of magnification as well as the viewpoint (the icon images were rendered with the raycaster). Figure 2 shows how image size affects rendering time of the screen filling shots (a), (d), (f), (g), and (i) of Figure 3. Finally, Table 3 lists the average frame time for the 24 randomly generated views. For these random views we set the magnification factors such that the object just filled the screen.

In Figure 3, we observe that the image quality achieved with TEX shows severe color-bleeding artifacts due to the non-opacity weighted colors [48], as well as staircasing. The latter is due to the limited precision of the framebuffer and can be reduced by increasing the number of slices.

Volpack shear-warp performs much better, with quality similar to raycasting and splatting whenever the resolution of the image matches the resolution of the baseplane (Figure 3(a), (d), and (i)). For the other images, the rendered baseplane image was of lower resolution than the screen image and had to be magnified using bilinear interpolation in the warping step. This leads to excessive blurring, especially for the Marschner-Lobb dataset, where the magnification is over 6 (Figure 3(k)). A more fundamental drawback can be observed in the 45˚ neghip view in Figure 3, where — in addition to the blurring — significant aliasing in the form of staircasing is present. This is due to the ray sampling rate being less than 1.0, and can be disturbing in animated viewing of some datasets (see provided animations).

The Marschner-Lobb dataset renderings for raycasting and splatting demonstrate the differences of point sampling (raycasting) and sample averaging (splatting). While raycasting's point sampling misses some detail of the function at the crests of the sinusoidal waves, splatting averages across the waves and renders them as blobby rims. For the other datasets the averaging effect is more subtle, but still visible. For example, raycasting renders the skull and the magnified blood with somewhat crisper detail than splatting does, but suffers from aliasing artifacts, if the sampling rate is not chosen appropriately. However, the quality is quite comparable, for all practical purposes.

But even though the quality is comparable, there are still subtle differences in the images rendered by each algorithm. These differences are most apparent for the fuel, neghip, and shockwave datasets. For example, the red cloud is completely missing in the image rendered by splatting.

From Table 3, we observe that the frame times for both TEX and shear-warp are always substantially faster than those of raycasting and splatting, in most cases by an order of one or two magnitudes. Both TEX and shear-warp consistently achieve frame times in the subsecond range. In TEX all data is always sliced and composited by the graphics hardware in brute force. Shear-warp's frame times are a function of the number of relevant voxels and opaqueness. It takes roughly three times longer to render the translucent shockwave dataset than the opaque skull, although both have about the same number of relevant voxels. This was to be expected and is due to Volpack's early ray termination. An interesting case is the blood vessel dataset, where splatting's frame time is only about twice that of TEX. This is because here the rendered active voxel list of splatting was only a very small percentage (0.5%, see Table 2) of the brute-force rendered TEX volume. Both TEX and shear-warp are relatively insensitive to magnification since the number of rendered pixels is held constant. For the shear-warp algorithm, the interpolation of the base plane into the magnified screen image takes little time.

Table 3 also shows that compactness of the dataset has no major effect on splatting: Both the blood vessel of low compactness and the fuel injection of high compactness render at about the same time. This was to be expected, since splatting's voxel lists discards spatial coherency. Further, we observe that splatting does best when the relevant voxel list is small, as is the case for both the blood vessel and the fuel injection. We see that early splat elimination is quite effective since the pixel content has a large effect on rendering time: Although the opaque blood vessel (low pixel content) has twice the amount of relevant voxels, it requires about the same frame time as the fuel injection (medium pixel content). The same is true for the skull and shockwave datasets. Both have a similar number of relevant voxels, but the shockwave, with high pixel content, renders twice as slow as the skull.

On the other hand, raycasting shows its strength for medium and highly compact datasets with a low number of non-contributing samples along the ray (i.e., the shockwave dataset in Table 3). In contrast, a high number of non-contributing samples can dominate the rendering time, as observed with the vessel and fuel datasets. In both cases most of the rays are cast solely through non-relevant voxels wasting render time. This is different with the skull dataset, where early ray termination skips most empty space. The associated costs of early ray termination are low, since it is a simple comparison of the α–value with a specified threshold. In contrast, splatting's early splat elimination has high associated costs, and that is why splatting takes considerably longer to render the skull dataset. The shockwave dataset, where the low opacity of all voxels prevents both early ray termination and early splat elimination, exposes the differences in cost for trilinear interpolation vs. footprint mapping. Since the rendering time for raycasting is almost seven times lower than that of splatting, we conclude that the mapping of footprint kernels is costlier than trilinear interpolation, at least at moderate screen sizes.

The magnification plots of Figure 1 indicate that it is better to rasterize a small number of large footprints, even when they fill the entire screen, than to rasterize and transform a large number of small footprints. Good examples for this are the skull and the neghip datasets where at large magnification the number of splats that survive frustum culling is small, but their footprints are large.

Figure 2 illustrates the relationship of image dimension vs. rendering time. But before we go about analyzing them, there are a few facts that are worth mentioning:

- For splatting, the amount of relevant voxels is independent of image dimension. The increases in rendering time are due to (i) larger footprints to be rasterized and (ii) larger sheetbuffers to be convolved to update the occlusion map. Thus, datasets with many sheets (large depth) and many visible footprints will be more sensitive to screen size increases.
- For raycasting, the number of trilinear interpolations and compositings are linearly related to screen size. No further major dependencies exist.

We observe that for both the fuel and the blood vessel datasets the differences between splatting and raycasting become more pronounced as the screen size increases, and that the neghip dataset continues to have a balanced cost ratio for the constituents of the two algorithms. However, the shockwave and the skull dataset exhibit a reversal of the rendering cost relationships at larger screen sizes. Both datasets render faster with splatting in this case. This is due to the fact that although the cost for footprint rasterization and occlusion map maintenance grows with screen size, the time for transforming the voxels and for setting up the footprint rasterization (i.e., mapping the voxel center and computing the footprints screen extent) does not. There seems to exist a critical screen size for each dataset in which the cost for the increased number of trilinear interpolations overtakes the cost for footprint mappings, due to the screen size-independent components in this operation.

6 CONCLUSIONS

Both high-performance renderers, i.e., 3D texture mapping and shear-warp, have sub-second rendering times for the more moderately-sized datasets. The quality of the display obtained with the texture mapping approach we have used is limited, due to the lack of bit accuracy. A framebuffer color/RGBA resolution of least 16 bits or more would be desirable. Simulations are needed to determine the minimum number of bits needed for good image quality. On the other hand, the quality of shear-warp rivals that of the much more expensive raycasting and splatting when the object magnification is about unity. Handling higher magnifications is possible by relaxing the condition that the number of rays must match the resolution of the volume. Although higher interpolation costs will be the result, the rendering frame rate will most likely still be high (especially if view frustum culling is applied). A more serious concern is the degradation of image quality at off-axis views. In these cases, one could use a volume with extra interpolated slices, which is Volpack's standard solution for higher image resolutions. But the fact that shear-warp requires an opacity-encoded volume makes interactive transfer function variation a challenge[1]. In applications where these limitations do not apply, shear-warp proves to be a very useful algorithm for volume rendering.

The side-by-side comparison of splatting and raycasting yielded interesting results as well: We saw that image-aligned splatting offers a rendering quality similar to that of raycasting. It, however, produces smoother images due to the z-averaged kernel and the anti-aliasing effect of the larger Gaussian filter. It is hence less likely to miss high-frequency detail. Raycasting is faster than splatting for datasets with a low number of non-contributing samples. On the other hand, splatting is better for datasets with a small number of relevant voxels and sheetbuffers. It seems that rotational angiography visualization would benefit from a splatting algorithm, while typical semitransparent scientific data would benefit from raycasting. Since the quality is so similar and the same transfer functions yield similar rendering results, one could build a renderer that applies either raycasting or splatting, depending on the number of relevant voxels and the level of compactness. One could even use different renderers in different portions of the volume, or for the rendering of disconnected objects of different compactness in the scene.

1. If no pre-classification is provided, Volpack renders at a lower frame rate.

An important outcome of this study is splatting's strength when it comes to large volume magnifications, required in high-resolution displays, such as PowerWalls and CAVEs. Splatting can generate a high-quality image faster than raycasting in this case, and does not cause the extensive blurring of shear-warp.

Based on the insight gained, can we resolve the weaknesses of the algorithms? For raycasting, the problem areas are the transparent regions in front of the opaque object portions, while for splatting the problem areas are the non-transparent regions behind the opaque object portions. This may be a compromising fact for splatting, since large datasets may have a lot more material hidden in the back than empty regions in front. In addition, a great number of powerful techniques exist for raycasting to guide rays quickly through irrelevant volume regions: bounding volumes, space-leaping [49], multi-resolution grid traversal [33], and PARC [37]. It has yet to be determined if these techniques also work well with highly irregular objects such as the blood vessel dataset, where splatting does so well. Further, volumes with a high percentage of relevant voxels, such as the shockwave dataset, will not be able to benefit much of space leaping techniques, and thus our current conclusions will continue to hold for these data sets.

Much fewer work has been done on accelerating splatting, and most work has focussed on the speed of footprint rasterization [14]. Our analysis indicates that future research must focus on reducing the cost for handling invisible material (i.e., sheet convolution, occlusion culling, and voxel transformations). Octree encoding could be used, along with a simultaneous occlusion map/volume traversal scheme similar to that employed in shear-warp. Using smaller kernels, such as the Crawfis-Max kernel [8], will also provide faster renderings and bring the kernel size closer to that used in raycasting. We have measured speedups of 25% to 45% when reducing the kernel radius from the standard 2.0 to 1.58, without a significant drop in quality.

We noticed that there were subtle differences in the images rendered by different algorithms. This is due to the fact that each method uses a somewhat different interpolation scheme, but all use the same transfer function. The post-DVRI pipeline before shading can be written as: $f' = T(f \otimes \text{kernel})$, where T is the transfer function and $\otimes$ is the convolution operator. Clearly, f' will not be identical for the different algorithms, and therefore the images will not be exactly alike. Thus, unless one can come up with a master transfer function from which all others can be algorithmically derived, one needs to fine tune each transfer function for the algorithm at hand.

In this study, we wanted to make certain that we conducted our experiments and analyses in plausible scenarios, and not artificial ones that may never occur in real life. Hence our choice of real-life datasets. And although these chosen benchmark datasets form a rather comprehensive mix and allowed a number of insightful conclusions, we still feel that many questions have been left unanswered. We therefore plan to add a set of procedural datasets to the mix where we can keep the ratio of occlusion to depth constant, and where we can keep the ratio of non-empty to empty pixels constant as well. This controlled and scalable environment would allow us to test hypotheses quickly and in a concise way. We also plan to perform a detailed profiling of the components of the four volume renderers, such as interpolation, transfer function look-ups, shading, compositing, visibility test, and occlusion-map maintenance (the last two are for splatting only). Further, we would like to incorporate various popular acceleration methods into the existing algorithms and evaluate their effects. In light of the performance of the shear-warp algorithm in this study, it seems also worthwhile to investigate if some of shear-warp's concepts can be ported to both raycasting and splatting, but without compromising the current rendering quality. Finally, we would also like to study the effect of perspective viewing and super- and subsampling in z on the rendering result. Large datasets of GB size are becoming increasingly popular these days. The current benchmarks have to be expanded in this direction. We can create a series of increasingly larger datasets that are derived from supersampling the current low-resolution benchmark datasets. This way we keep compactness and pixel content constant, and only change the number of voxels. Research challenges also still exist in devising extensions to shear-warp and 3D texture mapping algorithms to adequately handle the per-pixel (or sample) classification and shading.

ACKNOWLEDGEMENTS

We would like to thank Jürgen Schulze-Döbold for helpful discussion. The skull dataset is courtesy of Siemens Medical Systems, Forchheim, Germany and the blood vessel dataset is courtesy of Philips Research Labs, Hamburg, Germany. Finally, the fuel injection dataset is courtesy of SFB 382 of the DFG. This work was sponsored in part by the DOE ASCI Program, the German Research Council (DFG), the Hewlett-Packard Corporation, and a seed grant from the State University of New York at Stony Brook.

REFERENCES

[1] http://www-graphics.stanford.edu/software/volpack

[2] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang., "VolVis: a diversified volume visualization system", *Proc. Visualization' 94*, pp. 31-38, 1994.

[3] D. Bartz and M. Meißner, "Voxels versus polygons: A comparative approach for volume graphics", *Proc. Volume Graphics'99*, pp. 33-48, Swansea, 1999.

[4] M.J. Bentum, B.B.A. Lichtenbelt, T. Malzbender., "Frequency analysis of gradient estimators in volume rendering", *IEEE Trans. on Vis. and Comp. Graph.*, vol. 2, no. 3, pp. 242-254, 1996.

[5] J. F. Blinn, "Light reflection functions for simulation of clouds and dusty surfaces", *Proc. SIGGRAPH'82*, pp. 21-29, 1982.

[6] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", *1994 Symposium on Volume Visualization*, pp. 91-98, 1994.

[7] I. Carlbom, "Optimal filter design for volume reconstruction", *Proc. Visualization'93*, pp. 54-61, 1993.

[8] R. Crawfis, N. Max, "Texture splats for 3D scalar and vector field visualization", *Visualization'93*, pp. 261-266, 1993.

[9] F. Dachille, K. Kreeger, B. Chen, I. Bitter, A. Kaufman, "High quality volume rendering using texture mapping hardware", *Proc. 1998 Siggraph/Eurographics Workshop on Graph. Hardw.*, pp. 69-76, 1998.

[10] J. Danskin and P. Hanrahan, "Fast algorithms for volume raytracing", *1992 Workshop on Volume Visualization*, pp. 91-98, 1992.

[11] R. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering", *Proc. SIGGRAPH'88*, pp. 65-74, 1988.

[12] K.H. Hoehne B. Pfiesser, A. Pommert, M. Riemer, T. Schiemann, R. Schubert, U. Tiede., "A virtual body model for surgical education and rehearsal", *IEEE Computer*, vol. 29, no. 1, 1996.

[13] J. Huang, R. Crawfis, and D. Stredney, "Edge preservation in volume rendering using splatting", *1998 Symp. Vol. Vis.*, pp. 63-69, 1998.

[14] J. Huang, K. Mueller, N. Shareef, R. Crawfis, "FastSplats: optimized splatting on rectilinear grids," *Proc. Visualization'2000 (to appear).*

[15] J. T. Kajiya and B.P. Von Herzen, "Ray tracing volume densities", *Proc. SIGGRAPH '84*, pp. 165-174, 1994.

[16] K. Kwansik, C. Wittenbrink, A. Pang, "Maximal-abstract differences for comparing direct volume rendering algorithms," Hewlett-Packard Research Lab Technical Report 2000-40, available at http://www.hpl.hp.com/techreports/2000/HPL-2000-40.html

[17] W. Krueger, "The application of transport theory to the visualization of 3D scalar fields", *Computers in Physics 5*, pp. 397-406, 1991.

[18] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation", *Proc. SIGGRAPH '94*, pp. 451- 458, 1994.

[19] M. Levoy, "Display of surfaces from volume data", *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

[20] M. Levoy, "Efficient ray tracing of volume data", *ACM Trans. Comp. Graph.*, vol. 9, no. 3, pp. 245-261, 1990.

[21] M. Levoy, "Display of surfaces from volume data", *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

[22] B. Lichtenbelt, R. Crane, S. Naqvi, *Introduction to Volume Rendering*, Prentice-Hall, 1998.

[23] W. E. Lorensen and H. E. Cline, "Marching cubes: a high resolution 3D surface construction algorithm", *SIGGRAPH'87*, pp. 163-169.

[24] A. Gaddipati, R. Machiraju, R. Yagel "Steering Image generation using Wavelet Based Perceptual Metric", Comp. Graph. Forum, vol. 16, no. 3, pp. 241-251, 1997.

[25] S. Marschner and R. Lobb, "An evaluation of reconstruction filters for volume rendering", *Proc. Visualization'94*, pp. 100-107, 1994.

[26] N. Max, "Optical models for direct volume rendering", *IEEE Trans. Vis. and Comp. Graph.*, vol. 1, no. 2, pp. 99-108, 1995.

[27] M. Meißner, U. Hoffman, and W. Straßer, "Enabling classification and shading for 3D texture mapping based volume rendering", P*roc. Visualization'99*, pp. 207-214, 1999.

[28] T. Moeller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and Design of Filters Using a Taylor Series Expansion", *IEEE Trans. Vis. and Comp. Graph.*, vol. 3, no. 2, pp. 184-199, 1997.

[29] T. Moeller, R. Machiraju, K. Mueller, and R. Yagel, "A comparison of normal estimation schemes," *Proc. Visualization' 97*, pp. 19-26, 1997.

[30] K. Mueller, T. Moeller, and R. Crawfis, "Splatting without the blur", *Proc. Visualization'99*, pp. 363-371, 1999.

[31] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels", *IEEE Trans. Vis. and Comp. Graph.*, vol. 5, no. 2, pp. 116-134, 1999.

[32] K. Mueller and R. Crawfis, "Eliminating popping artifacts in sheet buffer-based splatting", *Proc. Visualization'98*, pp. 239-245, 1998.

[33] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive ray tracing for isosurface rendering", *Proc. Visualization '98*, pp. 233-238, 1998.

[34] T. Porter and T. Duff, "Compositing digital images", *Computer Graphics (SIGGRAPH '84 Proceedings)*, pp. 253-259, 1984.

[35] P. Sabella, "A rendering algorithm for visualizing 3D scalar fields", Proc. SIGGRAPH'88, pp. 51-58, 1988.

[36] J. van Scheltinga, J. Smit, and M. Bosma, "Design of an on-chip reflectance map", *Proc. 10th Eurographics Workshop on Graph. Hardw.*, pp. 51-55, 1995.

[37] L. Sobierajski and R. Avila, "A hardware acceleration method for volumetric ray tracing", *Proc. Visualization'95*, pp. 27-34, 1995.

[38] P.C. Teo and D.J. Heeger, "Perceptual image distortion," *Proc. 1st Intern. Conf. on Image Processing*, pp. 982-986, 1994.

[39] U. Tiede, K.H. Hoehne, M. Bomans, A. Pommert, M. Riemer, G. Wiebecke, "Investigation of medical 3D-rendering algorithms", *IEEE Comp. Graph. & Appl.*, vol. 10, no. 2, pp. 41-53, 1990.

[40] U. Tiede, T. Schiemann, K.H. Hoehne, "High quality rendering of attributed volume data," *Proc. Visualization'98*, pp. 255-262, 1998.

[41] H. Tuy and L. Tuy, "Direct 2D display of 3D objects", *IEEE Comp. Graph. & Appl.*, vol. 4 no. 10, pp. 29-33, 1984.

[42] C. Upson and M. Keeler, "VBUFFER: Visible Volume Rendering", Proc. SIGGRAPH'88, pp. 59-64, 1988.

[43] A. van Gelder and K. Kim, "Direct volume rendering with shading via three-dimensional textures", 1*996 Symp. on Vol. Vis.*, pp. 23-30, 1996.

[44] D. Voorhies and J. Foran, "Reflection vector shading hardware", *Proc. SIGGRAPH'94*, pp. 163-166, 1994.

[45] R. Westerman and T. Ertl, "Efficiently using graphics hardware in volume rendering applications", *SIGGRAPH'98*, pp. 169-177, 1998.

[46] L. Westover, "Footprint evaluation for volume rendering", *SIGGRAPH'90*, pp. 367-376, 1990.

[47] P.L. Williams and S.P. Uselton, "Metrics and generation specifications for comparing volume-rendered images," *The Journal of Visualization and Computer Animation*, vol. 10, pp. 159-178, 1999.

[48] C. Wittenbrink, T. Malzbender, and M. Goss, "Opacity-weighted color interpolation for volume sampling", *1998 Symposium on Volume Visualization*, pp. 135-142, 1998.

[49] R. Yagel and Z. Shi, "Accelerating volume animation by space-leaping", *Proc. Visualization'93*, pp. 63-69, 1993.

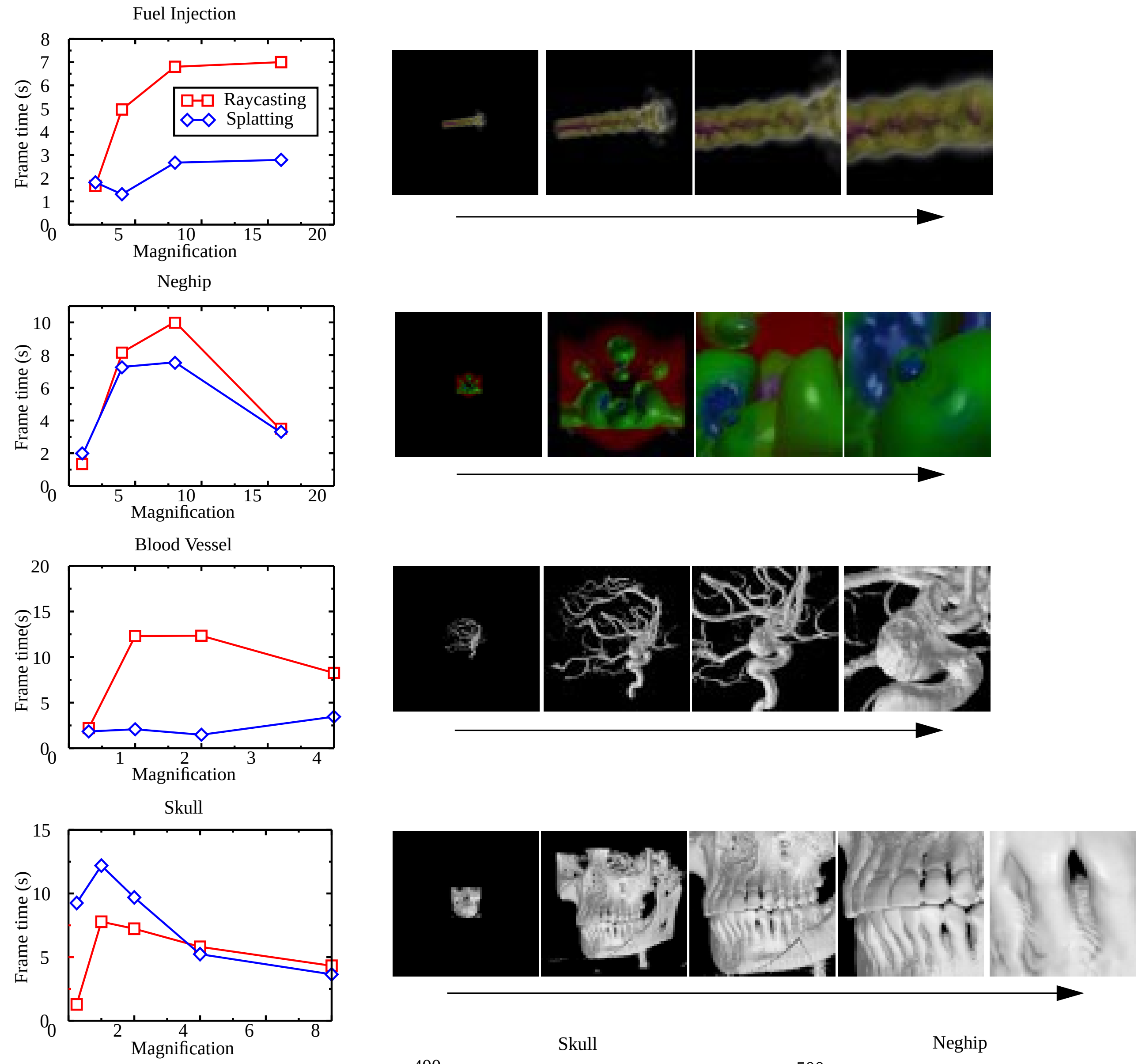
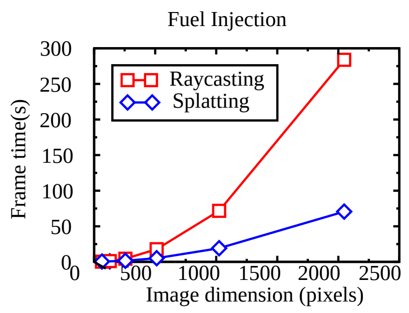


Figure 1: Frame time vs. magnification; image size is 256^2.



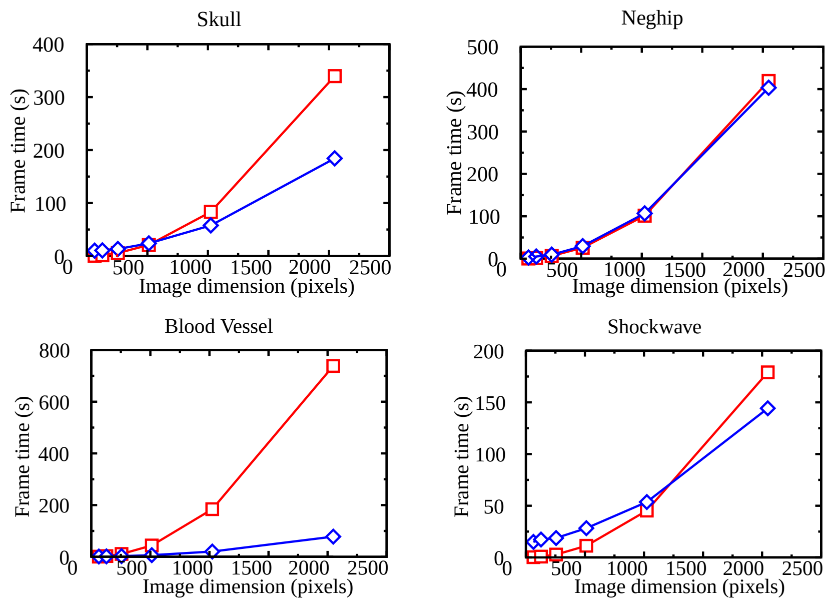


Figure 2: Frame time vs. frame size

CT Head (Skull)

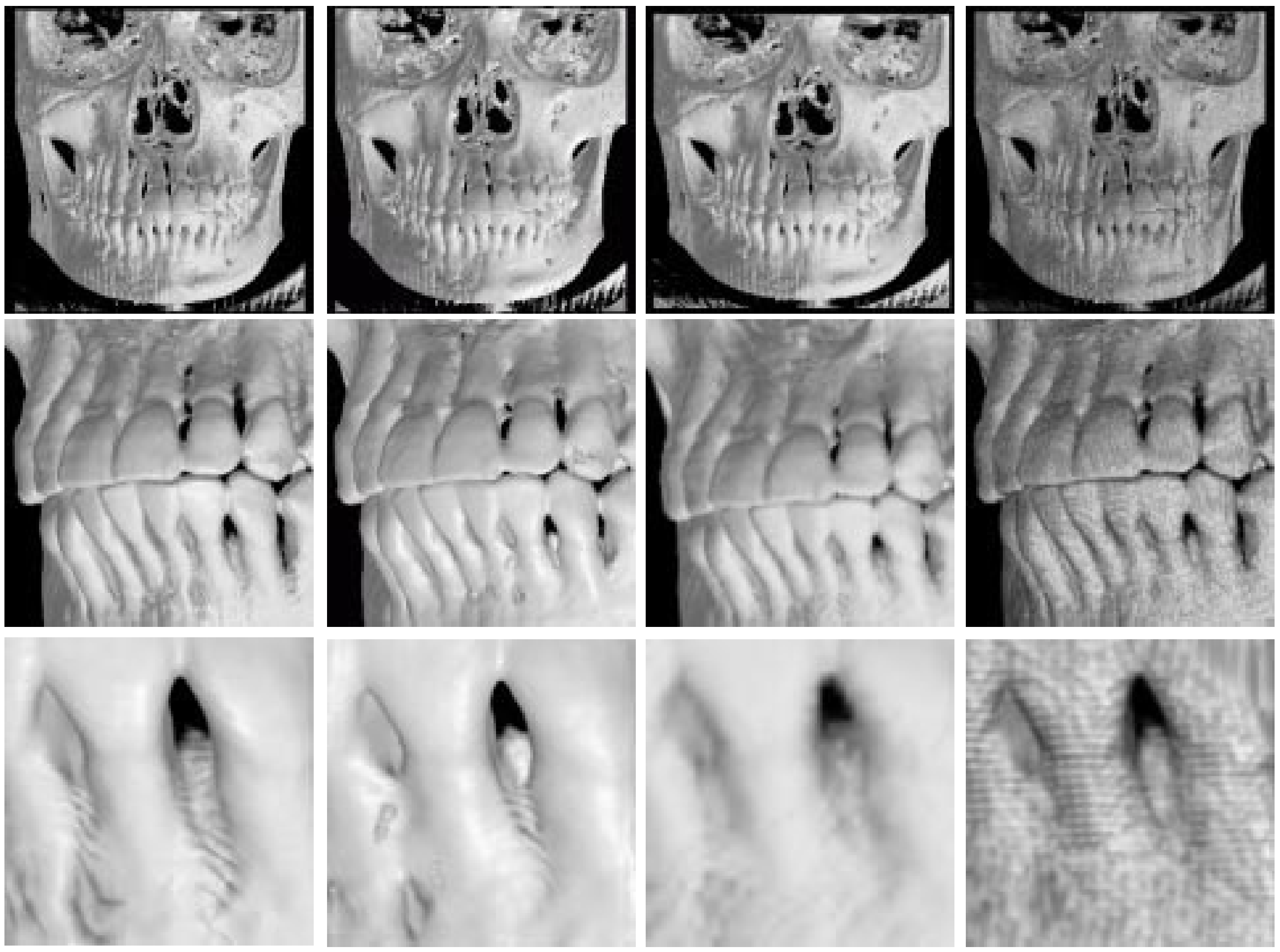
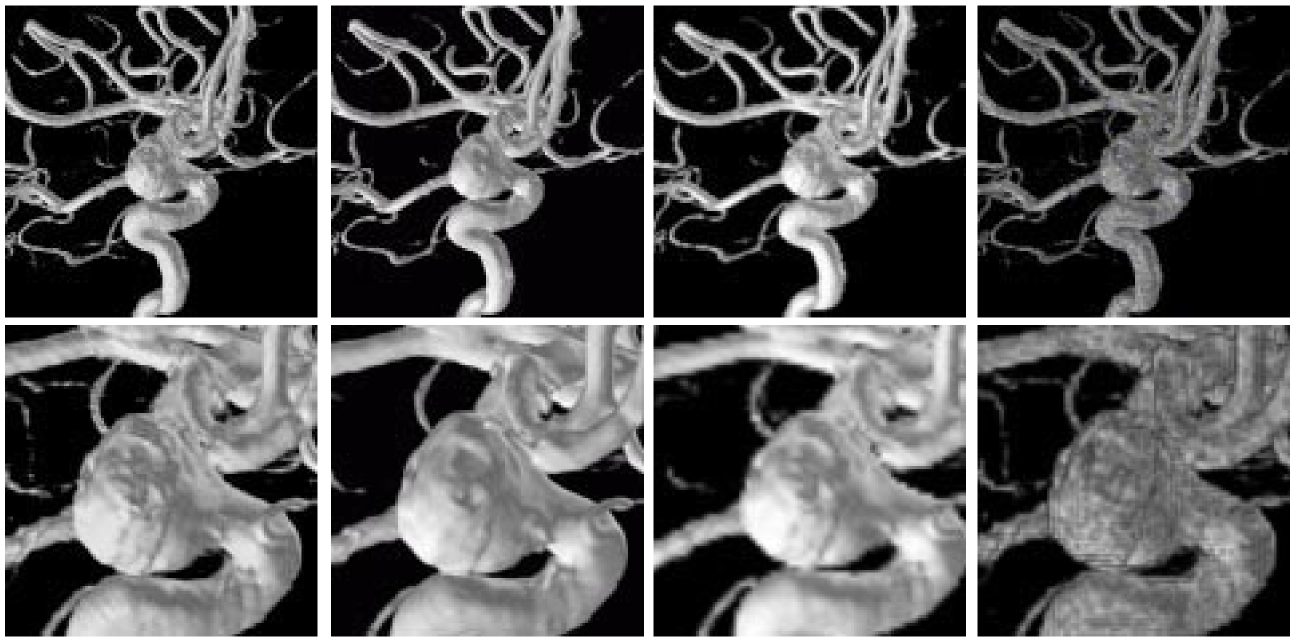


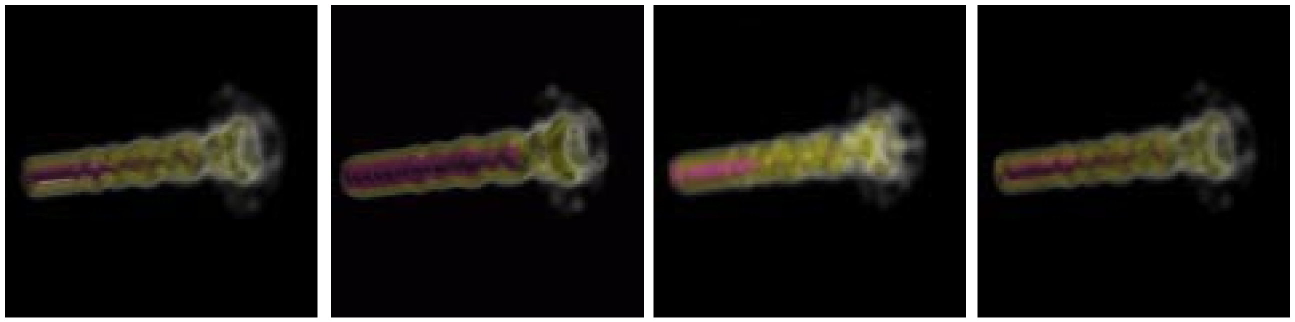
Figure 3

MRI Head (Blood Vessel)



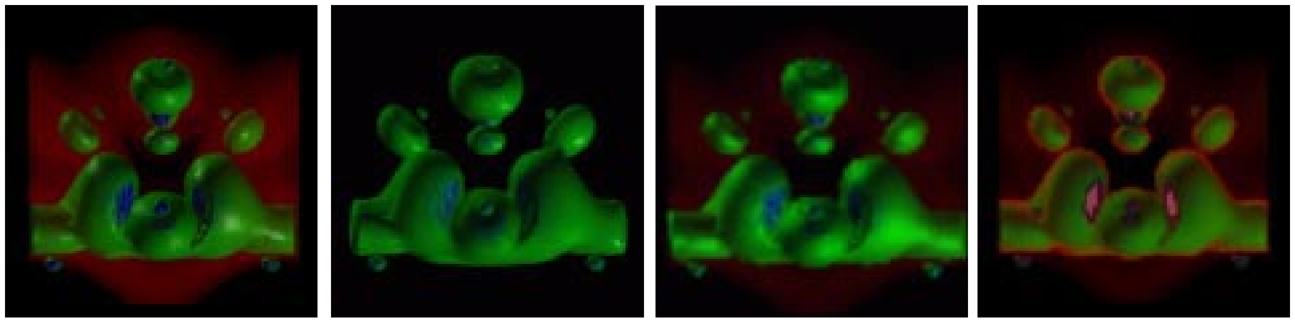
Raycasting **Splatting** **Shear-Warp** **Texture Mapping**

Fuel Injection

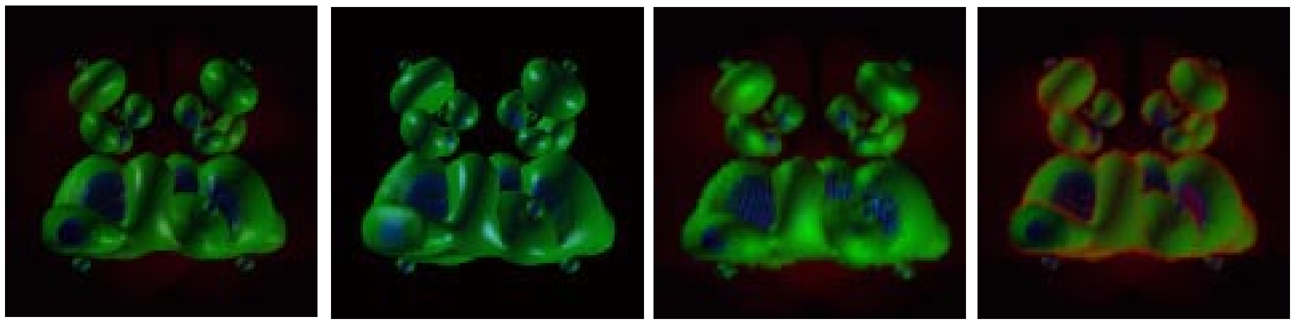


(f) Magnification: 5

Neghip

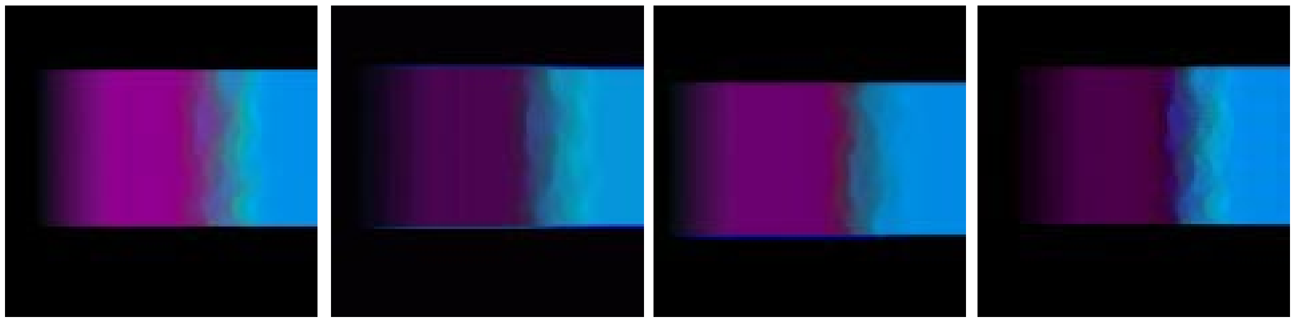


(g) Magnification: 5



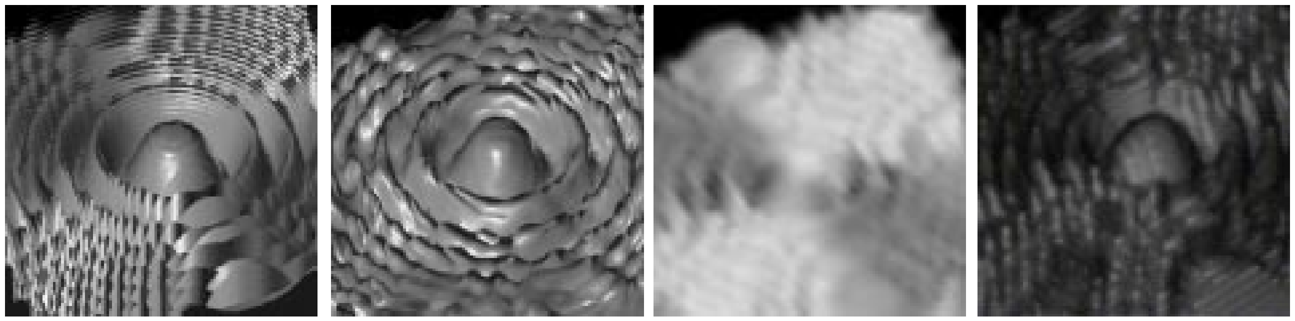
(h) Viewed at 45°

Shockwave



(i) Magnification: 1

Marschner-Lobb Function



(k) Magnification: 6